МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Кваліфікаційна наукова
праця на правах рукопису

РАДЧЕНКО ЄВГЕН ОЛЕКСАНДРОВИЧ

УДК 004.62 : 004.056.5 : 004.492.2

ДИСЕРТАЦІЯ
АЛГОРИТМІЧНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМ ЗАХИСТУ МУЛЬТИМЕДІЙНИХ ДАНИХ КОРИСТУВАЧІВ МЕРЕЖІ ІНТЕРНЕТ

Спеціальність 121 – Інженерія програмного забезпечення

Галузь знань 12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.
_____ Є. О. Радченко

Науковий керівник:
Сулема Євгенія Станіславівна
кандидат технічних наук, доцент

Київ – 2020

АНОТАЦІЯ

Радченко Є. О. Алгоритмічне та програмне забезпечення систем захисту мультимедійних даних користувачів мережі Інтернет. − Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення. – Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, 2020.

Мультимедійні дані складають значну частку даних, що передаються через мережу Інтернет та зберігаються у різноманітних хмарних сховищах. Частина цих мультимедійних даних є приватними даними користувачів (особисті фотографії, відео та аудіозаписи). Водночас, спостерігається тенденція збільшення кількості користувачів у соціальних мережах та месенджерах, в результаті чого переважна більшість користувачів мережі Інтернет має певну долю власних персональних даних на серверах різноманітних компаній. Ця персональна інформація є конфіденційною і не має розповсюджуватися без згоди власника. Проте, відомі випадки використання персональних мультимедійних даних, опублікованих у приватних бесідах соціальних мереж та месенджерах, з метою таргетування рекламних оголошень. Отже, зберігаючи дані поза локальним комп’ютером, користувач втрачає повний контроль за доступом до цих даних. Проблема передачі конфіденційних мультимедійних даних існує і у телемедицині, де передбачене використання комп’ютерних та телекомунікаційних технологій для обміну медичною інформацією у вигляді зображень та відео (наприклад, рентгенівські знімки, відеозаписи ендоскопічного обстеження тощо). При передачі конфіденційних даних пацієнта через мережу Інтернет існує ймовірність витоку інформації, а передача медичних даних у відкритому вигляді є неприпустимою з правової точки зору.

Для захисту конфіденційних даних користувачів мережі Інтернет можуть використовуватись різноманітні методи, включаючи методи стеганографічного захисту. Перевагою стеганографічного захисту є те, що він дозволяє приховати сам факт існування конфіденційної інформації, що дозволяє використовувати стеганографічні методи для забезпечення конфіденційності у звичайних месенджерах, соціальних мережах, тощо. Для підвищення рівня захисту мультимедійних даних доцільно використовувати крипто-стеганографічні методи, які передбачають попереднє шифрування даних перед їх стеганографічним вбудовуванням.

Під час реалізації методів захисту мультимедійних даних у вигляді програмного продукту необхідно враховувати не лише рівень захисту даних, а також інші вимоги, виконання яких є важливим для зручної роботи користувача. Основною з цих вимог є висока швидкість оброблення користувацьких даних. Аналіз існуючого програмного забезпечення свідчить про те, що на сьогодні спостерігається потреба у розроблені нових програмних продуктів, які дозволятимуть користувачам мережі Інтернет забезпечувати конфіденційність своїх мультимедійних даних. При цьому необхідно забезпечити зручність та простоту розроблення таких програмних продуктів для різних галузей застосування та різних груп користувачів. З урахуванням цих особливостей та потреб постає актуальна **науково-практична проблемна задача** розроблення універсальної архітектури програмної системи захисту мультимедійних даних, а також програмних компонентів, що реалізують нові крипто-стеганографічні методи захисту мультимедійних даних користувачів.

Метою дисертаційної роботи є підвищення ефективності програмних засобів крипто-стеганографічного захисту мультимедійних даних користувачів мережі Інтернет за рахунок вдосконалення архітектури програмної системи захисту та розроблення нових алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних, що забезпечують надійність

захисту та підвищення швидкодії процедур оброблення мультимедійних даних користувача.

У першому розділі дисертаційної роботи здійснено комплексний аналіз методів та програмних засобів стеганографічного захисту персональних мультимедійних даних у мережі Інтернет. Проведений аналіз показав необхідність розроблення нових програмних рішень для захисту мультимедійних даних користувачів мережі Інтернет та дозволив сформулювати вимоги до програмного забезпечення процесів захисту мультимедійних даних.

У другому розділі запропоновано алгоритмічне забезпечення крипто-стеганографічного захисту мультимедійних даних, а саме, розроблено три методи крипто-стеганографічного захисту: метод на основі схеми відповідності бітів, метод на основі дерева Хаффмана та метод на основі псевдовипадкового вбудовування.

У третьому розділі розроблено універсальну архітектуру програмної системи крипто-стеганографічного захисту мультимедійних даних; запропоновану архітектуру реалізовано у вигляді програмного забезпечення з використанням методу на основі схеми відповідності бітів, методу на основі дерева Хаффмана та методу на основі псевдовипадкового вбудовування.

У четвертому розділі досліджено запропоновану архітектуру, зокрема, здійснено порівняльний аналіз розроблених алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних за часом виконання, стеганографічною стійкістю та стійкістю до статистичних атак.

У дисертаційній роботі отримано ряд **нових наукових результатів**, зокрема, **уперше** запропоновано універсальну архітектуру програмної системи захисту мультимедійних даних користувачів мережі Інтернет, використання якої дозволяє спростити процес розроблення програмного забезпечення систем захисту мультимедійних даних та яка, на відміну від існуючих, забезпечує

можливість використання довільних методів крипто-стеганографічного захисту даних і отримання мультимедійних даних з різних програмних середовищ.

Уперше запропоновано алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, характерною рисою якого є можливість поєднання з іншими методами LSB-стеганографії для підвищення їх стеганографічної стійкості та який, на відміну від відомих, ґрунтується на застосуванні процедури шифрування на основі схеми відповідності бітів і логічної функції, що дозволяє забезпечити захист мультимедійних даних та підвищити швидкодію процедури вбудовування стегоданих у понад 5 разів.

Уперше розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, який, на відміну від відомих, ґрунтується на використанні процедури побудови дерева Хаффмана на основі зображення-ключа, що дозволяє підвищити рівень захисту конфіденційних графічних даних від підбору ключів та статистичних стегоатак у середньому у 17,4 разів.

Уперше розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, визначальною рисою якого є застосування процедури псевдовипадкового вбудовування даних із застосуванням двох генераторів псевдовипадкових чисел, та який характеризується високою стійкістю до підбору ключів за рахунок зростання ентропійних характеристик та змінної кількості ключів, що дозволяє забезпечити захист мультимедійних даних.

За матеріалами дисертації опубліковано 6 наукових праць, зокрема, 3 наукових статті, з яких 2 статті опубліковано у закордонних фахових виданнях третього квартиля (Q3), які реферуються базою Scopus, та 1 стаття опублікована у науковому виданні, що входить до наукових фахових видань України, і 3 публікації у матеріалах науково-технічних конференцій.

Ключові слова: прикладне програмне забезпечення, програмна система, архітектура програмного забезпечення, захист мультимедійних даних, крипто-стеганографічні методи.

SUMMARY

Radchenko Y. Algorithmic support and software of Internet user's multimedia data protection systems. − Qualifying scientific work, the manuscript.

PhD thesis in the field of knowledge 12 Information technologies in a specialty 121 Software engineering. – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, 2020.

Multimedia data make up a significant proportion of data transmitted over the Internet and stored in various cloud storage. Some of this multimedia data is private user data (personal photos, videos and audio recordings). At the same time, there is a tendency to increase the number of users in social networks and messengers, as a result the vast majority of Internet users have some of their own personal data on the servers of various companies. This personal information is confidential and should not be disclosed without the consent of the owner. However, there are known cases of using personal multimedia data published in private conversations in social networks and messengers to target advertisements. Therefore, by storing data outside the local computer, the user loses complete control over access to this data. The problem of confidential multimedia data transmission also exists in telemedicine, where computer and telecommunication technologies are used to exchange medical information in the form of images and videos (for example, X-rays, endoscopic videos, etc.). When transmitting confidential patient data over the Internet, there is a possibility of information leakage, but the transfer of medical data in the open is legally unacceptable.

A variety of methods can be used to protect the confidential data of Internet users, including steganographic protection methods. The advantage of steganographic

protection is that it allows to hide the fact of the existence of confidential information, which allows to use steganographic methods to ensure confidentiality in regular messengers, social networks, and so on. To increase the level of multimedia data protection, it is advisable to use crypto-steganographic methods, which provide data pre-encryption before steganographic embedding.

When implementing methods of multimedia data protecting in the form of a software product, it is necessary to take into account not only the level of data protection, but also other requirements, the fulfillment of which is important for the user's convenience. The main of these requirements is the high speed of user data processing. Analysis of existing software shows that today there is a need to develop new software products that will allow Internet users to ensure the confidentiality of their multimedia data. It is necessary to ensure the convenience and simplicity of developing such software products for different applications and different user groups. Taking into account these features and needs, there is an urgent **scientific and practical problem** of developing an universal architecture of the software system for multimedia data protection, as well as software components that implement new crypto-steganographic methods for user`s multimedia data protection.

The purpose of the dissertation is to increase the efficiency of software for Internet users multimedia data crypto-steganographic protection by improving the architecture of the software protection system and developing new algorithmic and software methods for multimedia data crypto-steganographic protection.

In the first section of the dissertation a comprehensive analysis of methods and software for steganographic protection of personal multimedia data on the Internet is given. The analysis showed the need to develop new software solutions for the Internet users multimedia data protection and allowed to formulate software requirements for multimedia data protection processes.

The second section proposes algorithmic support for multimedia data crypto-steganographic protection, namely, developed three methods of crypto-steganographic protection: a method based on the bit correspondence scheme, a method based on the Huffman tree and a method based on pseudo-random embedding.

In the third section the universal architecture of the software system for multimedia data crypto-steganographic protection was developed; the proposed architecture is implemented in the form of software using a method based on the bit correspondence scheme, a method based on the Huffman tree and a method based on pseudo-random embedding.

The fourth section examines the proposed architecture, in particular, a comparative analysis of the developed algorithmic and software methods of multimedia data crypto-steganographic protection by embedding time, steganographic stability and resistance to statistical attacks.

The dissertation provides a number of new scientific results, in particular, the universal architecture of the software system for Internet user's multimedia data protection, which simplifies the process of software development of multimedia data protection systems and provides the ability to use arbitrary methods of crypto-steganographic data protection and provide multimedia data from different software environments, has been **proposed for the first time**.

An algorithmic software method of multimedia data crypto-steganographic protection, the characteristic feature of which is the possibility of combining with other methods of LSB-steganography to increase their steganographic stability and which, unlike the known ones, relies on the encryption procedure based on bit correspondence scheme and logic function that allows to protect multimedia data and increase the speed of embedding procedure in more than 5 times, has been **developed for the first time**.

An algorithmic software method of crypto-steganographic multimedia data protection which, unlike the known ones, is based on the use of the Huffman tree

procedure based on the key-image, which allows to increase the level of protection of confidential graphic data agains the key search in 17.4 times, has been **developed for the first time**.

An algorithmic software method of multimedia data crypto-steganographic protection, the defining feature of which is the application of pseudo-random data embedding procedure using two pseudo-random number generators, and which is characterized by high resistance to the key search due to increasing entropy characteristics and variable number of keys that enables multimedia data protection, has been **developed for the first time**.

Based on the dissertation, 6 scientific works **were published**, in particular, 3 scientific articles, 2 articles of which were published in foreign professional publications of the third quartile (Q3), which are referenced by Scopus, and 1 article was published in a scientific publication included in scientific professional publications of Ukraine, and 3 publications were published in the materials of scientific and technical conferences.

Keywords: application software, software system, software architecture, multimedia data protection, crypto-steganographic methods.

Список публікацій здобувача / List of publications of the applicant:

- статті в закордонних фахових виданнях третього квартиля (Q3), які реферуються базою Scopus / articles in foreign professional journals of the third quartile (Q3) which are referenced by the Scopus (2):

1. Radchenko Ye., Dychka I., Sulema Ye., Suschuk-Sliusarenko V., Shkurat O. Steganographic Protection Method Based on Huffman Tree. Advances in Intelligent Systems and Computing. Springer Verlag, Germany, 2019. Vol. 902, P. 283–292. ISSN : 21945357.

2. Hu Zh., Dychka I., Sulema Ye., Radchenko Ye. Graphical Data Steganographic Protection Method Based on Bits Correspondence Scheme. International Journal of Intelligent Systems and Applications (IJISA). China, 2017. Vol. 9. No. 8, P. 34–40. ISSN : 20749058.

- стаття в науковому фаховому журналі України / article in Ukrainian professional journal (1):

3. Сулема Є.С., Радченко Є.О. Метод стеганографічного захисту мультимедійних даних на основі процедури псевдовипадкового вбудовування. Наукові вісті КПІ, 2020. № 1, С. 40–47.

- матеріали науково-технічних конференцій / materials of the scientific conferences (3):

4. Радченко Є.О., Сулема Є.С. Спосіб стеганографічного захисту графічних даних на основі схеми відповідності бітів та аналізу візуальних властивостей контейнера. Матеріали доповідей Шостої Міжнародної науково-практичної конференції з сучасних проблем кодування, захисту й ущільнення інформації. Вінниця, Україна, 2017. С. 51–53.

5. Радченко Є. О., Сулема Є. С. Метод стеганографічного захисту WAV-файлів. Збірник тез доповідей 12-ї наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019). НТУУ «КПІ». 2019. С. 99–104.

6. Sulema Ye. S., Radchenko Ye. O. Algorithm of graphical data stegonagraphic protection based on bits difference transform. Збірник тез доповідей 8-ї наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2016). НТУУ «КПІ». 2016. С. 254–258.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES – Advanced Encryption Standard (симетричний алгоритм блочного шифрування).

BMP – Bitmap Picture (формат файлу зображень растрової графіки).

DES – Data Encryption Standard (алгоритм для симетричного шифрування).

E2EE – End-To-End Encryption (наскрізне шифрування).

ECDH – Elliptic curve Diffie–Hellman (протокол Діффі-Хеллмана на еліптичних кривих).

FLAC – Free Lossless Audio Codec (вільний аудіокодек без втрат).

GIF – Graphics Interchange Format (формат обміну зображеннями).

GUI – Graphical User Interface (графічний інтерфейс користувача).

HMAC – Hash-based Message Authentication Code (хеш-код автентифікації повідомлень).

HTML – HyperText Markup Language (мова розмітки гіпертексту).

IP – Internet Protocol (Інтернет протокол).

JPEG – Joint Photographic Experts Group (формат файлу зображень растрової графіки).

LSB – Least Significant Bit (найменш значущий біт).

MPEG – Moving Picture Experts Group (експертна група з питань рухомого зображення).

MQTT – MQ Telemetry Transport (мережевий протокол).

PCM – Pulse Code Modulation (імпульсно-кодова модуляція).

PNM – Portable Anymap Format (формат файлу зображень растрової графіки).

PNG – Portable Network Graphics (формат файлу зображень растрової графіки).

QML – Qt Meta Language (декларативна мова програмування).

RGBA – Red-Green-Blue-Alpha color model (червоний-зелений-синій-альфа, колірна модель).

RS – Regular-Singular method (метод статистичного аналізу).

SHA – Secure Hash Algorithm (алгоритм криптографічного хешування).

SSL – Secure Sockets Layer (рівень захищених сокетів).

TCP – Transmission Control Protocol (протокол керування передачею).

TLS – Transport Layer Security (захист на транспортному рівні).

UDP – User Datagram Protocol (протокол датаграм користувача).

WAV – Waveform audio format (формат аудіофайлів).

XML – eXtensible Markup Language (розширювана мова розмітки).

XMPP – eXtensible Messaging and Presence Protocol (розширюваний протокол обміну повідомленнями про присутність).

ГПВЧ – генератор псевдовипадкових чисел.

ДКП – дискретне косинусне перетворення.

ООП – об’єктно-орієнтоване програмування.

ОС – операційна система.

ВСТУП

Актуальність теми. Мультимедійні дані складають значну частку даних, що передаються через мережу Інтернет та зберігаються у різноманітних хмарних сховищах. Частина цих мультимедійних даних є приватними даними користувачів (особисті фотографії, відео та аудіо-записи). Водночас, спостерігається тенденція збільшення кількості користувачів у соціальних мережах та месенджерах, в результаті чого переважна більшість користувачів мережі Інтернет має певну долю власних персональних даних, в тому числі, мультимедійних, на серверах різноманітних компаній. Ця персональна інформація є конфіденційною і не має розповсюджуватися без згоди власника. Проте, відомі випадки використання персональних мультимедійних даних у приватних бесідах соціальних мереж та месенджерах з метою таргетування рекламних оголошень. Отже, зберігаючи дані поза локальним комп'ютером, користувач втрачає повний контроль за доступом до цих даних.

Проблема передачі конфіденційних даних існує і у телемедицині, де передбачене використання комп'ютерних та телекомунікаційних технологій для обміну медичною інформацією. При передачі конфіденційних даних пацієнта через мережу Інтернет існує ймовірність витоку інформації, а передача медичних даних у відкритому вигляді є неприпустимою з правової точки зору.

Для захисту конфіденційних даних користувачів мережі Інтернет можуть використовуватись різноманітні методи, включаючи методи стеганографічного захисту. Перевагою стеганографічного захисту є те, що він дозволяє приховати сам факт існування конфіденційної інформації, що дозволяє використовувати стеганографічні методи для забезпечення конфіденційності у звичайних месенджерах, у соціальних мережах, тощо. Для підвищення рівня захисту мультимедійних даних доцільно використовувати крипто-стеганографічні методи, які передбачають попереднє шифрування даних перед їх стеганографічним вбудовуванням.

Під час реалізації методів захисту мультимедійних даних у вигляді програмного продукту необхідно враховувати не лише рівень захисту даних, а також інші вимоги, виконання яких є важливим для зручної роботи користувача. Основною з цих вимог є висока швидкість оброблення користувацьких даних. Аналіз існуючого програмного забезпечення свідчить про те, що на сьогодні спостерігається потреба у розроблені нових програмних продуктів, які дозволятимуть користувачам мережі Інтернет забезпечувати конфіденційність своїх мультимедійних даних та задовольнятимуть іншим функціональним та нефункціональним вимогам до програмного забезпечення. При цьому необхідно забезпечити зручність та простоту розроблення таких програмних продуктів для різних галузей застосування та різних груп користувачів. З урахуванням цих особливостей та потреб постає актуальна науково-практична проблемна задача розроблення універсальної архітектури програмної системи захисту мультимедійних даних, а також програмних компонентів, що реалізують нові крипто-стеганографічні методи захисту мультимедійних даних користувачів.

Зв'язок роботи з науковими програмами, планами, темами. Дослідження за темою дисертаційної роботи провадилось у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» в рамках виконання держбюджетної науково-дослідної роботи «Розроблення та дослідження методів оброблення, розпізнавання, захисту та зберігання медичних зображень в розподілених комп'ютерних системах» (номер державної реєстрації 0117U004267).

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення ефективності програмних засобів крипто-стеганографічного захисту мультимедійних даних користувачів мережі Інтернет за рахунок вдосконалення архітектури програмної системи захисту та розроблення нових алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних, що забезпечують надійність захисту та підвищення швидкодії процедур оброблення мультимедійних даних користувача.

Відповідно до поставленої мети основними задачами дослідження є:

- аналіз вимог до програмних систем захисту мультимедійних даних;
- розроблення універсальної архітектури програмної системи захисту мультимедійних даних, яка призначена для спрощення процесу розроблення програмного забезпечення захисту мультимедійних даних користувачів мережі Інтернет;
- аналіз методів крипто-стеганографічного захисту даних з точки зору особливостей їх програмної реалізації;
- розроблення ефективних алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних, які враховують надлишковість мультимедійних даних для маскування конфіденційних стегоданих при їх передачі у мережі Інтернет;
- дослідження ефективності розроблених алгоритмічно-програмних методів, включаючи експериментальне дослідження швидкодії роботи, ймовірності підбору ключів та статистичного стегоаналізу відповідних методів захисту даних;
- розроблення та дослідження експериментальної програмної системи захисту мультимедійних даних.

Об’єкт дослідження – процеси розроблення програмних систем захисту інформації.

Предмет дослідження – методи розроблення програмного забезпечення систем захисту мультимедійних даних користувачів мережі Інтернет.

Методи дослідження: теорія алгоритмів, теорія програмування, теорія програмних систем, методи захисту даних.

Наукова новизна одержаних результатів полягає у наступному:

1. **Уперше** запропоновано універсальну архітектуру програмної системи захисту мультимедійних даних користувачів мережі Інтернет,

використання якої дозволяє спростити процес розроблення програмного забезпечення систем захисту мультимедійних даних та яка, на відміну від відомих, забезпечує можливість використання довільних методів крипто-стеганографічного захисту даних і отримання мультимедійних даних з різних програмних середовищ.

2. **Уперше** запропоновано алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, характерною рисою якого є можливість поєднання з іншими методами LSB-стеганографії для підвищення їх стеганографічної стійкості та який, на відміну від відомих, ґрунтується на застосуванні процедури шифрування на основі схеми відповідності бітів і логічної функції, що дозволяє забезпечити захист мультимедійних даних та підвищити швидкодію процедури вбудовування стегоданих у понад 5 разів.
3. **Уперше** розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, який, на відміну від відомих, ґрунтується на використанні процедури побудови дерева Хаффмана на основі зображення-ключа, що дозволяє підвищити рівень захисту конфіденційних графічних даних від підбору ключів та статистичних стегоатак у середньому у 17,4 разів.
4. **Уперше** розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, визначальною рисою якого є застосування процедури псевдовипадкового вбудовування даних із застосуванням двох генераторів псевдовипадкових чисел, та який характеризується високою стійкістю до підбору ключів за рахунок зростання ентропійних характеристик та змінної кількості ключів, що дозволяє забезпечити захист мультимедійних даних.

Практичне значення одержаних результатів полягає у спрощенні процесу розроблення нових програмних продуктів та сервісів захисту

мультимедійних даних користувачів мережі Інтернет на основі алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних.

Запропоновані алгоритмічно-програмні методи захисту мультимедійних даних застосовані при виконанні науково-дослідної роботи «Розроблення та дослідження методів обробки, розпізнавання, захисту та зберігання медичних зображень в розподілених комп'ютерних системах» (номер держреєстрації 0117U004267) та апробовані у ТОВ «Відео Інтернет Технології» для подальшого застосування при розроблені програмного забезпечення.

Особистий внесок здобувача. Всі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто. У публікаціях, написаних у співавторстві, здобувачеві належать такі результати. У роботі [1] здобувачем запропоновано метод захисту графічних даних на основі дерева Хаффмана. У роботі [2] здобувачем запропоновано метод на основі схеми відповідності бітів. У роботі [3] здобувачем запропоновано метод на основі процедури псевдовипадкового вбудовування. У роботі [4] здобувачем запропоновано схему відповідності бітів. У роботі [5] здобувачем запропоновано програмну процедуру захисту аудіо-даних. У роботі [6] здобувачем запропоновано програмну процедуру шифрування даних.

Апробація результатів дисертації. Основні результати дисертаційного дослідження доповідалися та обговорювалися на міжнародних та національних наукових та науково-практичних конференціях:

1. VI Міжнародна науково-практична конференція "Методи та засоби кодування, захисту й ущільнення інформації", Вінниця, 2017.
2. XII Наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2019)», Київ, 2019.
3. VIII Наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2016)», Київ, 2016.

Публікації. Основні результати дисертаційної роботи опубліковані у 6 наукових працях, зокрема, у 3 наукових статтях, з яких 2 статті опубліковано у

закордонних фахових виданнях третього квартиля (Q3), які реферуються базою Scopus, та 1 стаття опублікована у науковому виданні, що входить до наукових фахових видань України, і 3 публікації у матеріалах науково-технічних конференцій.

Структура роботи. Дисертаційна робота складається з вступу, чотирьох розділів, висновків, п`яти додатків та списку літератури, що включає 151 найменування. Загальний обсяг роботи становить 188 сторінок, у тому числі 140 сторінок основного тексту, 22 рисунки, 23 таблиці.

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ ЗАХИСТУ ПЕРСОНАЛЬНИХ МУЛЬТИМЕДІЙНИХ ДАНИХ В МЕРЕЖІ ІНТЕРНЕТ

1.1 Аналіз предметної галузі та вимог до програмного забезпечення захисту мультимедійних даних користувачів мережі Інтернет

У наш час мультимедійні дані використовуються майже у кожній галузі. Їх популярність пояснюється зручністю сприйняття людиною, що характеризується швидкістю обробки отриманої інформації. Адже завдяки таким властивостям як колір, підсвічування, розмір об’єкту тощо можна виділити найбільш значущі місця у графічних даних.

Більшість користувачів мережі Інтернет обмінюються між собою мультимедійними даними за допомогою месенджерів, електронної пошти та соціальних мереж. Але не завжди ці дані захищені належним чином.

Загалом засоби захисту інформації можна поділити на наступні групи:

1. Апаратні або технічні. Вирішують проблему захисту інформації за допомогою фізичних апаратних пристроїв [121]. До таких пристроїв відносяться генератори шумів, мережеві фільтри тощо [42, 3].
2. Програмні засоби. До даної групи засобів належать програмні комплекси для ідентифікації користувачів, контролю доступу до певних даних, резервного копіювання та віддаленого зберігання найбільш важливих масивів даних, захисту від несанкціонованого доступу.
3. Змішані апаратні та програмні засоби.
4. Організаційні та організаційно-правові. До даної групи відносяться засоби забезпечення приміщень з урахуванням вимог безпеки, прокладка кабельної системи з урахуванням вимог обмеження доступу та ін.

Середньостатистичний користувач мережі Інтернет часто не має можливості використовувати апаратні засоби захисту інформації, оскільки це

потребує певних фінансових витрат та професійних знань. Тому, на даний час найбільш доступними та популярними є саме програмні засоби.

Найчастіше для програмного захисту даних застосовують методи криптографії. Криптографія у сучасному її розумінні – область наукових, прикладних, інженерно-технічних досліджень, заснована на фундаментальних поняттях математики, фізики, теорії інформації та складних обчислень [29]. У класичній моделі системи секретного зв'язку є два учасники, які повністю довіряють один одному та в яких існує необхідність обміну даними, що не призначені для третіх сторін. Такий тип інформації називається конфіденційним. Отже, першочерговою задачею криптографії є захист конфіденційних даних від сторонніх осіб. Ще однією важливою задачею є забезпечення цілісності та непідробності повідомлення. Адже при відправленні секретного повідомлення воно може бути перехоплене та відредаговане.

Розробити надійний алгоритм шифрування даних, що не використовує для кодування та декодування нічого, крім секретного повідомлення, є надскладною задачею [50]. Тому переважна більшість існуючих криптографічних алгоритмів використовують ключі.

У класичному випадку ключ – це послідовність чисел певної довжини, що була створена за певними правилами на основі псевдовипадкових чисел. Використовуючи ключ, відправник та отримувач секретного повідомлення можуть кодувати і декодувати відповідно вміст цього повідомлення. Якщо для обох цих операцій використовується один і той самий ключ, такий підхід називають симетричною криптографією [52]. Часто симетричні алгоритми є досить простими та швидкими, але недостатньо надійними для певних задач. Тому, на даний момент, частіше використовується асиметрична криптографія, яка передбачає, що для кодування та декодування конфіденційної інформації використовуються різні ключі. Існує також гібридна криптографія. У даному випадку конфіденційне повідомлення шифрується симетричним ключем, а сам ключ додатково шифрується асиметричним алгоритмом [28]. Даний процес

складається з двох кроків, але при великих об’ємах даних виграє у асиметричних алгоритмів з точки зору швидкодії, оскільки ключ, що шифрується асиметрично у гібридних криптографічних методах часто має відносно малий розмір.

Криптографічні алгоритми часто використовуються у мережі Інтернет у сучасних клієнтах систем миттєвого обміну повідомленнями, які називаються месенджерами. Месенджери часто використовують свій центральний сервер, на якому зберігається вся історія листування кожного з користувачів. Таким чином, особи, які мають доступ до даних сервера, можуть вивчити вміст листування. Таким чином виникає необхідність у повній конфіденційності бесіди і в цьому випадку доводиться використовувати наскрізне шифрування (End-to-End Encryption, E2EE [74]) – метод передачі даних, коли доступ до повідомлень мають виключно користувачі, задіяні у спілкуванні. Розглянемо принципи роботи і шифрування даних у найпопулярніших месенджерах.

Facebook Messenger – офіційний месенджер, розроблений як складова найбільшої в світі соціальної мережі Facebook. Додаток побудовано на базі відкритого протоколу MQTT [91]. Цей месенджер дозволяє обмінюватися файлами і фотографіями. Мобільна версія месенджера надає голосовий та відеозв'язок, а також можливість передавати геомітки [19]. Протокол MQTT (MQ Telemetry Transport) – це полегшений мережевий протокол публікації-підписки, який передає повідомлення між пристроями. Протокол часто працює за мережевим протоколом TCP/IP [76], однак будь-який мережевий протокол, який забезпечує впорядковані двонаправленні з’єднання без втрат, може підтримувати MQTT [83]. Facebook підтримує також і наскрізне (End-to-End) шифрування з секретними чатами. Для цього створюється секретний діалог, вміст якого є доступним виключно для створювача діалогу і його співрозмовника – користувача, запрошеного в секретний чат. Наскрізне шифрування в секретних чатах засноване на базі протоколу Signal. Протокол Signal [68] (раніше відомий як TextSecure Protocol) – криптографічний протокол для забезпечення наскрізного шифрування голосових викликів, відеодзвінків і миттєвих

повідомлень. Протокол Signal був створений для роботи в асинхронному режимі, коли одна зі сторін може бути офлайн деякий невизначений проміжок часу.

WhatsApp – безкоштовний мультиплатформовий месенджер з підтримкою голосового зв'язку. Дозволяє пересилати текстові повідомлення, зображення, відео та аудіо. Також існують версії для комп'ютерів і мобільних пристроїв під управлінням різних операційних систем [53]. WhatsApp використовує модифікований протокол Extensible Messaging and Presence Protocol (XMPP, раніше відомий як Jabber). XMPP [105] (eXtensible Messaging and Presence Protocol) – розширюваний протокол для обміну текстовими повідомленнями, відео, голосовими повідомленнями і файлами у мережі та заснований на XML [4]. Завдяки даному протоколу також можна обмінюватись інформацією про статус користувача у даний час (online/offline). Технології, засновані на протоколі XMPP, розгорнуті на децентралізованій клієнт-серверній архітектурі з необмеженою кількістю вузлів [106]. WhatsApp також підтримує наскрізне шифрування (End-to-End) на базі протоколу Signal. Шифрування може використовуватись у групових чатах та поширюється на всі типи повідомлень: текст, фото, відео і голосові повідомлення. У реалізації використовуються алгоритми ECDH на Curve25519 [137], AES-256 [71], AES-GCM [94], HMAC-SHA256. Протокол Діффі-Хеллмана [122] на еліптичних кривих – криптографічний протокол, що дозволяє обом користувачам отримати загальний секретний ключ використовуючи відкриті (не захищені) канали зв'язку маючи пару відкритий-закритий ключ. Для обчислення загального ключа користувачі повинні попередньо узгодити параметри використовуваної еліптичної кривої [33]. Curve25519 також є еліптичною кривою, яка є одним з варіантів протоколу Діффі-Хеллмана. З цієї причини Curve25519 може бути реалізований за допомогою схеми узгодження ключів по Діффі-Хеллману (ECDH) для еліптичної кривої. Ця властивість дозволяє Curve25519 обчислювати загальні ключі, якими можна обмінюватись і по незашифрованому каналу. У кожного

учасника в WhatsApp є довгостроковий ідентифікаційний ключ, який використовується для обчислення цього загального секретного ключа.

У наш час існує безліч криптографічних алгоритмів: AES [135], Serpent [65], Twofish [149], Blowfish [95], 3DES, CAST5 [114]). Розширений стандарт шифрування (AES) визначає криптографічний алгоритм, який можна використовувати для захисту електронних даних [61]. Алгоритм AES має симетричний блоковий шифр (розмір блоку 128 біт, ключ 128/192/256 біт), який може шифрувати і дешифрувати інформацію. Шифрування перетворює дані в незрозумілу форму (зашифрований текст).

HMAC (hash-based message authentication code) – в криптографії один з механізмів перевірки цілісності інформації, що дозволяє гарантувати те, що до даних, які передаються або зберігаються у ненадійному середовищі, не внесено жодних змін сторонніми особами. При розрахунку HMAC використовується криптографічна хеш-функція SHA-256.

Для одного з найпопулярніших месенджерів Telegram був створений протокол MTProto. Даний протокол – об'єднання декількох криптографічних протоколів та хеш-функцій (RSA-2048, DH-2048, AES, SHA-1, MD-5). MTProto допускає використання шифрування end-to-end з опціональною звіркою ключів. Протокол MTProto підтримує множинні транспортні режими, такі як HTTP, TCP, UDP [73]. Також була реалізована функція секретних чатів з наскрізним шифруванням (AES-256) відправлених повідомлень.

Ще одним популярним способом обміну мультимедійними даними є використання соціальних мереж – комунікаційних каналів, що охоплюють різноманітні сфери суспільства. Водночас, їх архітектура базується на централізації великої кількості інформації про користувачів, що створює багато ризиків для конфіденційності, через порушення, витоки та все більшу нормалізацію програм спостереження. Незважаючи на дані недоліки, невелика кількість постачальників соціальних мереж пропонують послуги наскрізного

шифрування (E2EE) своїм користувачам через складність використання та впровадження.

Більшість соціальних мереж, для захисту даних користувачів, обмежуються тільки впровадженням захисту каналів зв’язку, наприклад, вмикаючи протоколи SSL [98] та TLS [99]. Протоколи SSL та TLS – це криптографічні протоколи, які забезпечують автентифікацію та шифрування даних між серверами та програмами, що працюють через мережу (наприклад, клієнт, який підключається до веб-сервера). Це означає, що якщо ці протоколи правильно розгорнуті, можна відкрити канал зв’язку до довільної служби в Інтернеті та безпечно обмінюватись інформацією, знаючи, що дані не потраплять до чужих рук і будуть отримані неушкодженими [104]. Протокол SSL є попередником протоколу TLS. Протягом останніх років було випущено декілька нових версій цих протоколів для усунення вразливостей та підтримки більш сильних, більш захищених цифрових наборів та алгоритмів. Варто зазначити, що SSL та TLS просто посилаються на обмін певним повідомленням, який відбувається між клієнтом і сервером. Дана операція узгоджує загальний секрет та тип шифрування, який буде використовуватися [77].

Аби посилити захищеність персональних даних користувачів, провайдери соціальних мереж часто надають змогу використовувати двофакторну автентифікацію [41].

Двофакторна автентифікація – це додатковий рівень безпеки, який використовується для того, щоб переконатися, що люди, які намагаються отримати доступ до онлайн-акаунта, це ті, за кого вони себе видають [110]. Спочатку користувач вводить своє ім'я та пароль. Тоді, замість негайного отримання доступу, йому потрібно буде надати додаткову інформацію, наприклад, пін-код, відповідь на секретне питання, токен на смартфоні, скан сітківки ока, відбиток пальця тощо.

Проте це також не гарантує безпеку персональної інформації та мультимедійних файлів, якщо зловмиснику вдалося викрасти, наприклад,

телефон. Крім того, як правило більшість користувачів використовують один і той самий пароль до кількох Інтернет-ресурсів, у тому числі і акаунтів соціальних мереж, тому зловмиснику легко отримати доступ до всіх застосунків і порталів користувача.

Користувачі численних додатків і соціальних мереж, як показує досвід останніх років, мало захищені від витоків особистої інформації. Всі дані, як правило, зберігаються централізовано на серверах. Їх можуть зламати хакери і викласти інформацію в загальний доступ.

Головним недоліком використання хмарних сховищ є втрата користувачами контролю за доступом до своїх даних. Відомі випадки злому хакерами популярних хмарних сховищ та витоку приватної мультимедійної інформації у мережу Інтернет. Тому питання ефективного захисту персональних даних користувачів в хмарних сховищах є дуже актуальним [79].

Ще одним джерелом ризиків для мультимедійних даних користувача у мережі Інтернет є таргетована реклама.

Таргетування – інструмент Інтернет-маркетингу, що надає можливість встановити певні умови демонстрації рекламного оголошення [9]. У перекладі з англійської мови, термін «target» означає «ціль», «мету». Виходячи з цього, йдеться про вид реклами, що налаштовується на цільову аудиторію. Вона використовується у різноманітних популярних соціальних та медійних мережах. На даний момент, таргетована реклама надає багато можливостей для бізнесу. У першу чергу, такий вид реклами ефективний для швидкого просування своїх послуг та товарів у мережі Інтернеті, зокрема: онлайн-магазинів, веб-сайтів, одиничних товарів. Таргетована реклама корисна для популяризації бренду, оскільки чим частіше він буде з’являтися в мережі, тим більше користувачів дізнається про нього. Отже, таргетована реклама виконує наступні функції:

- швидкий набір аудиторії на акаунті/сторінці/групі;
- збільшення кількості трафіка на зовнішні ресурси (сайти, інтернет-магазини);

- допомагає покупцям краще розумітися на товарі, бути більш інформованими, що підвищує ефективність продажу в офлайн-точках.

Соціальну мережу, що надає можливість збирати та оброблювати інформацію про користувачів з метою таргетування називають агентом. Для запуску таргетованої реклами визначають цільову аудиторію та створюють “персональний портрет” користувача за допомогою опублікованих даних на його сторінці. Таким чином формується база інтересів та рекламних оголошень, які з великою ймовірністю можуть зацікавити поточного користувача. Більшість сучасних алгоритмів надають можливість розпізнавати та аналізувати вміст фотографій та використовувати отриману інформацію з метою додавання користувачів до певної групи. Наступним етапом є аналіз реакції користувачів на рекламні матеріали за допомогою алгоритмів штучного інтелекту з метою визначення користувачів, які з більшою ймовірністю стануть клієнтами [48]. За офіційними даними агенти збирають інформацію лише за допомогою вподобань («лайків») та фотографій користувача та його друзів у соціальній мережі. Але за неофіційними даними агенти також використовують інформацію з персонального листування, фото, відео чи посилань, які були відправлені або прийняті у бесіді з співрозмовником. Користувачами неодноразово проводились експерименти, у яких створювався новий акаунт в агенті без історії та навмисно у персональному листуванні з іншим акаунтом згадувався той чи інший факт. Після цього в акаунті відображались рекламні оголошення, напряму пов’язані зі згаданим фактом, що підтверджує наявність витоку інформації, в тому числі мультимедійного характеру.

Отже, можна зробити висновок, що за офіційними та неофіційними даними, для формування бази використовуються наступні способи збору інформації:

- пошукові запити у мережі Інтернету;

- активність на певних веб-сторінках (час перебування, кількість її відвідування, тематика сайту, довготривале утримання курсору миші на певному матеріалі).
- вподобанні або вподобанні друзями пости;
- ігри у соціальних мережах;
- покупки на сайтах;
- камера на смартфонах;
- імена і типи файлів на пристрої;
- персональні листування;
- телефоні розмови;
- розмови поруч з електронними пристроями з встановленим агентом;
- дані про місце перебування.

Враховуючи факт, що переважна більшість користувачів Інтернет зареєстровані у соціальних мережах, використовують месенджери для листування, мають поштові скриньки тощо, виникає проблема ефективного захисту персональних мультимедійних даних у мережі Інтернет.

Проведений аналіз засобів захисту даних користувачів Інтернет дозволяє зробити висновок про те, що існуючі рішення, які використовуються у соціальних мережах, є недостатніми для забезпечення повного захисту даних та файлів користувачів. При цьому потрібно зазначити, що практично кожен користувач мережі Інтернет має певну частину своїх даних у хмарному сховищі. Серед широко вживаних сервісів до хмарних сховищ відносяться:

- поштові скриньки;
- соціальні мережі;
- сховища файлів;
- сервіси загального доступу і обробки даних.

Але не завжди існує необхідність використовувати саме криптографічні методи. Іноді достатнім є забезпечення непомітності передачі даних. Дану задачу

вирішують методи стеганографії, що за продуктивністю не поступаються методам легковагової криптографії (Light Weight Cryptography) [57, 87].

Стеганографія – наука про захист секретної інформації, при якому приховується сам факт існування секретної інформації [15, 16].

Задачі, що вирішує комп'ютерна стеганографія [64, 123]:

- непомітна передача інформації;
- приховане зберігання інформації;
- захист авторського права [127];
- захист справжності документів;
- індивідуальний відбиток у системах електронного документообігу.

Для передачі секретних даних отримувачу, їх попередньо вбудовують у контейнер. Під контейнером у даному випадку слід розуміти цифрові дані, використання надлишковості яких дозволяє передавати допоміжну інформацію, не виявивши факт передачі [14]. Заповнений контейнер називають стего, а секретне повідомлення – стегоданими.

Стеганографічна стійкість контейнера визначається ймовірними діями або атаками на заповнений контейнер з ціллю виявлення або видалення, редагування стегоданих у випадку, якщо наявність даних не є секретом для стегоаналітика (особи, що здійснює стегоатаки на контейнер) [109].

Методи цифрової стеганографії [66, 112, 132, 133] можна розділити на шість основних груп [89]:

1. Методи, що працюють у просторовій області. Конфіденційні дані вбудовуються у значення пікселя. Прикладами такого підходу є LSB (Least Significant Bit), PVD (Pixel Value Differencing), BPC (Binary Pattern Complexity).

2. Методи, що приховують дані у частотній області. Більшість цих методів ґрунтується на дискретному вейвлет-перетворенні, дискретному косинусному перетворенні (ДКП) [17] та дискретному перетворенні Фур'є.

Дані алгоритми використовуються при ущільненні (JPEG [20] – ДКП, JPEG-2000 – вейвлет-перетворення) [81]. Таким чином, рекомендується обирати той стеганографічний алгоритм, завдяки якому буде проводитися ущільнення [51].

3. Метод векторного вбудовування. Даний метод вбудовує аудіодані у пікселі кадрів відеоконтейнера. Він заснований на стандарті кодування відео H.264/AVC.
4. Статистичні методи. Конфіденційні дані вбудовуються шляхом зміни декількох властивостей контейнера. Контейнер ділиться на блоки, а далі у кожен блок вбудовується лише один біт конфіденційної інформації.
5. Методи спотворень. Дана група методів використовується для вбудовування секретних даних шляхом спотворення сигналу.
6. Методи маскування та фільтрації. У таких методах секретне повідомлення накладають на оригінальний контейнер (наприклад цифрові водяні знаки [2, 69, 118]).

Найбільш розповсюдженими методами цифрової стеганографії є методи, що працюють у просторовій та частотній областях.

Переваги методів, що приховують дані у частотній області:

- секретне повідомлення захищене від шумів;
- секретне повідомлення захищене від фільтрів;
- секретне повідомлення захищене від ущільнення.

Недоліки цих методів:

- досить малий розмір секретного повідомлення відносно розміру контейнера;
- спотворення, що виникає у результаті зміни коефіцієнтів [22];
- складність використання;
- мала ступінь компресії для досягнення найкращих результатів.

Оскільки файли мультимедійних даних часто мають розміри набагато більші за файли текстових повідомлень, то головним критерієм для вибору групи методів є розмір вбудованих конфіденційних даних відносно розміру контейнера. Методи, що працюють у просторовій області, наприклад LSB-метод, надають можливість вбудовувати до 50% конфіденційних даних відносно розміру контейнера. Тому для стеганографічного захисту мультимедійних даних користувачів мережі Інтернет будуть розглядатись саме методи, що працюють у просторовій області.

Найчастіше у ролі контейнеру виступають графічні та аудіо-файли, оскільки вони характеризуються великою надлишковістю [138]. Базовим методом стеганографічного захисту є LSB-метод [13]. Суть даного методу полягає у заміні останніх найменш значущих бітів у контейнері на біти секретного повідомлення. LSB-метод та його модифікації добре підходять для форматів без ущільнення [32, 35], оскільки у результаті збереження файлу не втрачається частина секретної інформації.

Варто зазначити, що неможна виділити одну групу методів як найкращу, оскільки всі підходи мають свої особливості і призначені для оброблення даних різних форматів.

1.2 Порівняльний аналіз стеганографічних методів захисту мультимедійних даних

Розглянемо методи стеганографії, що передбачають оброблення даних у просторовій області. Оскільки у даній роботі представлено метод, що ґрунтується на використанні дерева Хаффмана, пропонується розглянути існуючі стеганографічні методи, що також базуються на використанні дерева Хаффмана.

Один з таких методів представлений у роботі [108]. Даний метод вбудовує конфіденційні текстові повідомлення у зображення. На початку зображення, що

виступає у ролі контейнера, ділиться на блоки 3х3 пікселя, що не перетинається між собою. Кожен канал зображення оброблюється окремо. Отже, кожен блок каналу можна представити у вигляді матриці розмірність 3х3. На основі конфіденційних текстових даних будується дерево Хаффмана, що і виступає ключем. Далі коди Хаффмана вбудовуються у блоки контейнера.

У роботі [72] представлено стеганографічний метод для вбудовування конфіденційних даних у зображення. Особливістю даного методу є використання процедури чотирьохрівневого зберігання розміру вбудованого двійкового масиву конфіденційних даних. Розглянемо основні етапи процесу вбудовування даного методу:

1. Для конфіденційних даних застосовується кодування Хаффмана результатом якого є дерево Хаффмана та конфіденційні дані у вигляді двійкового масиву у який записані коди Хаффмана.
2. Визначення розміру двійкового масиву кодів Хаффмана.
3. За допомогою процедури чотирьохрівневого зберігання розміру двійкового масиву, довжина бінарної послідовності вбудовується у перший блок пікселів розмірністю 8 на 8 пікселів зображення, що виступає у ролі контейнера.
4. За допомогою LSB-методу вбудовується побудоване дерево Хаффмана у пікселі, що не входять до початкового блоку, де зберігається довжина бінарної послідовності.
5. LSB-методом вбудовується сама бінарна послідовність.

Недоліком даного методу є те, що дерево Хаффмана вбудовується у контейнер разом із конфіденційним повідомленням. Таким чином, знаючи метод вбудовування, можна з легкістю декодувати секретні дані. Також, оскільки для вбудовування використовується LSB-метод, то даний метод є вразливим до статистичних атак.

Ще один стеганографічний метод вбудовування графічних даних у зображення був представлений у роботі [101]. Даний метод використовує міжпіксельне різницеве значення. Як і в попередньому методі, кожен канал оброблюється окремо. Перший елемент кожного рядка міжпіксельної різницевої матриці використовується як початкове значення, а решта елементів поточного рядка заповнюються як різниця попереднього та поточного значення. Таким чином, процес вбудовування конфіденційного зображення складається з наступних етапів:

1. Обчислення матиці міжпіксельних різниць для конфіденційних графічних даних.
2. Застосування кодування Хаффмана до матриці міжпіксельних різниць результатом якого буде дерево Хаффмана та бітова послідовність кодів Хаффмана.
3. За допомогою LSB-метода вбудовування дерева Хаффмана та бітової послідовності у зображення-контейнер.
4. Передача заповненого контейнера отримувачу.

Оскільки дерево Хаффмана вбудовується у контейнер разом із конфіденційним повідомленням, то знаючи метод за допомогою якого вбудовувались дані, можна з легкістю декодувати секретне повідомлення. Якщо ж передавати дерево Хаффмана окремо у вигляді симетричного ключа, то даний ключ є досить підозрілим для стегоаналітика. Оскільки даний метод для вбудовування використовує звичайний LSB-метод та вбудовує дані послідовно, то він є вразливим до статистичних атак.

Розглянемо метод з фрагментацією стегоданих, що вирішує дану проблему. Ідея даного методу полягає в захисті від несанкціонованого доступу до стегоданих шляхом застосування деякого унікального порядку розміщення даних у контейнері. При цьому враховується можливість того, що відомими є, як сама наявність стегоданих в контейнері, так і те, що (в яких байтах, бітах) потенційно може зберігатись інформація, яка відноситься до стегоданих. Що

залишається невідомим стегоаналітику – це те, яка саме частина даних у файлу є стегоданими, а яка – звичайними даними [58].

Процес вбудовування конфіденційних даних методом з фрагментацією стегоданих складається з наступних кроків:

1. Для кожного блоку для вбудовування генерується випадкове значення адреси.
2. Після обчислення адреси початку блоку даних обчислюється випадкове значення розміру даного блоку.
3. Перевіряється умова перетину блоків.
4. Від загального розміру конфіденційних даних віднімається розмір згенерованого блоку.
5. Ітерації повторюються, доки розмір конфіденційних даних не дорівнюватиме 0.
6. Після генерації векторів адрес блоків та їх розмірів починається вбудовування конфіденційних даних. Дані вектори є для даного методу ключами.

Перевагою даного методу є те, що він є стійким до певних видів статистичних атак, оскільки вбудовує дані не послідовно. Разом з тим, даний метод може працювати як з графічними даними, так і з аудіо. Також існує розпаралелена реалізація даного методу, що прискорює його швидкодію.

Основним недоліком даного методу є досить велика підозрілість ключів, оскільки обидва ключі представляють собою послідовності натуральних чисел.

Було проаналізовано програмну реалізацію методу з фрагментацією стегоданих. Аналіз показав, що адреси та розмір фрагментів генеруються за допомогою одного ГПВЧ. Таким чином, підібравши зерно для нього, можна отримати всю послідовність адрес та розмірів фрагментів. Для зерна існує 2^{32}варіантів. Таким чином, після аналізу програмного коду, ймовірність декодування стегоданих методом з фрагментацією стегоданих зводиться до наступної ймовірності:

$$P(F) = \frac{1}{2^{32}}$$

Розглянемо метод цифрової стеганографії на основі модифікації колірних параметрів зображення [8]. Даний метод є модифікацією LSB-методу та застосовується для вбудовування текстової інформації у зображення. Може бути адаптований для вбудовування графічних даних. Процес вбудовування конфіденційних даних складається з наступних етапів:

1. Авторський текст відповідно до кодування ASCII конвертується у числовий вигляд, символи замінюються відповідними числовими кодами з ASCII-таблиці.
2. За допомогою ключа визначається область у зображенні-контейнері для вбудовування даних.
3. Заміна значень трьох колірних компонент R, G та B відбувається не у двійковому вигляді, як у класичному LSB-методі, а у десятковому вигляді. Замінюються найменш значущі праві цифри відповідної компоненти.

Таким чином, для вбудовування одного символу конфіденційного повідомлення необхідно один піксель зображення-контейнера. При адаптуванні даного методу для вбудовування графічних даних, для приховування одного пікселя конфіденційного зображення необхідно три пікселі контейнера.

Перевагою даного методу є досить велика кількість конфіденційних даних, що можна вбудувати у поточний контейнер.

Порівняно з класичним LSB-методом об`єм конфіденційних даних, що можна вбудувати методом основі модифікації колірних параметрів зображення, приблизно на 40-50% більший.

Основним недоліком даного методу є те, що існує велика ймовірність заміни усіх 4-х молодших біт. Це може призвести до спотворень зображення, що можуть бути помітними неозброєним оком. Разом з тим, даний метод є нестійким до статистичних атак.

Розглянемо метод цифрової стеганографії на основі алгоритму шифрування AES (Rijndael).

Перед вбудовуванням стегоданих, даний метод шифрує стегодані за допомогою симетричного алгоритму блочного шифрування AES. Може працювати як з графічними даними, так і з аудіоданими.

У роботі [57] було проаналізовано швидкодію алгоритмів AES та 3DES і було визначено, що 3DES працює набагато повільніше, тому даний алгоритм не розглядався при порівнянні крипто-стеганографічних методів. Разом з тим, метод на основі AES при вбудовуванні у невеликі контейнери показує часові результати близькі до результатів методу з фрагментацією стегоданих у ідентичних умовах, але при вбудовуванні стегоданих великого розміру, метод на основі AES показує гірші часові показники процесу вбудовування даних. Виходячи з цього, можна зробити висновок, що у абсолютній більшості випадків метод на основі фрагментації стегоданих є швидшим за метод на основі алгоритму шифрування AES.

Оскільки алгоритм AES має можливість використовувати ключі розміром 128, 192 та 256 біт, то ймовірність підбору 256-го ключа дорівнює:

$$P \approx 1/2^{256} \approx 8{,}63^{-78}$$

Перевагою даного методу є висока криптографічна стійкість порівняно з методами описаними вище. Разом з тим, даний метод може працювати як з графічними даними, так і з аудіо.

Основними недоліками даного методу є час процесу вбудовування стегоданих та вразливість до статистичних атак.

Отже, проведений аналіз методів дозволив виявити низку недоліків, основними з яких є нестійкість до статистичних атак, низька візуальна стійкість заповненого контейнера, відсутність ключів або висока ймовірність їх підбору у випадку наявності.

1.3 Порівняльний аналіз програмних засобів захисту мультимедійних даних та аналіз вимог до програмного забезпечення захисту мультимедійних даних

Розглянемо програмні реалізації існуючих стеганографічних методів [88]. Оскільки метою дослідження є створення стеганографічних засобів захисту персональних мультимедійних даних користувачів, то програмні засоби для вирішення задач підтвердження авторства та походження даних розглядатися не будуть. Розглянемо найбільш відомі програмні засоби стеганографічного захисту даних:

1. S-Tools – програмний засіб для приховування конфіденційної інформації у графічних даних та аудіоданих [146]. Був розроблений Енді Брауном та відноситься до стеганографічних систем з послідовним вбудовуванням. Працює з форматами BMP, GIF, WAV. Має можливість шифрувати стегодані загальновідомими алгоритмами DES та IDEA.
2. OutGuess – стеганографічний інструмент для вбудовування конфіденційної інформації у графічні дані [145]. Підтримує зображення у форматах JPEG та PNM. Вбудовує стегодані таким чином, що вони рівномірно розподіляються по всьому контейнеру. Додатково може використовувати текстовий пароль для декодування стегоданих. Базується на модифікації молодших бітів обраного неодиничного та ненульового коефіцієнту JPEG значеннями секретного повідомлення.
3. JSteg – програмний засіб вбудовування стегоданих у зображення форматом JPEG [143]. Був розроблений Дереком Уфамом у 2004 році. Дає можливість вбудовувати повідомлення розміром до 12,8% відносно розміру контейнера. Базується на заміні молодшого біту отриманого після квантування частотних коефіцієнтів дискретного косинусного перетворення на біти секретного повідомлення. Не використовує ключів.

4. Hide4PGP – програмний засіб для вбудовування стегоданих у графічні та звукові контейнери [142]. Підтримує формати BMP, WAV, VOC. Перед вбудовуванням шифрує секретне повідомлення криптографічним алгоритмом. Є вразливим до статистичних атак.
5. Image steganography – засіб для приховування зображень у молодші біти зображення, що виступає у ролі контейнера. Перевагою даного засобу є те, що не потрібно завантажувати програмний додаток на комп'ютер, скористатись ним можна через сайт.
6. Hexa-Stego – простий програмний засіб для вбудовування конфіденційних файлів у графічні файли формату BMP.

Аналіз наявних у відкритому доступі програмних засобів дозволив виявити їх властивості, що наведені у табл. 1.1.

Таблиця 1.1 – Порівняння властивостей стеганографічних програм

Назва програми	Hide4PGP	Image Steganography	HexaStego	S-Tools	OutGuess	JSteg
Можливість працювати без ключів	Так	Так	Так	Ні	Ні	Так
Змінна кількість ключів	Відсутні ключі	Відсутні ключі	Відсутні ключі	Ні	Ні	Відсутні ключі
Шифрування даних	Ні	Ні	Ні	Так	Так	Ні
Підозрілість ключів	Відсутні ключі	Відсутні ключі	Відсутні ключі	Висока	Висока	Відсутні ключі
Кросплатформеність	Так	Онлайн	Ні	Ні	Ні	Так
Стійкість до статистичного аналізу	Ні	Так	Ні	Ні	Так	Ні
Можливість працювати з графічними даними та аудіо	Так	Ні	Ні	Так	Ні	Ні
Можливість масштабування	Ні	Ні	Ні	Ні	Ні	Ні
Точна відповідність стегоданих	Так	Ні	Так	Так	Так	Так

Аналіз існуючих програмних засобів дозволив виявити такі недоліки:

- не всі з розглянутих програмних засобів шифрують секретне повідомлення перед вбудовуванням, а вбудовують стегодані у відкритому вигляді [136];

- ключі, що використовуються для вбудовування конфіденційних даних, часто є результатом хеш-функції, а отже, є досить підозрілими для стегоаналітика;
- лише три програмних засоби є кросплатформеними (один з них онлайн) та можуть використовуватись у різних операційних системах;
- лише два програмних засоби є стійкими до статистичних атак;
- лише два з розглянутих програмних засобів підтримують одночасну роботу з аудіо та графічними файлами. Більшість з них орієнтована лише на окремі формати даних;
- жоден з програмних засобів не забезпечує можливість додавання нових крипто-стеганографічних методів захисту даних та можливості для постачання даних с інших програмних середовищ;
- не всі програмні засоби декодують дані не конвертуючи їх.

Проаналізувавши існуючі програмні засоби стеганографічного захисту персональних конфіденційних даних, можна зробити висновок, що жоден з розглянутих програмних засобів не шифрує стегодані перед вбудовуванням, не підтримує одночасно як графічні, так і аудіоконтейнери, не забезпечує стійкість до статистичних атак та можливість масштабування.

Вимоги до програмного забезпечення можна розділити на дві великі групи:

- функціональні вимоги, що описують поведінку системи, а саме роботу з даними, дії користувачів при роботі з цими даними;
- нефункціональні вимоги, що описують як система повинна працювати та якими властивостями та характеристиками вона повинна володіти.

У більшості випадків під нефункціональними вимогами розуміють вимоги, що визначають якісні характеристики розроблюваного програмного забезпечення, а саме продуктивність, надійність, масштабованість.

Таким чином, до програмної системи для захисту мультимедійних даних можна висунути наступні функціональні вимоги:

1. Шифрування стегоданих перед вбудовуванням. Для підвищення стійкості до декодування стегоаналітиком конфіденційні дані необхідно шифрувати перед вбудовування стеганографічними методами.
2. Використання ключів, що не будуть підозрілими для стегоаналітика (зображення, файли, тощо).
3. Підтримка графічних та аудіоданих. Для зручності використання програмної системи користувач повинен мати можливість вбудовувати мультимедійні дані у графічні та аудіоконтейнери.
4. Декодування конфіденційних даних без втрат.

Додатковою важливою вимогою до програмного забезпечення процесу захисту мультимедійних даних є підвищена швидкодія програмного застосунку.

Основними нефункціональними вимогами є:

1. Кросплатформеність для використання у різних ОС.
2. Стійкість до статистичних стегоатак.
3. Можливість розширення та інтегрування нових крипто-стеганографічних методів захисту мультимедійних даних користувачів. Для швидкого та простого масштабування системи, необхідно уніфікувати програмний інтерфейс для економії часу програміста при додаванні нових крипто-стеганографічних методів.
4. Можливість інтегрування нових способів постачання вхідних даних. Необхідна можливість постачання вхідних даних не тільки через головне вікно програмного застосунку, а наприклад, і через Інтернет-сервіси.

Отже, сформовано основні вимоги до програмного забезпечення для захисту конфіденційних персональних даних користувачів мережі Інтернет.

1.4 Висновки до розділу 1

У цьому розділі досліджено існуючі методи крипто-стеганографічного захисту даних та програмні засоби, що їх реалізують, а також проаналізовано

вимоги до програмного забезпечення процесів захисту мультимедійних даних користувачів мережі Інтернет. Проведене дослідження дозволяє зробити такі висновки.

1. Сформовано основні вимоги до прикладного програмного забезпечення для захисту конфіденційних мультимедійних даних користувачів мережі Інтернет з метою забезпечення можливості його подальшого масштабування, зокрема, додавання нових крипто-стеганографічних методів та інтегрування в інші програмні середовища.
2. Проведено порівняльний аналіз методів захисту мультимедійних даних. Визначено, що для забезпечення захисту конфіденційних мультимедійних даних користувачів мережі Інтернет доцільно розроблювати програмне забезпечення, яке реалізує одночасно декілька груп методів захисту.
3. Серед доступних у відкритому доступі програмних засобів крипто-стеганографічного захисту даних жоден не дозволяє шифрувати стегодані перед вбудовуванням, не дозволяє використовувати як графічні, так і аудіоконтейнери та не є стійким до статистичних атак. Цей факт дозволяє визначити напрямок подальшого дослідження та вимоги до архітектури програмної системи.

РОЗДІЛ 2. АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ПРОЦЕСІВ ОБРОБЛЕННЯ ДАНИХ ДЛЯ КРИПТО-СТЕГАНОГРАФІЧНОГО ЗАХИСТУ МУЛЬТИМЕДІЙНИХ ДАНИХ

2.1. Алгоритмічне забезпечення процесів оброблення даних для крипто-стеганографічного захисту графічних даних методом на основі схеми відповідності бітів

Цифрові графічні дані являють собою одну чи декілька матриць значень інтенсивності відтінку кольору пікселів зображення. Піксель – це найменший елемент зображення, що використовується у растровій графіці [131]. Він характеризується відтінком кольору, що описується певною колірною моделлю.

Колірна модель – це абстрактна модель, що характеризує способи відображення кольорів у вигляді набору чисел та, як правило, у вигляді трьох-чотирьох колірних компонент [126]. Однією з найбільш популярних колірних моделей є RGB [150]. Кожен піксель моделі RGB описується 24 або 32 бітами (RGBA), при цьому 32-бітний колір являє собою 24-бітний колір з допоміжним 8-бітним каналом, що заповнений нулями або являє собою альфа-канал. Альфа-канал задає прозорість зображення для кожного пікселя. Пустий альфа-канал використовують для прискорення оброблення зображень відеокартами, оскільки більшість сучасних комп’ютерів мають 32-бітну адресацію та 32-бітну шину даних. У колірній моделі RGBA існує 16777216 відтінків кольору та 256 градацій прозорості.

Основні переваги колірної моделі RGB:

- здатність передавати досить велику кількість відтінків кольорів;
- незалежність каналів (зміна одного з кольорів не впливає на інші);

Недоліки колірної моделі RGB:

- неможливість передати глибокі темні кольори;

- апаратна залежність;
- не дозволяє використовувати яскравість.

Вважається, що людське око помічає зміну яскравості більше, аніж зміну колірності [124]. Оскільки модель RGB не дозволяє це використовувати, то з точки зору стеганографії даний недолік можна розглядати як перевагу.

Розроблений метод на основі схеми відповідності бітів використовує даний факт для приховання персональних мультимедійних даних користувача. Цей метод ґрунтується на LSB-стеганографії (Least Significant Bit, найменший значущий біт) [60].

Основна ідея LSB-стеганографії полягає у змінні найменш значущих бітів у файлі-контейнері на біти повідомлення. У ролі контейнера можуть виступати графічні дані, аудіофайли та відеофайли. Різниця між порожнім та заповненим контейнером не повинна бути відчутною для органів сприйняття людини [66]. LSB-стеганографія не використовує ключів, а отже, для декодування повідомлення користувачам не потрібно обмінюватись жодною інформацією окрім заповненого контейнера.

Принцип LSB-стеганографії зображено на рис. 2.1.

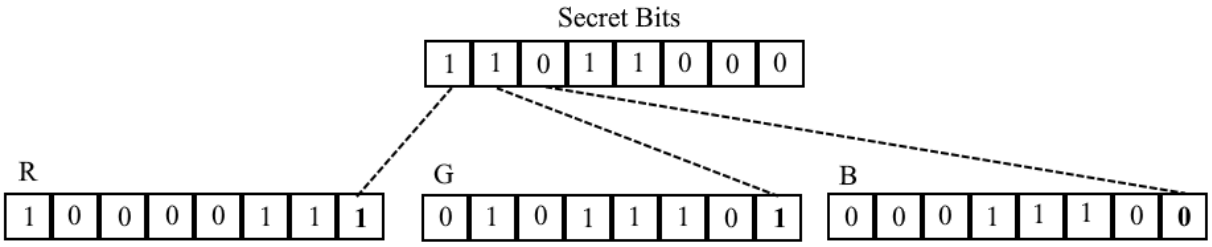

Рис. 2.1 Принцип LSB-стеганографії

Метод на основі схеми відповідності бітів є модифікацією базового LSB-методу [84, 93]. Його перевага полягає у більш високій надійності порівняно з оригіналом. Суть методу полягає у тому, що у контейнер вбудовується не

стегодані, а результат певної логічної функції аргументами для якої виступають біти стегоданих та старші (незмінні) біти контейнера [27, 40].

Логічні функції від двох змінних наведено у табл. 2.1.

Таблиця 2.1 – Таблиця логічних функцій

Аргументи		Логічні функції															
A	B	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Для використання у методі на основі схеми відповідності бітів підходять не всі логічні функції [47]. Функція повинна відповідати наступним критеріям:

- у 4 результатах логічної функції повинно бути 2 нулі та 2 одиниці;
- якщо A = 0, то результатами двох функцій (B = 0 та B = 1) повинні бути 1 та 0 у довільному порядку.

Даним критеріям відповідають 4 логічні функції – f6, f7, f10, f11 (табл. 2.2).

Метод на основі схеми відповідності бітів може використовуватись як з ключами, так і без.

Для коректного декодування стегоданих отримувач повідомлення повинен знати 2 ключі:

- логічну функцію;
- схему відповідності бітів.

У випадку, коли відправник та отримувач повідомлення домовились не використовувати ключі, за замовчуванням, як перший ключ, застосовується логічна функція F2 – виключна диз’юнкція (додавання за модулем 2) [60].

Таблиця 2.2 – Логічні функції для використання у методі на основі схеми відповідності бітів

Аргументи		Логічні функції			
A	B	F1 (f6)	F2 (f7)	F3 (f10)	F4 (f11)
0	0	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	1	1	0	1	0

Метод на основі схеми відповідності бітів дозволяє приховувати конфіденційні графічні дані у останні 1, 2, 3 або 4 біти кожної компоненти (R, G та B). При заміні більше 4 бітів, зображення, що виступає у ролі контейнера, досить сильно спотворюється.

Оскільки у багатьох випадках контейнер містить порожню компоненту, що відповідає за альфа-канал, то у методі на основі схеми відповідності бітів вбудовування у альфа-канал не відбувається, оскільки це може викликати підозру у стегоаналітика.

Залежно від кількості бітів у які буде вбудовуватись інформація змінюється і максимальний розмір повідомлення.

$$m = \frac{b \cdot (c - t - w - h - 1)}{8}, \tag{2.1}$$

де m – максимальний розмір конфіденційних графічних даних у байтах;

b – кількість біт, що змінюються;

c – розмір контейнера у байтах;

t – розмір метаданих контейнера у байтах;

w – кількість байт для запису ширини вбудованого зображення;

h – кількість байт для запису висоти вбудованого зображення.

Для мінімізації спотворення контейнера рекомендується змінювати один наймолодший біт.

За другий ключ береться схема відповідності бітів. Кожен біт стегоданих порівнюється зі старшим незмінним бітом контейнера за обраною логічною

функцією. Схема відповідності може бути передана відправником повідомлення отримувачу будь-яким іншим каналом зв'язку. У випадку, коли немає можливості передати ключі, у методі використовується схема відповідності біт за замовчуванням залежно від розміру конфіденційних графічних даних.

Очевидним є той факт, що чим менше метод спотворює контейнер, тим менша ймовірність розкриття факту наявності в ньому стегоданих. Тому залежно від розміру повідомлення у методі автоматично обирається мінімальна кількість змінюваних біт для вбудовування у контейнер.

Якщо виконується нерівність (2.2), то вбудовування відбувається зі зміною одного наймолодшого біту.

$$s < \frac{c-t-w-h-1}{8}, \tag{2.2}$$

де s – розмір стегоданих у байтах.

Схема відповідності біт при зміні одного наймолодшого біту за замовчуванням зображено на рис. 2.2. Наймолодші біти, що підлягають заміні, позначені буквою "M".

При виконанні нерівності (2.3) вбудовується повідомлення зі зміною двох наймолодших біт.

$$s < \frac{c-t-w-h-1}{4}, \tag{2.3}$$

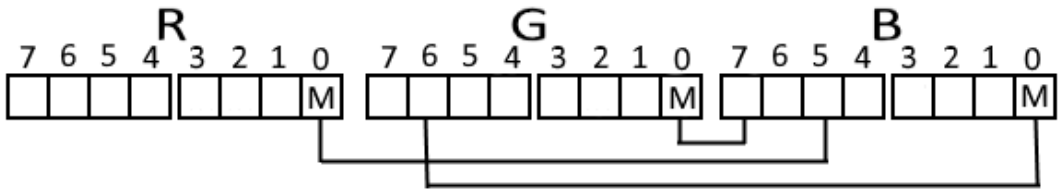

Рис. 2.2 Схема відповідності біт при зміні одного наймолодшого біту за замовчуванням

Схема відповідності біт при зміні двох наймолодших бітів за замовчуванням зображено на рис. 2.3.

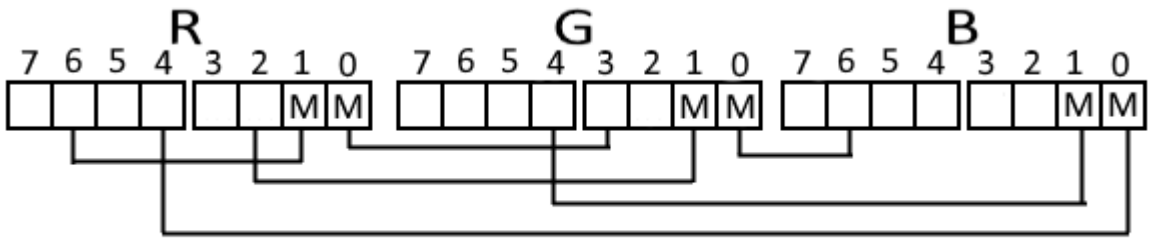

Рис. 2.3 Схема відповідності біт при зміні двох наймолодших бітів за замовчуванням

Якщо виконується нерівність (2.4), конфіденційні графічні дані вбудовуються зі зміною трьох наймолодших біт.

$$s < \frac{3 \cdot (c - t - w - h - 1)}{8}, \tag{2.4}$$

Схема відповідності біт при зміні трьох наймолодших бітів за замовчуванням зображено на рис. 2.4.

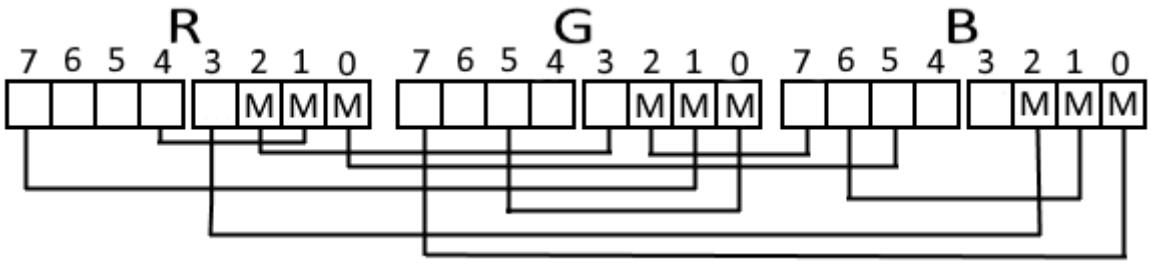

Рис. 2.4 Схема відповідності біт при зміні трьох наймолодших бітів за замовчуванням

При виконанні нерівності (2.5) вбудовування відбувається зі зміною чотирьох наймолодших бітів.

$$s < \frac{c - t - w - h - 1}{2}, \tag{2.5}$$

Схема відповідності біт при зміні чотирьох наймолодших бітів за замовчуванням зображено на рис. 2.5.

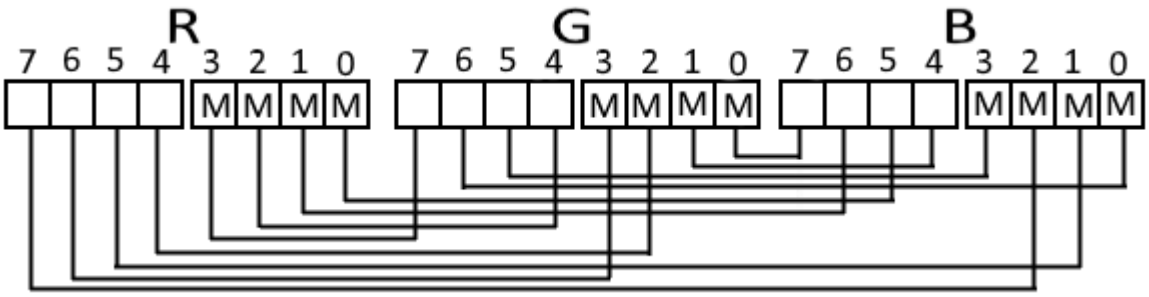

Рис. 2.5 Схема відповідності біт при зміні чотирьох наймолодших бітів за замовчуванням

Якщо жодна з нерівностей не виконується, вбудувати повідомлення неможливо, оскільки його розмір занадто великий для поточного контейнера. У такому випадку треба обрати інший контейнер.

Створена користувачем власна схема відповідності бітів повинна відповідати певному критерію: бітом у контейнері для порівняння за допомогою обраної логічної функції не може виступати біт, що в процесі вбудовування змінювався або може бути зміненим під час подальшого вбудовування повідомлення. Приклад неправильного зв’язування бітів зображено на рис. 2.6.

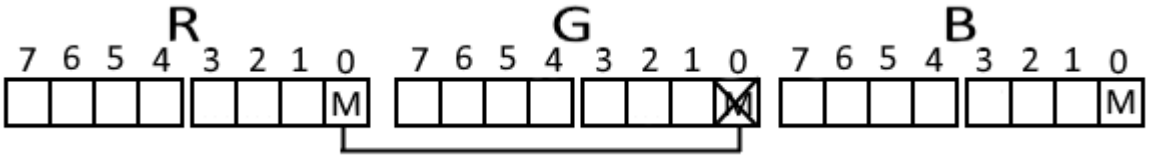

Рис. 2.6 Приклад неправильного зв’язування бітів

У випадку, зображеному на рис. 2.6, результат декодування може бути невірним. Оскільки після зміни біту на позицію G[0] при декодуванні біту R[0] результат порівняння бітів G[0] та R[0] за допомогою обраної логічної функції може відрізнятися від результату на етапі вбудовування повідомлення. Таким

чином, якщо декодовані біти відповідали за молодші біти у компонентах R, G та B зображення, що вбудовується, то декодоване зображення при візуальному порівнянні з оригіналом може не відрізнятися. Але у випадку, якщо декодовані біти відповідали за старші біти у компонентах, кольори певних пікселів у декодованому зображені та оригіналі можуть суттєво відрізнятися.

Декілька молодших бітів різних компонент можуть порівнюватись з одним старшим бітом будь-якої компоненти. На рисунку 2.7 видно, що біти на позиціях R[0] та G[0] порівнюються з бітом на позиції G[6]. При зміні одного біту не рекомендується повторювати старші біти для порівняння. При зміні чотирьох бітів навпаки, для зменшення ймовірності декодування повідомлення стегоаналітиком, рекомендується повторення однакових старших бітів.

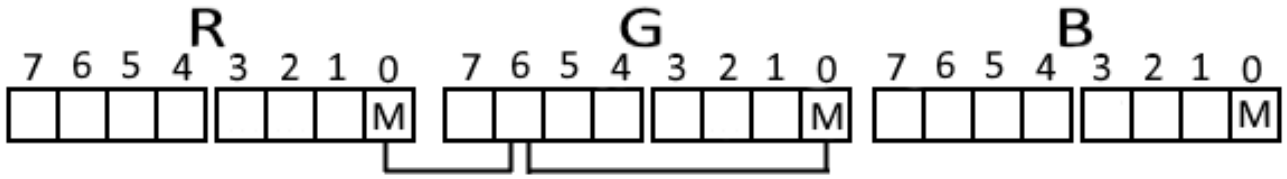

Рис. 2.7 Порівняння двох молодших біт з одним старшим

На початку процесу вбудовування стегоданих контейнер конвертується у масив байтів: спочатку компоненти R, G та B першого пікселя, далі компоненти R, G та B другого пікселя тощо. Таким чином у масиві будуть присутні усі пікселі контейнеру. Повідомлення, у свою чергу, конвертується у масив бітів.

Для коректного декодування вбудованого зображення крім інформації про значення компонент R, G та B пікселів, треба знати кількість бітів, що змінюються та роздільність прихованих графічних даних.

Кількість змінюваних бітів вбудовується у перший піксель контейнера. Оскільки дана величина може змінюватись від 1 (001 у двійковій системі числення) до 4 (100), то для її вбудовування необхідно лише три біти. Отже, у кожну компоненту R, G та B першого пікселя буде вбудовано по одному біту.

Ширина вбудованого зображення вбудовується у перші пікселі контейнера після пікселя з кількістю змінюваних бітів, а висота – у останні. Таким чином, вбудовування стегоданих у контейнер починається з першого пікселя, до якого не записана ширина конфіденційних графічних даних.

Теоретично, для запису ширини або висоти зображення, що вбудовується, може знадобитися як один піксель (для вбудовування зображення з розміром 7x7 при зміні одного біта), так і декілька. Для того, щоб додатково не зберігати інформацію про кількість пікселів, у які вбудовані ширина та висота конфіденційних графічних даних, на запис роздільності відводиться максимальна кількість пікселів.

Кожна компонента R, G та B займає один байт. Отже, один піксель без альфа-каналу займає 3 байти. Виходячи з формули (2.6) та знаючи максимальну кількість байт, що можна вбудувати у контейнер, можна порахувати максимальну кількість пікселів стегоданих:

$$p = \frac{m}{3}, \tag{2.6}$$

де p – максимальна кількість пікселів у вбудованому зображенні;
m – максимальний розмір конфіденційних графічних даних у байтах.

Теоретично, вбудоване зображення може мати розмір 1 x p або p x 1. Тому для запису ширини та висоти завжди відводиться така кількість пікселів, щоб була можливість вбудувати значення p.

Схематично перші чотири байти заповненого контейнера показані на рис. 2.8.

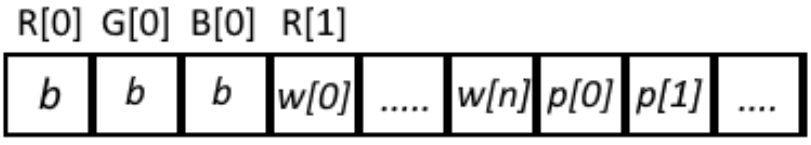

Рис. 2.8 Перші байти заповненого контейнера

На рис. 2.8 видно, що у перші три байти (компоненти R, G та B першого пікселя) вбудована інформація про кількість змінюваних біт у контейнері (b). У другий та наступні пікселі вбудовується інформація про ширину зображення ($w[0]...w[n]$).

Починаючи з першого пікселя, що не містить даних про ширину зображення ($p[0]$), вбудовується вже інформація про самі значення компонент R, G та B пікселів конфіденційних графічних даних.

Підсумувавши вищезазначене, можна виділити наступні кроки у методі на основі схеми відповідності бітів:

1. Конфіденційні графічні дані перевіряються на можливість вбудовування у обраний контейнер.
2. Якщо зображення можливо вбудувати, виходячи з його розміру визначається мінімальна кількість бітів, що змінюватимуться при вбудовуванні. У випадку, якщо контейнер замалий для обраних конфіденційних графічних даних користувача, пропонується обрати інший.
3. У перший піксель порожнього контейнера записується кількість змінюваних біт.
4. Визначається максимальна ширина та висота стегоданих, що можна вбудувати у контейнер та резервується починаючи з другого пікселя k пікселів для запису максимальної ширини зображення.
5. Вбудовується фактична ширина зображення.
6. У останні k пікселів вбудовується фактична висота стегоданих.
7. Починаючи з $k + 1$ пікселя вбудовується інформація про значення компонент R, G та B пікселів конфіденційних графічних даних за допомогою обраної логічної функції та схеми відповідності молодших та старших бітів.

   Діаграма діяльності вбудовування стегоданих зображена на рис. 2.9.

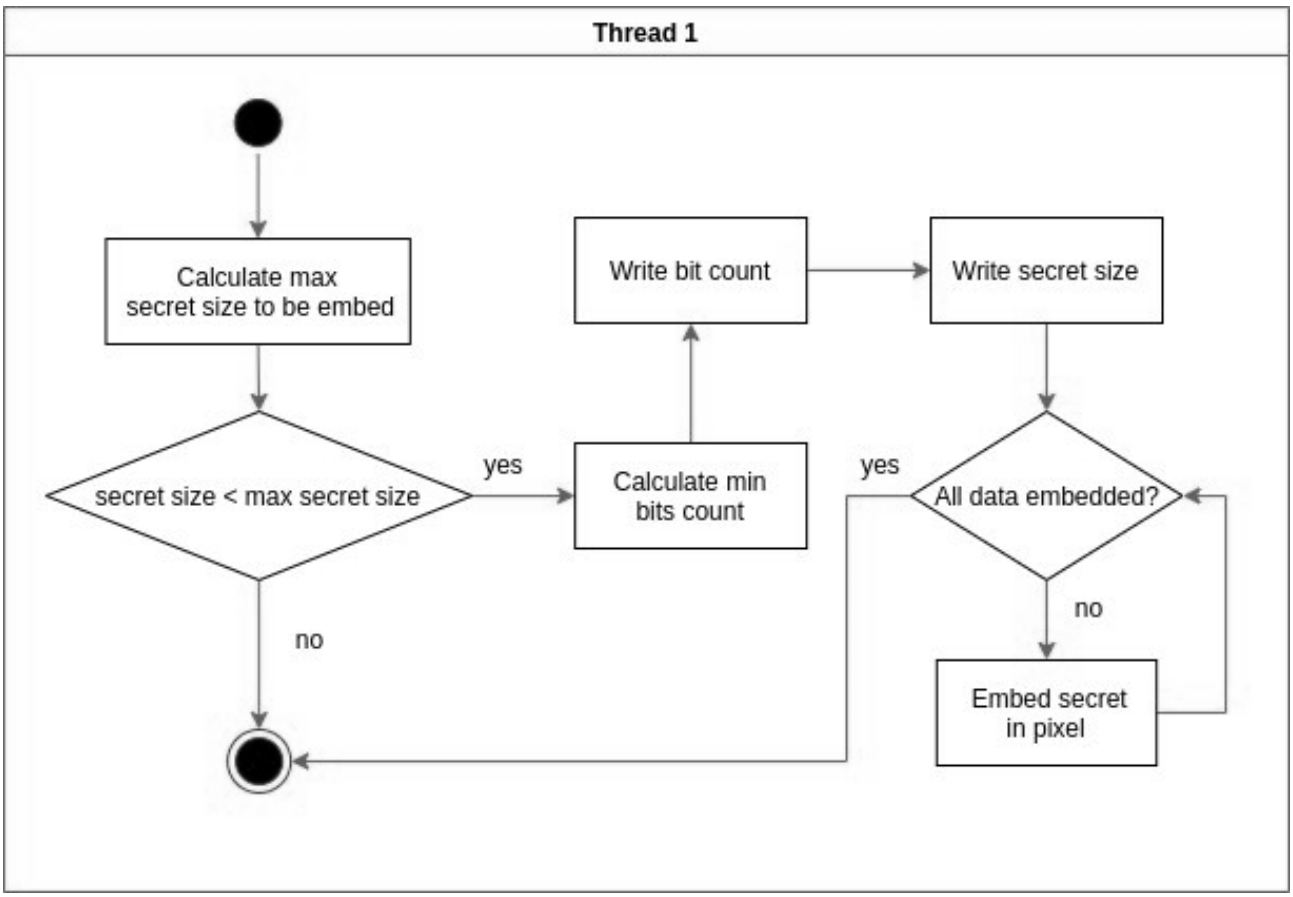

Рис. 2.9 Діаграма діяльності процесу вбудовування стегоданих

Декодування конфіденційних графічних даних користувача складається з наступних кроків:

1. Заповнений контейнер конвертується у масив байт наступним чином: перший байт – компонента R першого пікселя, другий – компонента G першого пікселя, третій – компонента B першого пікселя, четвертий – компонента R другого пікселя і т.д.
2. З першого пікселя заповненого контейнера зчитується кількість молодших бітів у байті, що змінюються на етапі вбудовування конфіденційних даних користувача.
3. Визначається максимальна розмір стегоданих, що можна вбудувати у контейнер використовуючи висоту та ширину порожнього контейнера та визначається необхідна кількість пікселів *K* у контейнері для вбудовування

максимальної ширини або висоти конфіденційного графічного повідомлення у пікселях.

4. Починаючи з другого пікселя зображення-контейнера зчитується ширина у пікселях конфіденційних графічних даних з перших K пікселів контейнера.
5. З останніх K пікселів у заповненому зображенні-контейнері зчитується висота прихованих конфіденційних графічних даних користувача у пікселях.
6. Починаючи з $K + 1$ пікселя послідовно зчитується інформація про значення компонент R, G та B пікселів конфіденційних графічних даних за допомогою обраної логічної функції та схеми відповідності молодших та старших бітів.
7. Заповнений масив байтів конвертується у зображення, що і є конфіденційними даними.

Після заповнення масиву компонентами R, G та B конфіденційних графічних даних, він конвертується у масив байтів та зберігається на носій як файл. Збережений файл і буде повідомленням.

2.2 Алгоритмічне забезпечення процесів оброблення даних для крипто-стеганографічного захисту графічних даних методом на основі дерева Хаффмана

Метод стеганографічного захисту графічних даних на основі дерева Хаффмана використовує зображення як ключ [85, 96, 147]. Основна ідея методу полягає у вбудовуванні не самих конфіденційних графічних даних, а кодів Хаффмана, що були отримані за допомогою зображення, що виступає у ролі ключа (далі ключ).

За ключ можна взяти лише ті зображення, компоненти R, G та B якого містять усі можливі значення [0; 255] хоча б один раз. Даний критерій

називається повнотою наявних байт [56]. Якщо хоча б одного значення не вистачає – зображення не може бути використане як ключ. Дана вимога висувається через те, щоб відправник та отримувач могли лише раз обмінятися ключем та використовувати його для багаторазової передачі графічних даних.

На початку роботи методу необхідно побудувати словник Хаффмана на основі ключа, де кожне ціле значення від 0 до 255 представлене у вигляді бінарного масиву.

Код Хаффмана – це вільний від префіксу код, що може давати найкоротшу середню довжину коду n для поточного вхідного алфавіту. Ефективність ущільнення визначається коефіцієнтом ущільнення. Ця міра дорівнює відношенню середнього числа біт на вибірку до ущільнення до середнього числа біт на вибірку після ущільнення [70]. Слід зауважити, що алгоритм ущільнення Хаффмана є алгоритмом ущільнення без втрат. Саме ця особливість використовується у методі стеганографічного захисту графічних даних на основі дерева Хаффмана.

Основна ідея полягає у тому, що знаючи ймовірність появи певних символів у повідомленні, можна описати процедуру побудови кодів змінної довжини, що складаються зі скінченної кількості бітів. Чим більша ймовірність появи символу, тим коротший його код. Оскільки жоден код не є префіксом іншого – це дозволяє однозначно декодувати інформацію.

Треба звернути увагу на той факт, що для кодування будь-якого контенту за допомогою коду Хаффмана треба двічі пройтися по ньому, саме тому даний метод називають двопрохідним. Для кодування тексту проходимо перший раз, щоб дізнатися частоту букв і на основі даної інформації побудувати дерево Хаффмана. Далі проходимо другий раз, щоб ущільнити за допомогою дерева Хаффмана кожну букву у вхідному тексті. У результаті отримано ущільнений текст.

Вхідними даними для алгоритму Хаффмана є набір символів з кількістю їх повторень у повідомленні, що ущільняється. На основі даного набору і будується дерево Хаффмана.

Розглянемо побудову дерева на прикладі.

Нехай є таблиця кількості повторень певних символів у повідомленні (табл. 2.3).

Таблиця 2.3 – Таблиця кількості повторень символів у повідомленні

Символ	a	b	c	d	e	f
Кількість повторень	20	14	45	8	1	33

Спочатку необхідно відсортувати дані у табл. 2.3 за спаданням кількості повторень. Отримаємо наступну таблицю (табл. 2.4).

Таблиця 2.4 – Відсортована таблиця кількості повторень символів у повідомленні за спаданням

Символ	c	f	a	b	d	e
Кількість повторень	45	33	20	14	8	1

Ці символи будуть кінцевими вузлами бінарного дерева Хаффмана [86]. Наступним етапом справа наліво створюються нові вузли, що об’єднують символ з найменшою кількістю повторень та наступний за ним. Вузли, що були об’єднані на цьому рівні у поточній ітерації, більше не використовуються. Ребро від нового вузла до вузла з меншою кількістю повторень буде мати вагу 1, у той час як ребро до вузла з більшою кількістю повторень – 0. Графічно результат першої ітерації зображено на рис. 2.10.

Наступним етапом проходимо за аналогією по новоствореним вузлам *cf, ab* та *de*. Результат роботи алгоритму Хаффмана після другої ітерації графічно зображено на рис. 2.11.

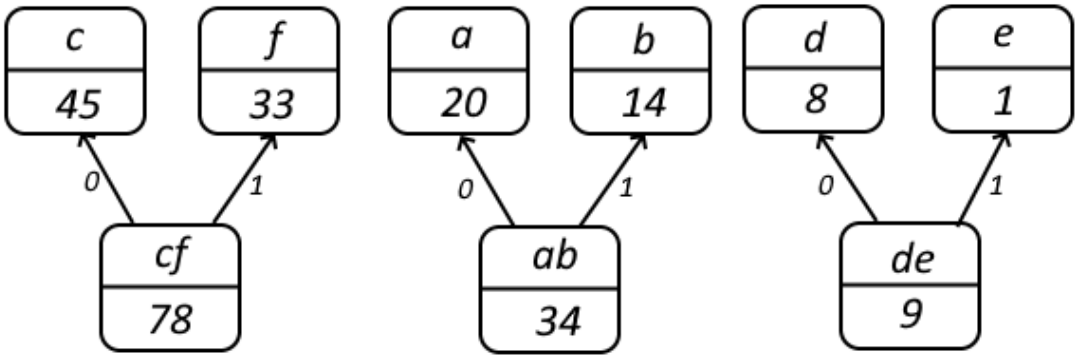

Рис. 2.10 Результат роботи алгоритму Хаффмана після першої ітерації

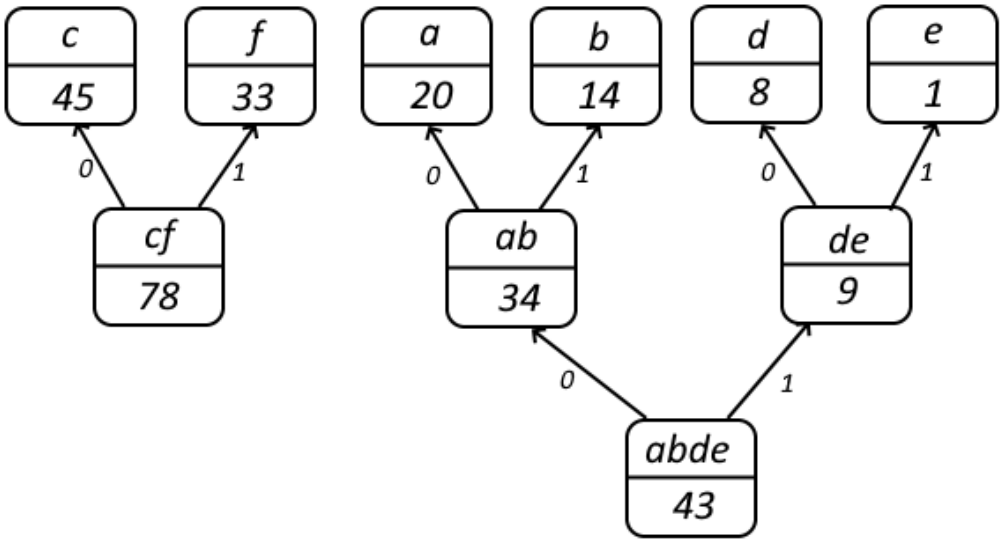

Рис. 2.11 Результат роботи алгоритму Хаффмана після другої ітерації

На третій ітерації об’єднуємо новостворений вузол *abde* та вузол для якого не знайшлось пари на попередній ітерації (у випадку, якщо кількість вузлів у попередній ітерації була непарна).

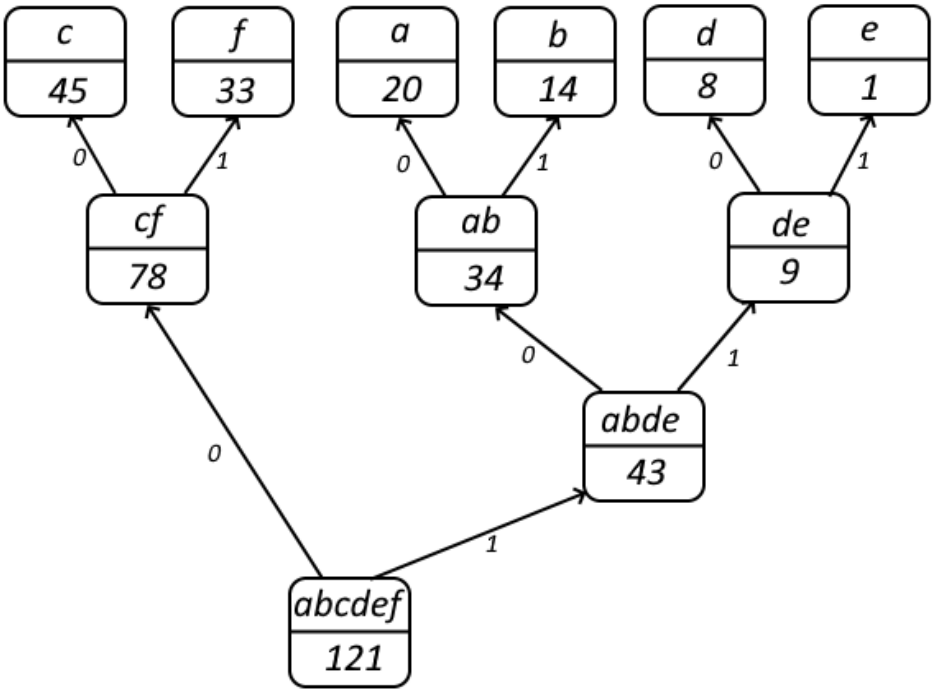

Рис. 2.12 Результат роботи алгоритму Хаффмана після третьої ітерації

Дерево вважається повністю завершеним, коли у нього є лише один корінь та відсутні вузли без вихідних ребр окрім початкових. Для того, щоб визначити код для кожного з символів що входять у повідомлення, треба пройти шлях від вузла з потрібним символом до кореня дерева та накопичувати біти (0 або 1) при кожному переміщенні з вузла на вузол. Отримана послідовність бітів і буде являти собою код даного символу у зворотному порядку. Коди кожного з вхідних символів показано у табл. 2.5.

Таблиця 2.5 – Коди Хаффмана

Символ	*a*	*b*	*c*	*d*	*e*	*f*
Код	100	101	00	110	111	01

Як видно з табл. 2.4 та табл. 2.5, найбільш часто повторювані символи *c* та *f* закодовані коротшим двозначним кодом, у той час, як символи, що зустрічаються рідше у повідомленні, мають більшу довжину бітового коду.

У методі стеганографічного захисту графічних даних на основі дерева Хаффмана побудова дерева відбувається, як розглянуто вище. Якщо у ключі є усі компоненти від 0 до 255, то дані значення виступають у ролі символу у наведеному прикладі. Таким чином, словник Хаффмана буде складатися з 256 символів.

У процесі кодування ключ не зазнає жодних змін, а отже не створює жодних підозр при передачі. Це дозволяє пересилати зображення, що виступає у ролі ключа, відкритими каналами зв’язку.

Суть методу полягає в тому, що замість вбудовування у контейнер самих стегоданих, у нього вбудовуються коди Хаффмана конфіденційного зображення, що отримані за допомогою ключа. Таким чином, наступним кроком після побудови дерева Хаффмана на основі ключа є конвертування повідомлення у масив байтів. Конвертування здійснюється таким же чином, як і у методі на основі схеми відповідності бітів: спочатку компонента R першого пікселя; далі компонента G першого пікселя; компонента B першого пікселя; компонента R другого пікселя тощо. Після конвертування отримуємо масив, кожен елемент якого знаходиться у проміжку [0; 255].

У загальному випадку, дерево Хаффмана будується двічі: при вбудовуванні конфіденційних графічних даних на стороні відправника та при декодуванні на стороні отримувача. Однак на стороні приймача при побудові додатково обчислюється довжина найкоротшого коду. Це зроблено для оптимізації процесу декодування даних.

На наступному етапі створюється результуючий порожній масив бітів, який і буде вбудований у контейнер після заповнення. Даний масив міститиме зашифроване конфіденційне зображення за допомогою словника Хаффмана. Заповнюється масив за таким алгоритмом:

1. Після створення словника Хаффмана на основі ключа, у циклі відбувається ітерування по масиву з байтами контейнера.

2. Для кожного значення байту у контейнері відбувається пошук коду у побудованому словнику Хаффмана.
3. Отриманий код додається до результуючого масиву бітів.

Після створення масиву кодів конфіденційного зображення відбувається перевірка на можливість вбудовування даного масиву в контейнер. Одним з недоліків методу стеганографічного захисту на основі дерева Хаффмана є те, що лише після заповнення масиву бітів для вбудовування можна проаналізувати, чи можливо вбудувати конфіденційні графічні дані у обраний контейнер.

На даному етапі слід зазначити, що метод на основі дерева Хаффмана може змінювати лише 1 останній біт на відміну від методу на основі схеми відповідності бітів, де була можливість змінювати від 1 до 4 бітів. Пояснення даному факту буде описано нижче. Таким чином, немає необхідності у вбудовуванні кількості змінюваних біт.

Перевірка на можливість вбудовування конфіденційних графічних даних відбувається наступним чином: виходячи з розміру двійкового масиву кодів Хаффмана визначається кількість байт для вбудовування. Далі, як і в методі на основі схеми відповідності бітів, визначається максимальний розмір даних у байтах, що можливо вбудувати в обраний контейнер. Якщо фактичний розмір масиву байт менший за максимальний можливий розмір масиву, повідомлення може бути вбудоване.

Для коректного декодування конфіденційних графічних даних треба визначити роздільність повідомлення на стороні отримувача. Як і в методі на основі схеми відповідності бітів, спочатку на основі розміру контейнера обчислюється максимальна кількість пікселів стегоданих p, що можна приховати. Наступним кроком значення p конвертується у двійкову систему числення та визначається кількість бітів b для запису значення p. Якщо b націло ділиться на 3, то для запису роздільності повідомлення резервується $\frac{b}{3}$ кількість пікселів. Оскільки у методі на основі дерева Хаффмана завжди змінюється лише

один наймолодший біт, то у кожен піксель можна вбудувати лише 3 біти стегоданих. У випадку, якщо $bmod3 \neq 0$, то кількість пікселів для резервування визначається за формулою:

$$r = toInt\left(\frac{b}{3}\right) + 1, \tag{2.7}$$

де *r* – кількість пікселів для резервування;

b – кількість біт для запису максимальної висоти або ширини *p* конфіденційних графічних даних;

toInt – функція для округлення параметра до меншого цілого значення, наприклад *toInt(2.45) = 2*.

Після визначення кількості пікселів для резервування *r*, у перші *r* пікселів записується ширина конфіденційних графічних даних, у останні *r* – висота.

Для більш ефективного стеганографічного захисту повідомлення, бітовий масив кодів Хаффмана вбудовується у контейнер не послідовно, а через певні інтервали *i*. Значення *i* також вбудовується у контейнер для коректного декодування конфіденційного зображення. Для вбудовування *i* зарезервовано 8 пікселів починаючи після останнього пікселя, у який вбудоване значення ширини. Оскільки у кожен піксель вбудовується 3 біти, то максимальне значення *i* визначається за формулою:

$$maxI = 2^{3 \cdot 8} - 1 = 2^{24} - 1 = 16777215, \tag{2.8}$$

де *maxI* – максимальне значення інтервалу *i*.

Даного значення досить для того, щоб на кожні 100000000 (100 мегапікселів) пікселів контейнера (роздільність 10000x10000 пікселів) вбудовувати по одному пікселю стегоданих. Значення для запису інтервалу у 8 пікселів обрано не випадково. Для більш надійного захисту стегоданих у якості контейнера слід обирати ексклюзивне зображення, що важко або неможливо знайти у мережі Інтернет. Тож для максимального захисту повідомлення, зображення у якості контейнера краще користувачу зробити самостійно.

Найпростішим способом створити своє зображення - фотозйомка. На даний час мобільний телефон – пристрій, що є у більш як 99% користувачів Інтернету. Флагманські мобільні пристрої на ринку мають камеру у 40 мегапікселів, але у більшості користувачів мобільні телефони з камерою роздільність якої не перевищує 12 мегапікселів [125]. Одна з найкращих цифрових камер, що представлені на ринку, має роздільність у 102 мегапікселя [34]. Значення для запису інтервалу у 8 пікселів дозволяє вбудовувати навіть конфіденційне зображення з розміром 1x1 у контейнер з роздільністю 102 мегапікселя. Звичайно, зображення з роздільністю 1x1 пікселів не інформативне і для запису одного пікселя немає ніякого сенсу використовувати такий контейнер, але метод надає таку можливість у випадку, коли сума довжин двійкового представлення кодів Хаффмана трьох компонент одного пікселя не перевищує 21 біт.

Інтервал *i* через який будуть вбудовуватись стегодані визначається за формулою:

$$i = toInt\left(\frac{m}{s}\right) - 1, \tag{2.9}$$

де *i* – інтервал у пікселях через який будуть вбудовуватись стегодані;
m – максимальна кількість байт, що можна вбудувати у контейнер;
s – фактична кількість байт конфіденційного зображення.

Якщо значення *i* дорівнює 0, то вбудовування відбувається у кожен піксель контейнера. У випадку, коли *i* дорівнює 1, вбудовування відбуватиметься у кожен другий піксель контейнера, 2 – кожен третій і т.д. Фактично *i* – кількість порожніх пікселів (без вбудованої інформації) між двома заповненими пікселями.

Наступним кроком значення *i* вбудовується у 8 перші пікселів за пікселями у котрі вбудована ширина повідомлення. Перші байти заповненого контейнера графічно зображено на рис. 2.13, де:

w – масив байтів зі значенням у двійковій системі числення ширини конфіденційного зображення;

n – максимальна кількість байт для запису значення ширини;

r – масив байтів для запису двійкового значення інтервалу;

p – масив байтів з вбудованими кодами Хаффмана конфіденційного зображення.

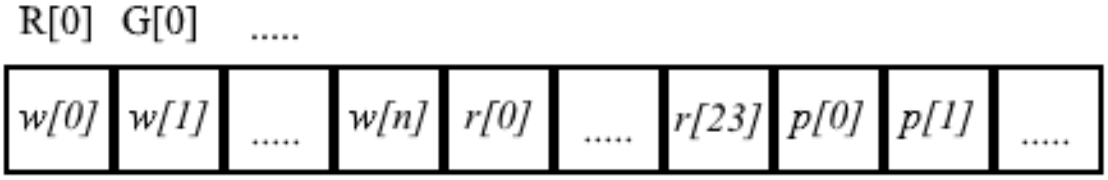

Рис. 2.13 Перші байти заповненого контейнера методом на основі дерева Хаффмана

Після вбудовування значення інтервалу починається вбудовування отриманих за допомогою ключа кодів Хаффмана. Для вбудовування даної інформації використовується модифікація методу на основі схеми відповідності бітів. При простій заміні наймолодшого біту у контейнері, деякі статистичні методи стегоаналізу (наприклад RS-метод) здатні розпізнати факт вбудовування повідомлення [128]. Зважаючи на даний факт, у методі на основі дерева Хаффмана аналізується весь байт у цілому.

На першому етапі, як і в методі на основі схеми відповідності бітів, за допомогою обраної схеми та логічної функції (або параметрів за замовчуванням) обчислюється значення біту, що треба вбудувати. Якщо отриманий біт та останній біт байту, що аналізується, співпадають, то даний байт залишається без змін і аналізується наступний. Якщо ж дані значення не співпадають, то до аналізуючого байту у псевдовипадковому порядку додається або ж віднімається 1. Таким чином, у певних випадках можуть змінитися всі 8 бітів аналізуючого байту. Даний метод ще називають “плюс/мінус один метод”.

Але є декілька виключень у випадку, коли останній біт байту та отриманий біт не співпадають:

- якщо значення байту дорівнює 0, то до даного байту завжди додається 1;
- якщо значення байту дорівнює 255, то від даного байту завжди віднімається 1.

На етапі декодування немає необхідності знати інформацію про те, чи у даному байті було додавання або віднімання одиниці, оскільки аналізується лише останній біт байту.

Після додавання або віднімання існує ймовірність того, що старші біти, що використовуються для порівняння за допомогою обраної схеми, можуть змінити своє значення. У такому випадку, декодувати повідомлення без спотворень, майже неможливо. Тому після зміни значення байту, поточний байт повторно аналізується.

Якщо старші біти, що використовуються у якості одного з двох аргументів логічної функції, у результаті додавання/віднімання не змінилися, вбудовування вважається успішний та аналізується наступний байт. Якщо ж старші біти було змінено, можливо 2 випадки:

1. Якщо старший біт було змінено у результаті додавання одиниці, то від значення байту віднімається 2. -1 у даному випадку бути не може, оскільки при додаванні 1 змінився би тільки наймолодший біт.
2. Якщо старший біт було змінено у результаті віднімання одиниці, то до значення байту додається 2. 256 у даному випадку бути не може, оскільки при відніманні 1 змінився би тільки наймолодший біт.

Діаграма діяльності процесу вбудовування зображена на рис. 2.14.

Після вбудовування конфіденційного повідомлення, заповнений графічний контейнер можна відправляти отримувачу. Для більш ефективного захисту вбудованих даних, контейнер та ключ краще відправляти різними каналами зв’язку.

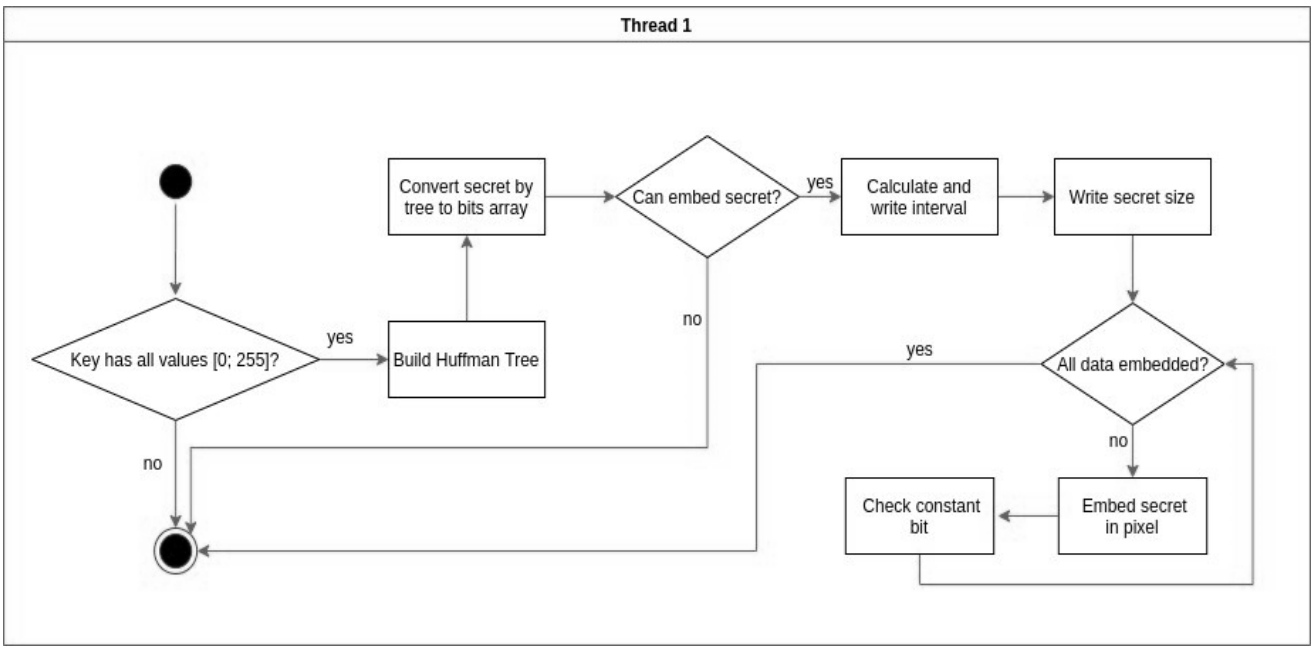

Рис. 2.14 Діаграма діяльності процесу вбудовування методу на основі дерева Хаффмана

Алгоритм декодування можна описати наступним чином:

1. Побудова дерева Хаффмана на основі отриманого ключа.
2. Конвертування заповненого контейнеру у масив байтів.
3. Визначення кількості пікселів для запису висоти та ширини конфіденційних графічних даних.
4. Зчитування висоти та ширини у пікселях конфіденційного зображення користувача.
5. Зчитування значення інтервалу.
6. Зчитування значення компонент R, G та B пікселів конфіденційного зображення.

Після заповнення масиву компонентами R, G та B конфіденційних графічних даних, він конвертується у масив байтів та зберігається на носій як зображення. Збережений файл і буде конфіденційним повідомленням користувача.

2.3 Алгоритмічне забезпечення процесів оброблення даних для крипто-стеганографічного захисту мультимедійних даних методом на основі процедури псевдовипадкового вбудовування

Розглянемо метод, який дозволяє вбудовувати конфіденційні мультимедійні дані у аудіодані.

На сьогодні існує велика кількість форматів збереження аудіо-даних. Найчастіше використовуються такі формати як MP3 і WAV [144, 151]. Тип формату визначається розширенням файлу.

Для ущільнення та кодування цифрових аудіоданих використовують кодеки [59]. Для деяких типів файлів кодек визначений досить неоднозначно. Наприклад, у форматі MP3 використовується кодек MPEG Layer-3, а в форматі MP4 можуть бути використані різні кодеки (H.264, MPEG-4 AVC) [129, 103].

Часто поняття формату і кодека не розділяють. Особливо, коли в форматі завжди використовується один і той же кодек. Для зручності, можна вважати, що формат є свого роду контейнером, в якому може бути записаний аудіо або відео сигнал за допомогою певного кодека. Деякі формати, як MP4 або FLV, можуть містити в собі як аудіо-потоки, так і відео-потоки.

Кодеки можна розділити на два види, залежно від типу ущільнення:

- з втратами якості звучання (MP3, WMA, OGG та ін.);
- без втрат якості звучання (FLAC, APE, ALAC та ін.).

Кодеки, що ущільнюють аудіо-дані, модифікують звуковий сигнал. З нього видаляються певні частоти, які людське вухо не здатне почути [130]. Кодеки без ущільнення даних здатні кодувати звук таким чином, що при декодуванні дані будуть повністю відновленні і співпадати з оригіналом.

Формат WAV являє собою файл-контейнер. Сам контент може бути ущільненим за допомогою різноманітних кодеків. Найбільш розповсюдженим є кодек Pulse Code Modulation (PCM) [115].

Формат WAV має ряд переваг перед іншими:

- сумісний з переважною більшістю аудіопристроїв;
- мінімальні зміни у оригінальному аудіо;
- містить всю необхідну інформацію, що була перетворена з аналогової форми в цифрову;
- може бути конвертований у інші формати аудіо-даних з мінімальними втратами якості або ж зовсім без втрат.

Отже, можна зробити висновок, що формат WAV є найбільш універсальним та у більшості випадків є першоджерелом для аудіо-даних у інших форматах.

Одним з найпопулярніших аудіо-форматів у мережі Інтернет є формат MP3. Він створений для кодування та зберігання звуку з невеликими для людського сприйняття втратами.

Алгоритм ущільнення, що використовується у форматі MP3 дозволяє суттєво зменшити розмір аудіо-даних до 12 разів. MP3 є потоковим форматом. Це означає, що звукова інформація при кодуванні розбивається на фрейми рівні по тривалості. Кожен фрейм має свій заголовок, параметри кодування та кодується незалежно від інших фреймів [37]. При програванні MP3-файлу послідовність декодованих фреймів забезпечує неперервне звучання. Цей підхід створює багато переваг для використання MP3:

- невеликий розмір файлу;
- легко перемотувати запис стрибаючи на потрібний фрейм;
- можливість програвати файли при неповному завантаженні з Інтернету;
- при перериванні завантаження MP3-файлу зберігається можливість програвання вже завантажених фреймів.

Для визначення об’ємів аудіоданих використовують поняття бітрейту. Бітрейт – це ширина полоси пропускання, що вимірюється у кількості біт на одну секунду [7]. Максимальний бітрейт MP3-формату 320 кбіт/с. При використанні даного значення у кодуванні використовуються виключно математичні

алгоритми ущільнення, тому якість звуку у даному випадку не падає. Саме тому файли формату MP3 досить погано ущільняються звичайними архіваторами.

При використанні менших значень бітрейту, використовується квантування – процес видалення звуків заснований на особливостях сприйняття звуку людиною [5]. Чим менше значення бітрейту, тим більше компонентів звукового сигналу буде видалено.

Ще одним популярним форматом є FLAC [140]. FLAC – це відкритий кодек, призначений для ущільнення аудіо-даних без втрат. Він ущільнює аудіо-дані не так ефективно як MP3, але якість звуку залишається незмінною порівняно з оригіналом. Одним з недоліків даного формату є порівняно висока обчислювальна складність алгоритму оброблення даних при формуванні файлу.

Зважаючи на всі переваги та недоліки описаних аудіо-форматів, для вбудовування конфіденційної інформації був обраний формат WAV. Теоретично, будь-який формат можна конвертувати у WAV за допомогою аудіо-конверторів [119]. Проте, наприклад, конвертація MP3-файлу у WAV-файл немає сенсу, оскільки інформація, необхідна для повноцінного звучання, вже була втрачена при попередній конвертації у MP3.

Такі формати, як FLAC, Apple Lossless, Monkey’s Audio та інші формати з ущільненням даних без втрат, також можуть виступати як контейнер при застосуванні методу на основі процедури псевдовипадкового вбудовування за умови попередньої конвертації у WAV-формат. Після вбудовування заповнений контейнер можна конвертувати у початковий аудіо-формат. Після цього декодування стегоданих залишається можливим з файлу у початковому форматі.

Аудіо-файли у форматах з ущільненням даних з втратами не підходять як контейнер для вбудовування, оскільки навіть при конвертації такого файлу у WAV, вбудовуванні стегодані можуть бути втрачені.

Метод на основі процедури псевдовипадкового вбудовування, що пропонується, є комплексним методом стеганографічного захисту мультимедійних даних у WAV-файлах [39, 46]. Метод використовує природні

властивості аудіо-даних, а саме, їх надлишковість [100]. Він відноситься до категорії крипто-стеганографічних методів, що використовують ключі для процесу вбудовування та декодування стегоданих. Ключ є комплексним та складається з наступних компонентів:

1. Довільне ціле невід'ємне число у проміжку [0, 4294967295], що виступає як «зерно» для першого генератора псевдовипадкових чисел.
2. Довільний файл.
3. Розмір повідомлення у байтах.
4. Логічна функція (не є обов'язковою, за замовчуваннями використовується сума за модулем 2).
5. Довільне ціле невід'ємне число у проміжку [0, 4294967295], що виступає як «зерно» для другого генератора псевдовипадкових чисел (не є обов'язковим, за замовчуванням використовується значення з п.1).

Цей метод є модифікацією LSB-методу у якому замінюються останні n бітів на біти повідомлення, де n – кількість байт на один семпл одного каналу. У мережі Інтернет, найбільш розповсюдженими є WAV-файли з двома каналами (стерео). Оскільки кожен семпл каналу найчастіше описується двома байтами, то при реалізації методу на основі псевдовипадкового вбудовування було взято $n = 2$. Отже, вбудовування відбувається у два наймолодші біти кожного семплу. Таким чином можна зробити висновок, що у якості контейнера для поточної реалізації розробленого методу можуть виступати WAV-файли, де кожен семпл одного каналу описується двома байтами. Але реалізація методу може бути модифікована таким чином, що контейнером може виступати WAV-файл с будь-якою кількістю каналів та байтів, що описують 1 семпл.

На першому етапі файл, що вбудовується, конвертується у масив байтів S. Реалізовано клас, що містить у собі масив байт та може надати доступ до будь-яких n бітів поточного масиву байт як для читання, так і для запису.

Наступним кроком, створюється масив U:

$$u = \frac{8 \cdot s}{n}, \quad (2.10)$$

де u – розмір масиву U;

s – розмір масиву байт S файлу, що вбудовується;

n – кількість змінюваних біт.

Після створення масиву U розміром u він заповнюється індексами елементів (0, 1, 2, …, u -1). Далі масив U перемішується за допомогою генератора псевдовипадкових чисел (ГПВЧ) [90]. У цьому методі застосовується вихор Мерсенна (Mersenne Twister) [107]. Цей ГПВЧ заснований на властивості простих чисел та підходить для вирішення більшості завдань. Порівняно з іншими аналогами, цей генератор забезпечує високу швидкодію [67], що є важливим для підвищення швидкодії програмного оброблення даних користувача. Також, обраний ГПВЧ не має таких недоліків, як:

- малий період;
- легко можна виявити статистичні закономірності;
- передбачуваність.

Після перемішування елементів у масиві U отримується результуючий масив індексів блоків по n бітів повідомлення перемішаних у псевдовипадковому порядку. Іншими словами, будь-який елемент у масиві U є індексом n бітів стегоданих.

Далі відбувається зчитування контейнеру у масив байт C. Після чого здійснюється зчитування метаданих WAV-файлу. Далі зберігається позиція байту, з якого починаються семпли аудіо-файлу.

Кількість семплів у контейнері визначається за формулою:

$$c_1 = \frac{c - m}{b}, \quad (2.11)$$

де c_1– загальна кількість семплів у контейнері у всіх каналах;

c – розмір контейнера у байтах;

m – розмір метаданих контейнера у байтах;

b – кількість байт у одному семплі.

Одним з найважливіших критеріїв стеганографічного захисту є унікальність контейнеру. Це пов’язано з тим, що якщо у стегоаналітика при перехопленні заповненого контейнера виникне підозра у тому, що у ньому приховані стегодані, і контейнер не буде унікальним, то стегоаналітик зможе знайти порожній контейнер, наприклад, в Інтернеті, та побайтово порівняти заповнений та порожній контейнери. Байти, що будуть відрізнятися і є байтами з даними. Знаючи кількість змінених байт можна підрахувати загальний об’єм конфіденційних даних.

На сьогодні у більшості користувачів мережі Інтернет є мобільні телефони з фотокамерою, що надає змогу робити якісні знімки з великою роздільністю. Тож з точки зору передачі конфіденційної інформації у графічних контейнерах, проблема унікальності контейнера вирішується фотографуванням будь-якого об’єкту з фотокамери. У випадку передачі конфіденційної інформації у аудіоконтейнерах ситуація складніша. Можна записати на диктофон мобільного телефону мелодію, текст або іншу інформацію та вбудувати стегодані туди, але зараз невелика кількість людей обмінюється аудіо-повідомленнями як файлами. Більшість популярних месенджерів підтримують обмін аудіо-повідомленнями, але приховати туди дані є досить непростою задачею для середньостатистичного користувача, оскільки саме аудіо-повідомлення має свій власний формат для кожної реалізації месенджера та здобувати його з месенджера у необробленому вигляді для вбудовування звичайно є неможливим. Через даний факт передати заповнений контейнер набагато простіше як аудіо-файл, а не як аудіо-повідомлення. Тож запропонований метод передбачає можливість використання у ролі контейнера WAV-файлів з мережі Інтернет, а саме випадок, якщо у стегоаналітика буде існувати порожній контейнер для побайтового порівняння його з заповненим.

Метод на основі процедури вбудовування повідомлення у псевдовипадкові семпли контейнера, окрім вбудовування самих стегоданих, додатково вбудовує шум у всі незаповнені семпли контейнера. Таким чином, при побайтовому порівнянні порожнього та заповненого контейнера, майже всі байти будуть змінені, тож декодувати повідомлення або хоча б визначити його розмір стає складною проблемою для стегоаналітика.

Наступним кроком перевіряється можливість вбудовування стегоданих у обраний контейнер:

$$\frac{8 \cdot s}{n} \leq c_1, \tag{2.12}$$

де c_1– кількість семплів у контейнері у всіх каналах;

s – розмір масиву байт S файлу, що вбудовується;

n – кількість змінюваних біт.

Якщо нерівність (2.12) виконується, то файл можливо вбудувати у контейнер. В іншому випадку необхідно обрати WAV-контейнер більшого розміру.

Таким чином, наступним кроком, відбувається процес ітерування по семплам контейнера та заміні останніх n бітів кожного семплу на біти повідомлення або псевдовипадкові біти для вбудовування шуму для підвищення стійкості. Ітерування відбувається наступним чином: перший семпл першого каналу; перший семпл другого каналу; другий семпл першого каналу; другий семпл другого каналу і т.д.

Прийняття рішення про вбудовування у поточний семпл конфіденційної інформації чи шуму відбувається за результатом розв’язання нерівності:

$$\frac{randTo(c_1)}{c_1} < \frac{u-j}{c_1-i}, \tag{2.13}$$

де c_1– кількість семплів у контейнері у всіх каналах;

$randTo(c_1)$– псевдовипадкове число у проміжку [0, c_1);

u– розмір масиву U;

i– індекс поточного семплу;

j– кількість вбудованих блоків по n бітів повідомлення.

Функція $randTo(c_1)$ використовує другий ГПВЧ. За замовчуванням використовується лінійний конгруентний метод [21]. Як зерно може використовуватись ключ. У випадку відсутності даного ключа використовується зерно з першого ГПВЧ, що використовувався для перемішування індексів блоків файлу.

У випадку виконання нерівності вбудовуються n бітів повідомлення у поточний семпл. В іншому випадку генеруються n псевдовипадкових бітів та вбудовуються як шум.

Наступним кроком файл, що виступає у ролі ключа, конвертується у масив бітів. До файлу-ключа немає жодних вимог, тож ключем може бути графічні дані, аудіодані, текстові файли та ін. будь-якого розміру.

Саме вбудовування даних за принципом схоже на вбудовування у методі на основі схеми відповідності бітів. У обох методах вбудовується результат обраної логічної функції. Але у методі на основі схеми відповідності бітів як аргументи логічної функції використовується біт секретного зображення та старший незмінний біт певної компоненти поточного пікселя. У даному ж випадку як аргументи виступають біт секретного мультимедійного файлу та біт файлу-ключа.

У випадку, якщо файл, що приховується більший за файл-ключ, то зчитування біту з файлу-ключа починається спочатку. Таким чином, файл-ключ може мати довільний розмір. Не рекомендується обирати малий файл-ключ, оскільки існує ймовірність того, що стегоаналітик зможе виявити певну закономірність.

За логічну функцію може бути обрана одна з функцій з табл. 2.2. За замовчуванням використовується сума за модулем 2.

Після вбудовування заповнений контейнер зберігається та є готовим до передачі отримувачу. Не рекомендується передавати заповнений контейнер,

файл-ключ, логічну функцію та зерна для генераторів псевдовипадкових чисел одним каналом зв'язку з метою мінімізації одночасного потрапляння заповненого контейнера та всіх ключів стегоаналітику.

Блок-схема процесу вбудовування конфіденційного файлу методом на основі процедури псевдовипадкового вбудовування зображена на рис. 2.15.

На етапі декодування починається зворотній процес. Отримувач має як один з ключів розмір конфіденційного файлу у байтах. Отже, спочатку створюється масив U_1розміром u_1:

$$u_1 = \frac{8 \cdot s_1}{n}, \tag{2.14}$$

де u_1 – розмір масиву U_1;

s_1 – розмір конфіденційного файлу у байтах;

n – кількість змінюваних біт.

На наступному кроці масив U_1 заповнюється індексами елементів. За допомогою другого ключа – зерна для першого ГПВЧ, масив U_1перемішується. Далі створюється бітовий масив S_1 розміром $s_1 \cdot 8$ для запису декодованих бітів.

Після цього, проходячи по семплам заповненого контейнера, зчитуються біти конфіденційної інформації та записуються до масиву S_1. Для розрізнювання семплів з корисною інформацією та зашумлених, на кожній ітерації відбувається перевірка нерівності (2.15).

$$\frac{randTo(c_{11})}{c_{11}} < \frac{u_1 - j_1}{c_{11} - i_1}, \tag{2.15}$$

де c_{11}– кількість семплів у заповненому контейнері у всіх каналах;

$randTo(c_{11})$– псевдовипадкове число у проміжку [0, c_{11});

u_1– розмір масиву U_1;

i_1– індекс поточного семплу;

j_1– кількість зчитаних блоків по n бітів повідомлення.

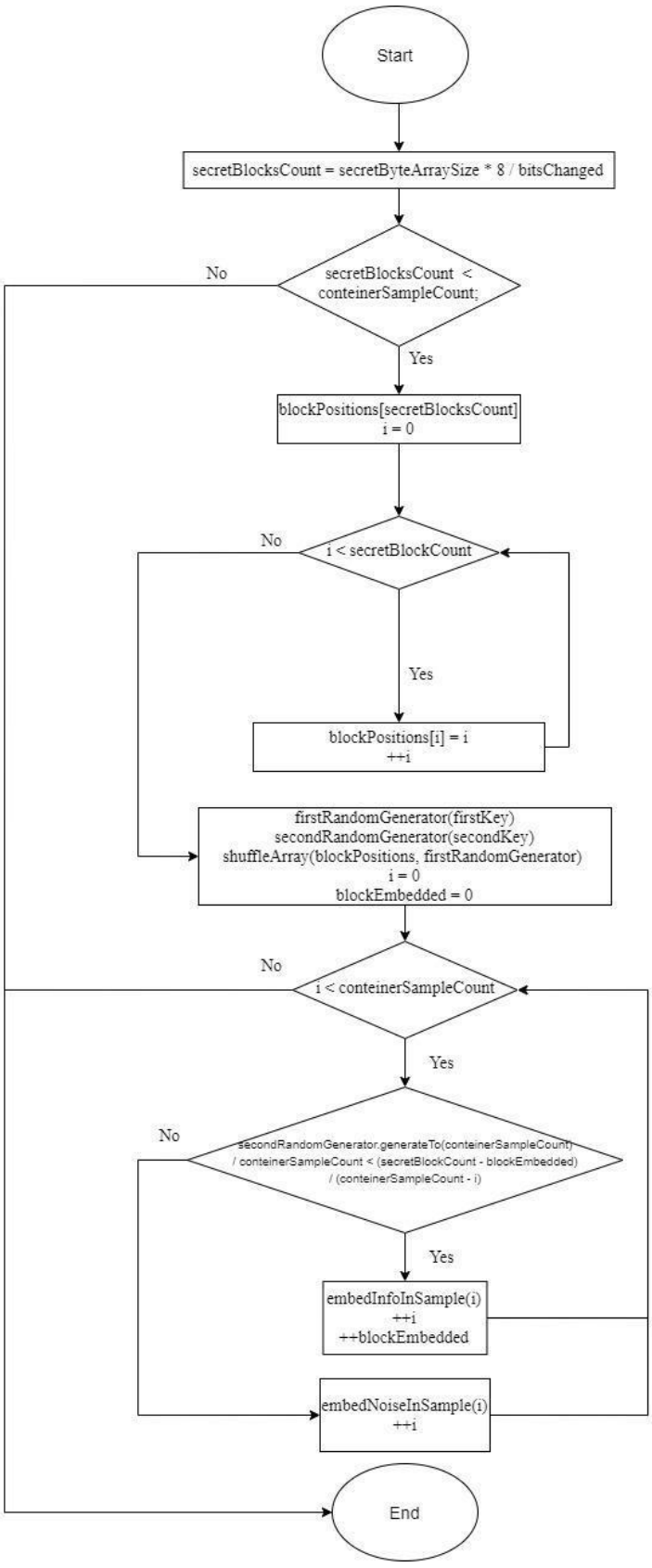

Рис. 2.15 Блок-схема процесу вбудовування методом на основі процедури псевдовипадкового вбудовування

На наступному кроці файл, що виступає у ролі ключа, конвертується у масив бітів з можливістю послідовного побітового зчитування. У разі виконання нерівності (2.15), з поточного семплу зчитуються дані.

Зчитування відбувається за допомогою логічної функції, що виступає ключем, або за допомогою функції сума за модулем два у разі відсутності ключа. Як аргументи функції виступають декодований біт конфіденційного файлу та біт з файлу-ключа. Результат логічної функції записується у створений бітовий масив S_1.

Після заповнення масиву S_1, він конвертується у байтовий масив та зберігається на носій як файл. Збережений файл містить відновлені мультимедійні дані користувача.

2.4 Висновки до розділу 2

У цьому розділі розроблено алгоритмічне забезпечення процесу захисту мультимедійних даних, яке спрямоване на зменшення часу оброблення даних, зменшення ймовірності підбору ключів та підвищення ефективності захисту даних від статистичних стегоатак.

Зокрема, отримано такі результати.

1. Процедура захисту на основі схеми відповідності бітів забезпечує підвищення ефективності оброблення та захисту мультимедійних даних за рахунок використання логічних операцій над старшими бітами компонент R, G та B контейнера та бітами конфіденційних графічних даних. Може бути адаптована для використання аудіо-контейнерів та об’єднана з існуючими методами LSB-стеганографії для підвищення їх стеганографічної стійкості.

2. Процедура захисту на основі дерева Хаффмана дозволяє забезпечити захист графічних даних за рахунок вбудовування композиції стегоданих та зображення, що виступає ключем.

3. Процедура захисту на основі псевдовипадкового вбудовування забезпечує неможливість декодування стегоданих за відсутності комплексного секретного ключа, що складається зі змінної кількості компонентів, кожна з яких може бути передана різними каналами зв’язку. Дозволяє вбудовувати дані користувацьких файлів поширених форматів мультимедійних даних.

РОЗДІЛ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПРОЦЕСІВ ОБРОБЛЕННЯ ДАНИХ ДЛЯ КРИПТО-СТЕГАНОГРАФІЧНОГО ЗАХИСТУ МУЛЬТИМЕДІЙНИХ ДАНИХ

3.1 Універсальна архітектура програмної системи захисту мультимедійних даних

Для забезпечення виконання вимог до програмного забезпечення крипто-стеганографічного захисту мультимедійних даних користувачів мережі Інтернет, що були сформульовані у розділі 1, пропонується універсальна архітектура програмної системи, що зображена на рис. 3.1.

Компоненти цієї архітектури можна розділити на чотири групи.

Перша група компонентів реалізує інтерфейс користувача (UI). Графічний інтерфейс користувача може розроблений засобами QML або шляхом створення нестандартного UI;

Друга група компонентів реалізує модуль логіки (Logical Unit), що являє собою набір класів для обміну даними між користувачем та методами вбудовування / декодування.

Третя група компонентів реалізує модуль кодування (Code Method Unit). Цей модуль реалізує процедури вбудовування стегоданих.

Четверта група компонентів реалізує модуль декодування (Decode Method Unit). Цей модуль реалізує процедури відновлення даних.

Таким чином, основними модулями програмної системи, які забезпечують безпосередню роботу з мультимедійними даними користувача, є модуль кодування та модуль декодування. Ці модулі реалізують певний набір методів стеганографічного захисту мультимедійних даних.

При виконанні тестового розроблення програмної системи за запропонованою універсальною архітектурою як базовий набір методів захисту використовувались наступні методи:

- LSB-метод;
- метод на основі схеми відповідності бітів;
- метод на основі дерева Хаффмана;
- метод на основі процедури псевдовипадкового вбудовування.

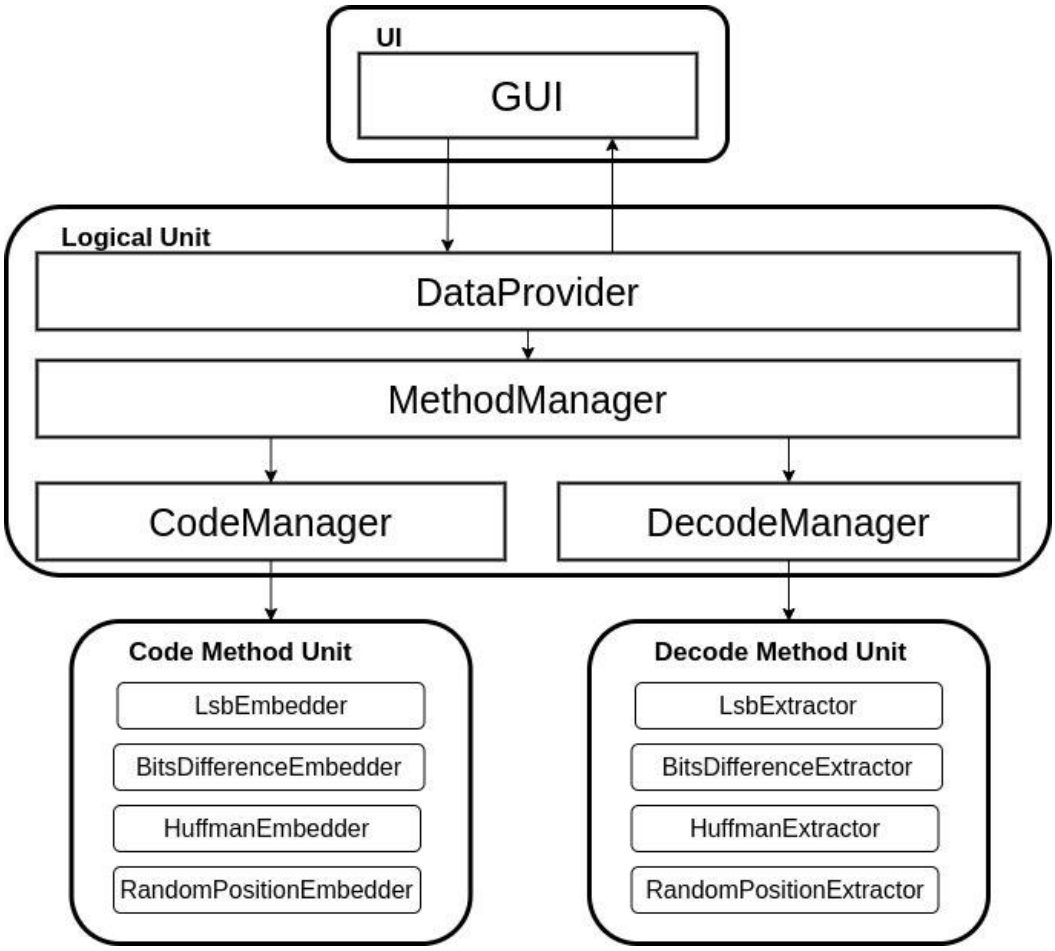

Рис. 3.1 Універсальна архітектура програмної системи

Кожен метод реалізований у власному класі. На кожен тип контейнера (зображення або аудіодані), що підтримує метод, створено окремий клас. Наприклад, вбудовування конфіденційних даних класичним LSB методом представлено абстрактним класом *LsbEmbeder*, від якого успадковані класи *AudioLsbEmbeder* та *ImageLsbEmbeder*, що використовуються залежно від типу контейнера. Для декодування даних реалізований абстрактний клас *LsbExtractor*, від якого успадковані *AudioLsbExtractor* та *ImageLsbExtractor*.

Загальну схему успадкування класів у розробленому програмному забезпеченні представлено на рис. 3.2.

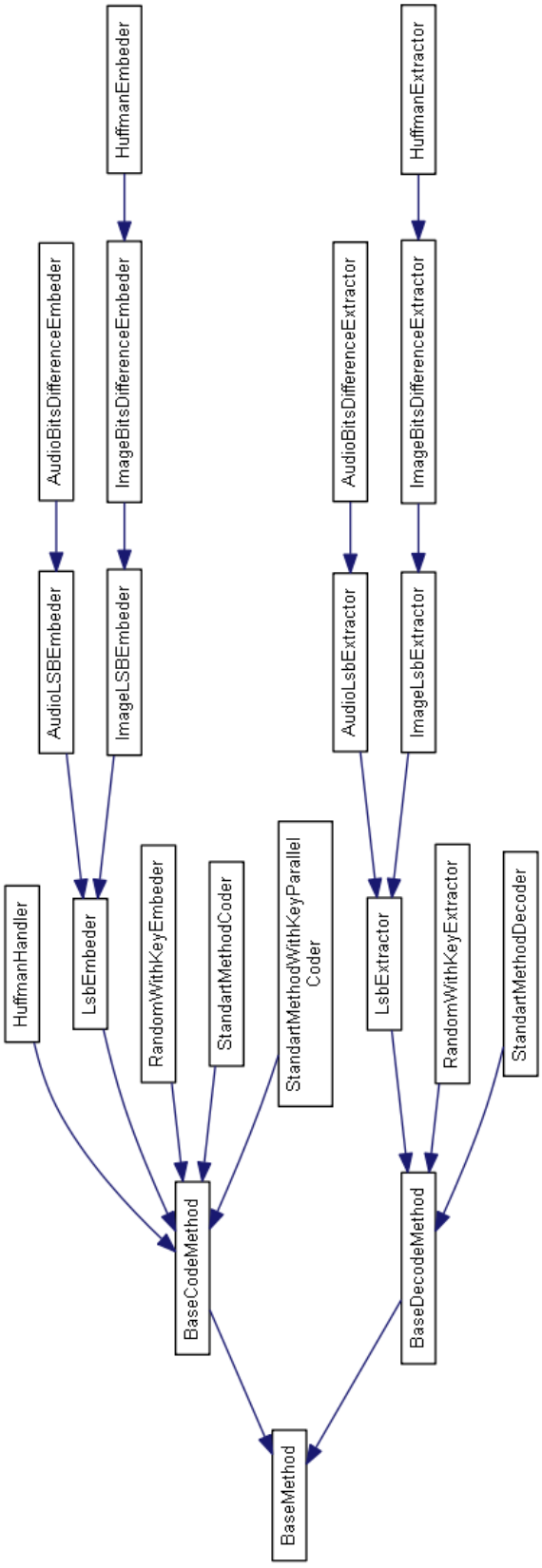

Рис. 3.2 Схема успадкованості класів

Усі реалізовані методи успадковуються від базового класу *BaseMethod* та зберігають шлях та ім’я заповненого контейнеру та ключ. Ці два параметри є присутніми у кожному методі, незалежно від того, вбудовує метод інформацію чи декодує. Також даний клас має один чистий віртуальний метод *virtual void initialize() = 0* для ініціалізації параметрів методу [77].

Від *BaseMethod* успадковуються класи *BaseCodeMethod* та *BaseDecodeMethod*. Перший має два допоміжні поля: шлях та ім’я порожнього контейнера та стегоданих. Також даний клас має чистий віртуальний метод *virtual bool code() = 0*, що перевизначається у дочірніх класах.

BaseDecodeMethod має поля, що визначають шлях та ім’я потенціального декодованого файлу. Також, як і *BaseCodeMethod*, даний клас має один чистий віртуальний метод *virtual bool decode() = 0.*

Розроблені методи у програмному коді мають наступні назви:

1. LSB-метод – *LsbEmbeder* та *LsbExtractor* для вбудовування та декодування конфіденційних даних відповідно з префіксом *Audio* або *Image* залежно від типу контейнера.
2. Метод на основі схеми відповідності бітів – *BitsDifferenceEmbeder* та *BitsDifferenceExtractor* для вбудовування та декодування конфіденційних даних відповідно з префіксом *Audio* або *Image* залежно від типу контейнера.
3. Метод на основі дерева Хаффмана – *HuffmanEmbeder* та *HuffmanExtractor* для вбудовування та декодування конфіденційних даних відповідно у графічні контейнери.
4. Метод на основі процедури псевдовипадкового вбудовування – *RandomWithKeyEmbeder* та *RandomWithKeyExtractor* для вбудовування та декодування конфіденційних даних відповідно у аудіоконтейнери.

Діаграма класів для базових методів зображена на рис. 3.3.

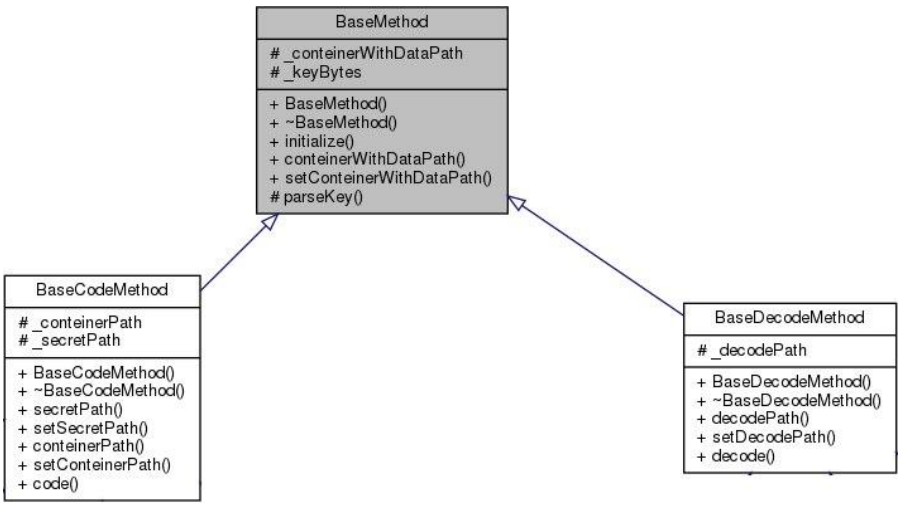

Рис. 3.3 Діаграма класів базових класів для всіх реалізованих методів

Діаграми класів для методів вбудовування та декодування зображені на рис. 3.4 та рис. 3.5 відповідно [25]. Запропонована архітектура дозволяє досить просто додавати нові крипто-стеганографічні методи, як для вбудовування стегоданих, так і для їх зворотного декодування.

На рис. 3.6 зображена діаграма послідовностей. *DataProvider* – абстрактний клас для обміну даними між користувачем та логічним модулем, що відповідає за вбудовування та декодування стегоданих (лістинг 3.1). Основною задачею даного класу є робота з *MethodManager* для передачі даних від користувача у методи. У поточній реалізації клас *QmlDataProvider* успадкований від *DataProvider*, оскільки програмний додаток має GUI реалізоване засобами QML. Отже, запропонована архітектура побудована таким чином, щоб мінімізувати роботу програміста у випадку, якщо програмний додаток треба розширити для постачання даних не через вікно програми, а наприклад, через сервіс. Для цього потрібно створити новий клас успадкований від *DataProvider* та перевизначити необхідні методи.

Таким чином, можна зробити висновок, що поточна реалізація програмного додатку легко масштабується для випадків:

- додавання нових методів;
- введення/виведення даних для роботи методів.

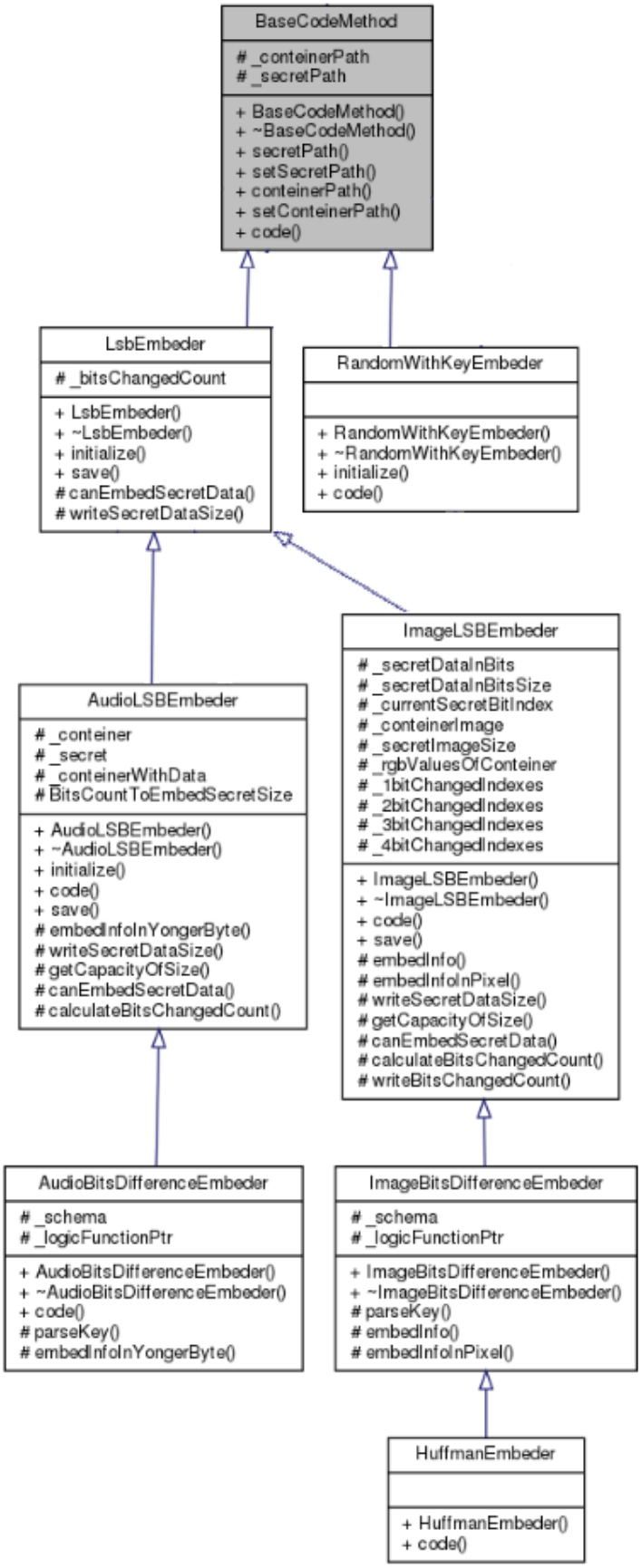

Рис. 3.4 Діаграма класів для реалізованих методів вбудовування

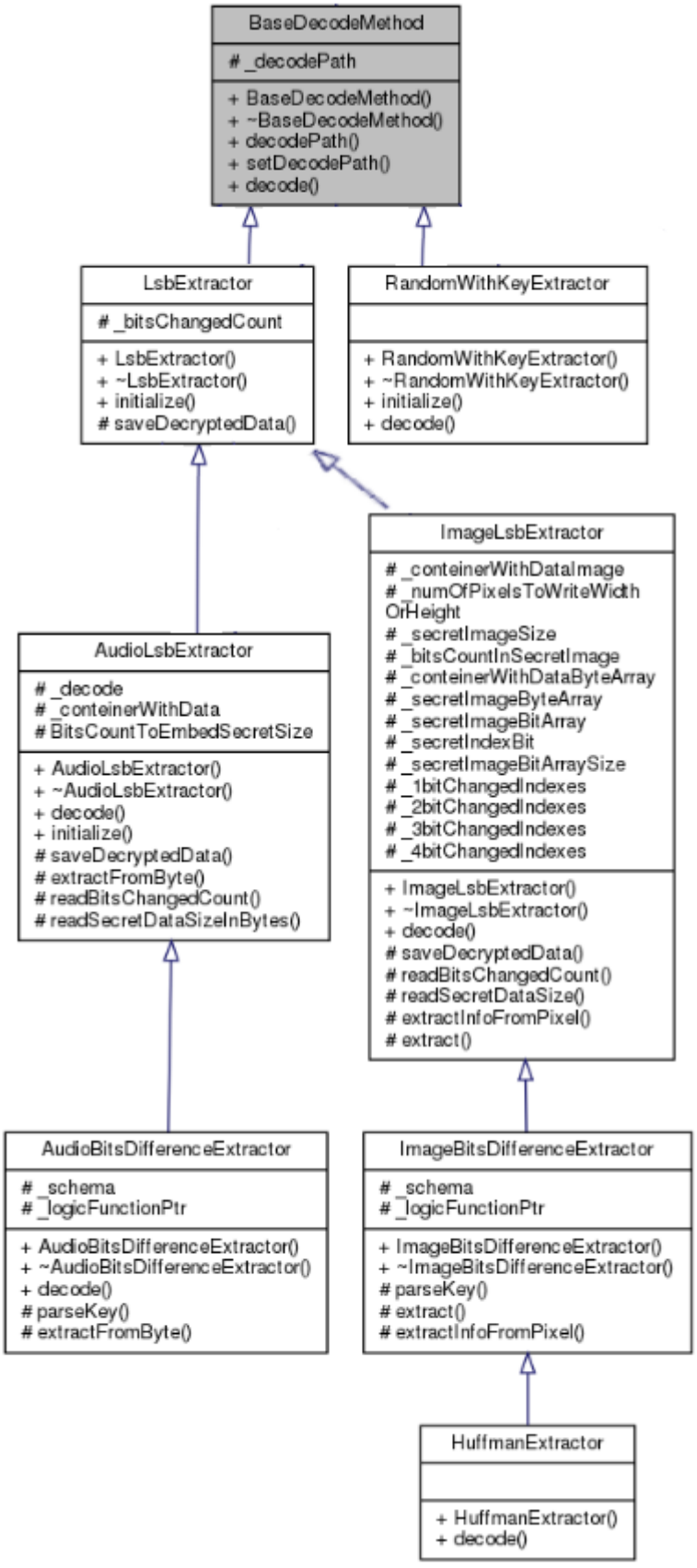

Рис. 3.5 Діаграма класів для реалізованих методів декодування

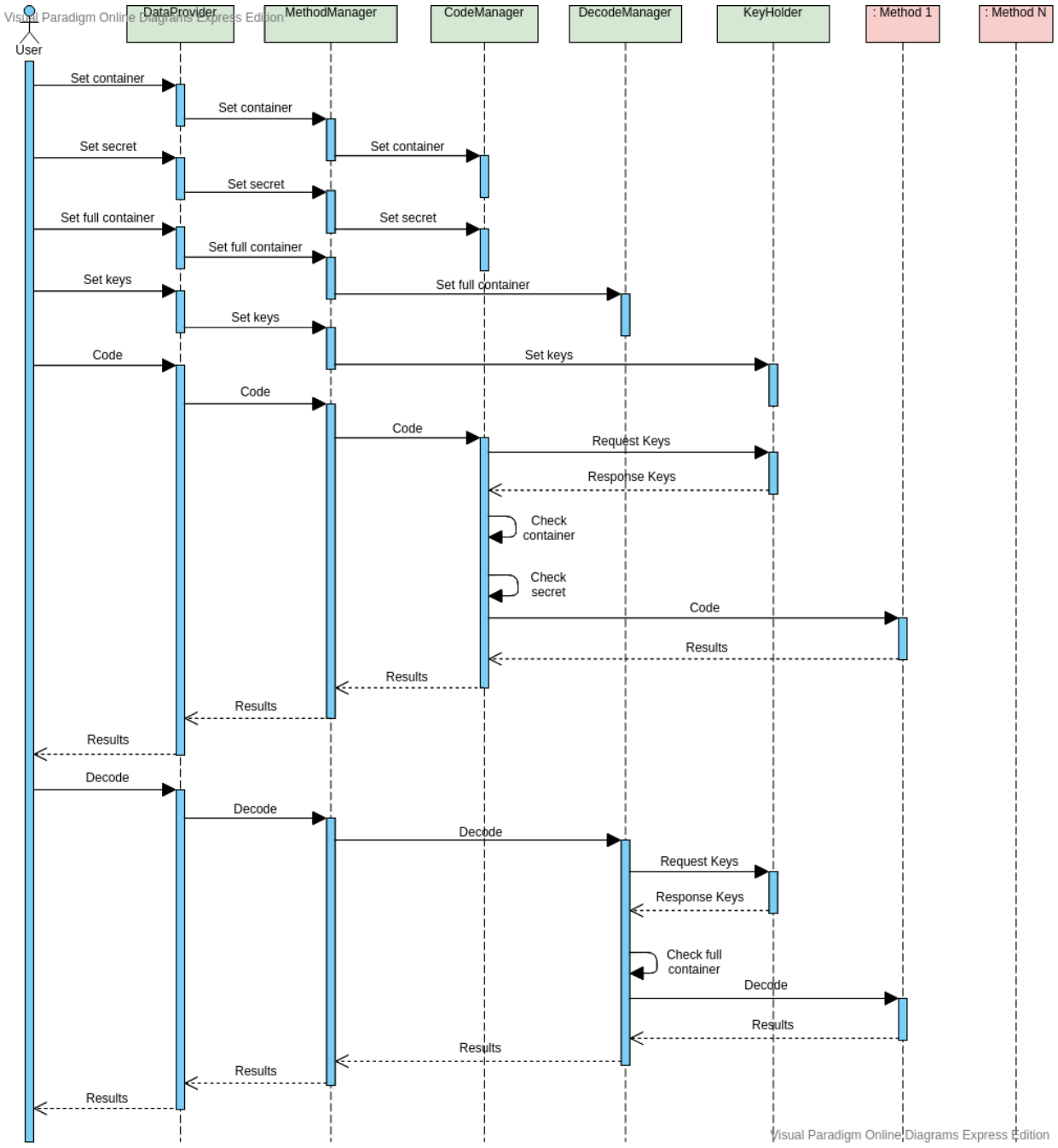

Рис. 3.6 Діаграма послідовності

При розробленні програмного забезпечення були використані наступні шаблони проектування:

- Singleton [36];
- Адаптер [141];
- Фасад [10];

- Компонувальник [30].

Лістинг 3.1. Інтерфейс для обміну даними.

```
class DataProvider
{
public:
    virtual void code() = 0;
    virtual void decode() = 0;

    virtual QString containerPath() const;
    virtual void setContainerPath(QString containerPath);
    virtual QString secretPath() const;
    virtual void setSecretPath(QString secretPath);
    virtual QString fullContainerPath() const;
    virtual       void       setFullContainerPath(QString
fullContainerPath);
    virtual QString decodePath();
    virtual bool coverIsAudio() const;
protected:
    MethodManager _methodManager;
};
```

Більшість використаних шаблонів є структурними, що дозволяє полегшити розроблення і оптимізувати програмний додаток.

Для розроблення GUI було обрано QML. QML – декларативна мова програмування для розроблення інтерфейсу користувача заснована на JavaScript. Для реалізації методів була обрана мова програмування C++ [45], оскільки вона має ряд переваг перед іншими:

- кросплатформеність [6];
- висока швидкодія;
- підтримка багатьох стилів програмування (структурне, об’єктно-орієнтоване, функціональне) [24];
- шаблони узагальнення алгоритмів для використання різних типів даних;
- поширеність (велика кількість літератури, документації, прикладів тощо).

Для крипто-стеганографічних методів ключем можуть виступати різні формати даних: число, текст, зображення, файл та ін. Усі ці формати можна представити як масив байтів. Виходячи з цього, кожен ключ, що буде використовуватись у додатку, повинен успадкувати клас *BaseKey* та перевизначити чисто віртуальний метод *std::vector<uint8_t>& toBytes()*. Узагальнюючий інтерфейс ключа зображено у лістингу 3.2.

Лістинг 3.2. Інтерфейс ключа.

```
class BaseKey
{
public:
    BaseKey() = default;
    virtual ~BaseKey() = default;
    virtual std::vector<uint8_t>& toBytes() = 0;
protected:
    std::vector<uint8_t> _bytes;
};
```

Для перевірки запропонованої універсальної архітектури в рамках дисертаційного дослідження було розроблено програмну систему, інтерфейс якої зображено у Додатку Г (рис. Г.1). Головне вікно програмного засобу умовно поділене на 5 зон:

1. Зона вибору режиму роботи (вбудовування або декодування).
2. Зона з кнопками для вибору порожнього контейнера, стегоданих та старту процесу вбудовування/декодування.
3. Зона відображення порожнього контейнера зліва та стегоданих справа у режимі вбудовування або заповненого контейнера зліва та декодованих стегоданих справа у режимі декодування.
4. Зона відображення повідомлень програми.
5. Зона для вибору та налаштування методів.

Зони 1, 4 та 5 є спільними для режиму вбудовування та декодування.

Така організація інтерфейсу дозволяє спростити роботу користувача з програмною системою.

3.2 Програмне забезпечення процесів оброблення даних для крипто-стеганографічного захисту графічних даних методом на основі схеми відповідності бітів

Для тестування було реалізовано класичний LSB-метод, а саме класи *LsbEmbeder* та *LsbExtractor*, що успадковані від *BaseCodeMethod* та *BaseDecodeMethod* відповідно. Реалізація розробленого методу досить схожа на реалізацію класичного LSB, тому клас, що реалізує метод на основі схеми відповідності бітів, успадковується від реалізації класичного LSB (лістинг 3.3).

Лістинг 3.3. Клас, що реалізує метод на основі схеми відповідності бітів.

```
class ImageBitsDifferenceEmbedder: public ImageLSBEmbeder
{
public:
    ImageBitsDifferenceEmbedder(...);
    virtual ~ImageBitsDifferenceEmbedder() override;
protected:
    bool embedInfoInPixel(...) override;
};
```

Розглянемо приклад роботи методу на основі схеми відповідності бітів. Нехай дані наступні вхідні дані:

- зображення cover.png з роздільністю 1024x1024 пікселів як контейнер (рис. Ґ.1);
- зображення secret.png з роздільністю 225x225 пікселів як конфіденційні графічні дані (рис. Ґ.2);

- логічна функція не задана у явному вигляді (за замовчуванням використовується сума за модулем 2);
- схема відповідності молодших і старших не задана у явному вигляді (використовуються схема за замовчуванням залежно від кількості змінюваних біт).

Виходячи з вхідних даних можна зробити висновок, що відправник та одержувач конфіденційних графічних даних домовились між собою не використовувати ключі для даного методу. В іншому випадку, при вбудовуванні та декодуванні можна змінити номер логічної функції та обрати створену самостійно схему відповідності бітів. Розроблена схема повинна бути збережена до файлу та мати розширення .asc для аудіоконтейнерів та .isc для графічних контейнерів відповідно.

Файл-ключ з розширенням .isc складається з 4 блоків, що розділені пустим рядком між собою:

1. Схема при зміні одного молодшого біту, складається з 3-х рядків.
2. Схема при зміні двох молодших бітів, складається з 6 рядків.
3. Схема при зміні трьох молодших бітів, складається з 9 рядків.
4. Схема при зміні чотирьох молодших бітів, складається з 12 рядків.

Отже загалом, розмір файлу-ключа з розширенням .isc складатиме 3 + 1 + 6 + 1 + 9 + 1 +12 + 1 = 34 рядки.

Кожен піксель описується 24 бітами (по 8 біт на кожну компоненту R, G та B). Звідси, якщо представити всі 3 компоненти у вигляді 24-х бітного числа, то при нумерації з 0, 16 біт даного числа буде молодшим бітом компоненти R, 8-й – молодший біт компоненти G, 0-й – молодший біт компоненти B відповідно. Таким чином, у файлі описується зв’язок молодших і старших бітів через пробіл у наступному форматі:

16 5

8 7

0 14

Файл-ключ з схемою відповідності для аудіофайлів заповнюється за аналогією та матиме менший об’єм, оскільки кожний семпл складається з 16 бітів.

Після вибору ключів треба проаналізувати можливість вбудовування конфіденційних даних у даний контейнер. Для даної задачі було реалізовано метод *canEmbedSecretData()* (лістинг 3.4).

Лістинг 3.4. Реалізація методу *canEmbedSecretData()*.

```
bool ImageLSBEmbeder::canEmbedSecretData() const
{
    uint coverAllPixelsCount = _conteinerImage.width() *
                          _conteinerImage.height();
    uint maxSecretImagePixelCount = coverAllPixelsCount /
    2;
    uint secretImagePixelCount = _secretImageSize.width()
*                     _secretImageSize.height();

    return secretImagePixelCount <
maxSecretImagePixelCount; }
```

Контейнер складається з 1024 * 1024 = 1048576 пікселів. Кожен піксель складається з трьох компонент R, G та B кожна з яких займає один байт. Виходячи з цього, масив байт контейнера буде складатися з 1048576 * 3 = 3145728 байтів.

Метод на основі схеми відповідності бітів дозволяє змінювати до чотирьох бітів у кожному байті. Оскільки байт складається з восьми бітів, то можна зробити висновок, що у кожний байт контейнера можна вбудувати половину байту зображення, що вбудовується. Це означає, що максимальна кількість байтів даного зображення дорівнює 3145728 / 2 = 1572864, а максимальна кількість пікселів:

$$\frac{1572864}{3} = 524288_{(10)} = 10000000000000000000_{(2)}$$

Для запису $10000000000000000000_{(2)}$при зміні одного останнього біту потрібно 20 байтів. Оскільки кожен піксель складається з трьох байтів, то потрібно 7 пікселів для запису ширини або висоти повідомлення.

Отже, максимальна кількість пікселів для запису значень компонент R, G та B вбудованого зображення дорівнює 524288 - 7 - 7 - 1 = 524273.

Кількість пікселів у конфіденційних графічних даних secret.png дорівнює 225 * 225 = 50625, що менше 524273. Виходячи з цього можна зробити висновок, що дане зображення можливо вбудувати у контейнер.

За формулою (2.2) виконується нерівність:

$$50625 < \frac{524273}{8}$$

Це означає, що стегодані можна вбудувати зі зміною одного наймолодшого біту ($b = 1_{(10)} = 001_{(2)}$).

Наступним етапом у перший піксель контейнера вбудовується значення b за схемою відповідності молодших і старших бітів зображеною на рис. 2.2. У даному випадку перший піксель контейнера характеризується наступними значеннями компонент:

$$\text{R} = 189_{(10)} = 1011\ 1101_{(2)}$$

$$\text{G} = 122_{(10)} = 0\mathbf{1}11\ 1010_{(2)}$$

$$\text{B} = 103_{(10)} = \mathbf{0}11\mathbf{0}\ 0111_{(2)}$$

Напівжирним шрифтом виділено старші біти, що згідно зі схемою за замовчуванням на рис. 2.2 беруть участь як один з двох аргументів у логічній функції додавання за модулем 2. Другим аргументом виступатиме перший біт b (001). Таким чином створюються пари для порівняння: на позицію R[0] - B[5] та 0; на позицію G[0] - B[7] та 0; на позицію B[0] - G[6] та 1.

Наступним етапом виконується логічна функція над обраними парами аргументів:

$$R[0] = B[5] \oplus 0 = 1 \oplus 0 = 1$$

$$G[0] = B[7] \oplus 0 = 0 \oplus 0 = 0$$

$$B[0] = G[6] \oplus 1 = 1 \oplus 1 = 0$$

Після виконання логічної функції над парами аргументів отримуємо наступний результат:

$$\text{R} = 189_{(10)} = 1011\ 1101_{(2)}$$

$$\text{G} = 122_{(10)} = 0111\ 1010_{(2)}$$

$$\text{B} = 102_{(10)} = 0110\ 011\mathbf{0}_{(2)}$$

Напівжирним шрифтом виділено біти, що змінилися. У даному випадку змінився лише один біт на позиції B[7]. Отже, перший піксель заповненого контейнера буде містити наступні компоненти R(189), G(122), B(102).

Наступним етапом вбудовується ширина та висота конфіденційних графічних даних. Ширина та висота вбудованого зображення дорівнюють $225_{(10)} = 11100001_{(2)}$пікселів. Для запису цих даних зарезервовано по 7 пікселів, але фактично, для того, щоб вбудувати роздільність повідомлення при зміні одного біту для даного випадку необхідно лише 3 пікселі на кожну компоненту (висота або ширина). У такому випадку, до двійкового представлення даних величин додаються на початок біти зі значенням 0 таким чином, щоб загальна кількість бітів складала 3 * 7 = 21. Після даної операції висота та ширина стегоданих виглядатиме наступним чином: $000000000000011100001_{(2)}$. У кожен піксель можна вбудувати 3 біти інформації, оскільки пікселів 7, то ширину та висоту треба привести до двійкового вигляду з 21 бітом.

За схемою відповідності старших та молодших бітів зображеною на рис. 2.2 та логічною функцією сума за модулем 2 двійкова послідовність $000000000000011100001_{(2)}$ вбудовується у перші 7 пікселів починаючи з другого (ширина) та в останні 7 пікселів (висота) обраного контейнера.

Далі, аналогічним чином, починається вбудовування значення компонент R, G та B зображення.

Після завершення процедури вбудовування, заповнений контейнер зберігається як файл та відправляється отримувачу.

Отримувач декодує отримане повідомлення наступним чином: з першого пікселя заповненого контейнера за схемою відповідності старших і молодших бітів зображених на рис. 2.2 та логічної функції сума за модулем 2 зчитується кількість бітів що були змінені.

Далі, за процедурою, що описана вище, визначається кількість пікселів, що зарезервовано під вбудовування роздільності зображення та відбувається зчитування цих даних.

На цьому етапі у отримувача повідомлення є вся важлива інформація для коректного зчитування значень компонент R, G та B вбудованого зображення. Тому створюється матриця пікселів розміром роздільності конфіденційних графічних даних (225x225) та заповнюється декодованими значеннями.

Після заповнення матриця зберігається як графічний файл, що і є прихованим мультимедійним повідомленням.

На рис. Ґ.7 видно, що порожній контейнер (зліва) та заповнений контейнер (справа) візуально не відрізняються.

3.3 Програмне забезпечення процесів оброблення даних для крипто-стеганографічного захисту графічних даних методом на основі дерева Хаффмана

Метод на основі дерева Хаффмана реалізований у двох класах: *HuffmanEmbeder* для вбудовування та *HuffmanExtractor* для декодування конфіденційних графічних даних. Дані класи успадковані від класів, що реалізують метод на основі схеми відповідності бітів. Це пов’язано з тим, що метод на основі дерева Хаффмана вбудовує бітовий масив кодів Хаффмана методом на основі схеми відповідності бітів. Тому деякі методи описаних вище класів перевизначені.

Класс *HuffmanEmbeder* описується наступним чином: лістинг 3.5.

Лістинг 3.5. Клас, що реалізує метод на основі дерева Хаффмана.

```
class HuffmanEmbeder: public ImageBitsDifferenceEmbeder
{
public:
    HuffmanEmbeder(...);
    bool code() override;
private:
    bool embedInfo() override;
    bool embedInfoInPixel(...) override;
    bool canEmbedSecretData() const override;
    uint8_t getCapacityOfSize() override;
    void writeHolePixelCount(...);
    //поля класу
};
```

Розглянемо приклад роботи методу на основі дерева Хаффмана. Нехай дані наступні вхідні дані:

- зображення cover.png з роздільністю 1300x1300 пікселів у якості контейнера (рис. Ґ.5);
- зображення secret.png з роздільністю 150x150 пікселів у якості конфіденційних графічних даних (рис. Ґ.6);
- зображення key.png з роздільністю 1600x1600 пікселів у якості ключа (рис. Ґ.4);
- логічна функція не задана у явному вигляді (за замовчуванням використовується сума за модулем 2);
- схема відповідності молодших і старших не задана у явному вигляді (використовуються схема за замовчуванням при зміні одного наймолодшого біту).

На першому етапі кодування будується дерево Хаффмана на основі ключа. Зображення, що виступає у ролі ключа зображено на рис. Ґ.4.

Дане зображення перевіряється на наявність усіх компонент у проміжку [0; 255]. Водночас з перевіркою, підраховується кількість кожної компоненти у зображенні. Воно містить усі компоненти.

Наступним етапом будується дерево Хаффмана та зберігається до файлової системи. Файл має розширення .hf (Huffman). Першим рядком файлу є шлях та назва зображення-ключа для якого побудоване дерево. Далі записуються відсортовані значення від 0 до 255 включно та через пробіл їх коди Хаффмана. При повторному використанні даного зображення-ключа програма зчитає побудоване дерево Хаффмана з файлу, що суттєво прискорить процес вбудовування/декодування графічних даних, адже побудова дерева займає значний час порівняно з самим процесом вбудовування/декодування.

Найбільш повторюваним значенням компоненти у ключі є 0. Воно повторюється 1067146 разів. Найменш повторюваним значенням є 108 (11734 повторень). Після побудови дерева Хаффмана, код Хаффмана для байту зі значенням 0 складається з 3 бітів (замість 8 як в звичайному LSB або методу на основі схеми відповідності бітів), а код для байту зі значенням 108 займає 9 бітів (замість тих же 8).

Виходячи з даних результатів, на перший погляд може здатися, що у контейнер методом на основі дерева Хаффмана можна вбудувати набагато більше інформації ніж звичайним LSB або методом на основі схеми відповідності бітів. Але це не завжди так. Отримані коди Хаффмана обчислюються виходячи з байтів ключа, а не конфіденційного зображення. Тож не факт, що у конфіденційному зображенні найбільш повторюваним значенням буде 0, а найменш повторюваним – 108. У даному випадку, розмір бітового масиву s_0 для вбудовування звичайним LSB обчислюється за формулою:

$$s_0 = w \cdot h \cdot 3 \cdot 8 = 150 \cdot 150 \cdot 24 = 540000$$

де s_0– розмір бітового масиву конфіденційного зображення;

w – ширина у пікселях конфіденційного зображення;

h – висота у пікселях конфіденційного зображення.

Розмір бітового масиву s_1 для вбудовування у контейнер отриманого за допомогою дерева Хаффмана дорівнює 564261. Виходячи з отриманих даних, розмір бітового масиву зріс приблизно на 4.5%. Можна зробити висновок, що розмір бітового масиву для вбудовування напряму залежить від схожості стегоданих та ключа.

Максимальний розмір повідомлення *m*, що можна вбудувати у даний контейнер дорівнює 633742 байти. Дане конфіденційне зображення *s* займає 564261 / 8 = 70532.625 байтів. Після округлення у більшу сторону отримується 70533 байти. Оскільки 70533 менше за 633742, то конфіденційне зображення можна вбудувати у поточний контейнер.

Цю функціональність реалізовано у методі *canEmbedSecretData*() (лістинг 3.6).

Лістинг 3.6. Реалізація методу *canEmbedSecretData*().

```
bool HuffmanEmbeder::canEmbedSecretData() const
{
    uint maxSecretImageBitsCount =
_conteinerImage.width() *
    _conteinerImage.height() * 3;
    return _secretDataInBitsSize <
maxSecretImageBitsCount;
}
```

Наступний кроком визначається за формулою (2.9) інтервал *i* у пікселях, через який буде вбудовуватись конфіденційне зображення:

$$i = toInt\left(\frac{m}{s}\right) - 1 = toInt\left(\frac{633742}{70533}\right) - 1 = 7$$

Отже, секретне повідомлення буде вбудоване у кожний 8 піксель контейнера. Значення інтервалу записується у методі *writeHolePixelCount()* (лістинг 3.7).

Лістинг 3.7. Реалізація методу *writeHolePixelCount()*.

```
writeHolePixelCount (uint
startPositionAfterWrittenWidthInByte, uint
holePixelCount)
{
    std::bitset<24> holeBitSet(holePixelCount);
    for(size_t i = startPositionAfterWrittenWidthInByte;
i <
    startPositionAfterWrittenWidthInByte +
                _MaxHolePixelCount * 3; i += 3)
    {
        if(_rgbValuesOfConteiner[i] & 1 != holeBitSet[i
-
    startPositionAfterWrittenWidthInByte])
        {
                _rgbValuesOfConteiner[i] ^= 1;
        }
        if(_rgbValuesOfConteiner[i + 1] & 1 !=
holeBitSet[i -
    startPositionAfterWrittenWidthInByte + 1])
        {
                _rgbValuesOfConteiner[i + 1] ^= 1;
        }
        if(_rgbValuesOfConteiner[i + 2] & 1 !=
holeBitSet[i -
    startPositionAfterWrittenWidthInByte + 2])
        {
                _rgbValuesOfConteiner[i + 2] ^= 1;
        }
    }
}
```

Далі за аналогією, як і в методі на основі схеми відповідності бітів, визначається кількість пікселів для резервування для запису максимальної висоти або ширини *w* конфіденційних графічних даних, що можна вбудувати у поточний контейнер. У даному випадку це 6 пікселів. Вбудовується значення

ширини повідомлення у перші 6 пікселів контейнера та значення висоти у останні 6 пікселів. Починаючи з 7-го пікселя вбудовується значення інтервалу *i* у наступні 8 пікселів.

Наступним етапом відбувається ітерування по масиву байт контейнера починаючи з позиції *p*:

$$p = 3 \cdot (w + h) = 3 \cdot (6 + 8) = 42,$$

де *w* – кількість пікселів для запису максимальної ширини або висоти конфіденційного зображення;

h – кількість пікселів для запису значення інтервалу (завжди дорівнює 8).

У три байти на позиціях *p* (42), *p* + 1 (43), *p* + 2 (44) вбудовується три біти повідомлення. Дані байти відповідають за компоненти R (значення байту 68), G (84) та B (107) 15-го пікселя. Перші три біти для вбудовування false, true, true. За допомогою схеми відповідності бітів за замовчуванням при зміні одного наймолодшого біту та логічної функції сума за модулем 2 отримується біт для вбудовування у компоненту R зі значення true.

$$68_{(10)} = 01000100_{(2)}$$

Видно, що молодший біт дорівнює значенню false. Тож до значення 68 треба додати або відняти 1. За допомогою псевдовипадкового числа у даному випадку обирається операція віднімання. Тож тепер значення компоненти R дорівнює 67. Оскільки у обраної схеми відповідності бітів жоден молодший біт не порівнюється зі старшими бітами компоненти R, то немає сенсу аналізувати нове значення.

$$84_{(10)} = 01010100_{(2)}$$

Молодший біт значення 84 (компонента G) дорівнює false, а біт, що треба вбудувати дорівнює true. У даному випадку була обрана операція додавання, тож нове значення компоненти G дорівнює:

$$85_{(10)} = 01010101_{(2)}$$

За обраною схемою відповідності бітів у компоненті G 6-й біт порівнюється з молодшим бітом компоненти B, тож він повинен бути незмінним після додавання/віднімання. Після додавання видно, що 6-й біт залишився незмінним, тому аналіз компоненти G вважається завершеним. Переходимо до компоненти B поточного пікселя:

$$107_{(10)} = 01101011_{(2)}$$

Вбудувати у молодший біт даної компоненти треба значення false. Оскільки останній біт дорівнює значенню true – необхідна операція додавання/віднімання. У псевдовипадковому порядку обрана функція додавання. Нове значення компоненти B:

$$108_{(10)} = 01101100_{(2)}$$

За обраною схемою відповідності бітів у компоненті B 5-й та 7-й біти порівнюється з молодшими бітам компоненти R та G відповідно, тож вони повинні бути незмінними після додавання/віднімання. Після додавання видно, що 5-й та 7-й біти залишились незмінними, тому аналіз даного пікселя вважається завершеним.

За аналогією вбудовується повідомлення у наступні байти контейнера.

При вбудовуванні у 1268 байт (компонента B) контейнеру виникає наступна ситуація: значення байту B дорівнює:

$$191_{(10)} = 10111111_{(2)}$$

Вбудувати у молодший біт необхідно значення false. За псевдовипадковою функцією обирається операція додавання. Нове значення B дорівнює:

$$192_{(10)} = 11000000_{(2)}$$

Після операції додавання видно, що 7-й біти залишився незмінними, а от 5-й біт змінився. Оскільки була обрана операція додавання, то при аналізі поточного байту, обирається зворотна операція – віднімання. Спочатку значення повертається в початкове (191), а потім від нього віднімається 1. По суті, зразу віднімається 2.

Отже, нове значення компоненти B:

$$190_{(10)} = 10111110_{(2)}$$

При повторному аналізі видно, що 5-й та 7-й біти не змінилися і останній біт має значення false. Аналіз байту компоненти B вважається завершеним.

Після вбудовування всієї інформації, заповнений контейнер зберігається на електронний носій та готовий то передачі отримувачу повідомлення.

Отримувач декодує отримане повідомлення наступним чином: спочатку, на основі зображення-ключа будується дерево Хаффмана. Воно повинно бути ідентичним дереву на етапі вбудовування.

На наступному етапі визначається кількість пікселів, що зарезервовано під вбудовування ширини та висоти повідомлення та відбувається зчитування цих даних. У даному випадку у перші 6 та останні 6 пікселів вбудовано значення ширини та висоти відповідно конфіденційного зображення.

Далі, у наступних 8 пікселях (починаючи з 7-го пікселя заповненого контейнера) зчитується значення інтервалу. У даному прикладі значення інтервалу дорівнює 7.

На цьому етапі у отримувача повідомлення є вся важлива інформація для коректного зчитування значень компонент R, G та B конфіденційного зображення. Тому створюється матриця пікселів з висотою та шириною конфіденційних графічних даних (150x150) та заповнюється декодованими значеннями.

Декодування значень компонент починається з 14 пікселя або 42 байту (при нумерації з 0). Довжина найкоротшого коду Хаффмана серед усіх дорівнює 3 біти. Тож з 14 пікселя можна зчитувати початок коду Хаффмана з усіх трьох компонент не перевіряючи його наявність у дереві. Після зчитування з поточного пікселя, довжина коду дорівнює 3 біти (по одному з кожної компоненти). Тепер можна починати пошук по дереву Хаффмана. Коду зі значенням 011 у дереві не існує, тож починається обробка 15 пікселя. Оскільки значення інтервалу дорівнює 7, то пікселі під номерами з 15 по 21 пропускаються та зчитування

починається з пікселя під номером 22. Після зчитування значення біту з компоненти R, воно додається до поточного коду. Зараз код Хаффмана дорівнює 0111. У дереві Хаффмана такого коду не існує, зчитування продовжується. Після обробки компонент G та B, код Хаффмана 011100 не знайдено. Переходимо до пікселя під номером 30 (індекс поточного пікселя + значення інтервалу) та зчитуємо біти з нього. Тільки після зчитування компоненти B, код Хаффмана 011100001 знайдено у дереві. Даному коду відповідає значення байту 153. Дане значення записується до масиву байт конфіденційного зображення.

Після заповнення усього результуючого масиву байт, він зберігається як графічний файл з визначеною раніше шириною та висотою, що і є вбудованим мультимедійним повідомленням. На рис. Ґ.7 видно вбудоване зображення (зліва) та декодоване конфіденційне зображення (справа) не відрізняються.

3.4 Програмне забезпечення процесів оброблення даних для крипто-стеганографічного захисту мультимедійних даних методом на основі процедури псевдовипадкового вбудовування

Метод на основі процедури псевдовипадкового вбудовування реалізований у двох класах: *RandomWithKeyEmbeder* для вбудовування та *RandomWithKeyExtractor* для декодування конфіденційних файлів. Дані класи успадковані від базових класів *BaseCodeMethod* та *BaseDecodeMethod*.

Розглянемо приклад роботи методу на основі процедури псевдовипадкового вбудовування. Нехай дані наступні вхідні дані:

1. Аудіо-файл у WAV форматі cover.wav як контейнер з наступними параметрами:
    - частота дискретизації – 44100 Герц;
    - кількість байт на 1 семпл – 2;
    - кількість каналів – 2;
    - розмір контейнера – 48185620 байт.

2. Зображення secret.png з роздільністю 350x350 пікселів у якості конфіденційних графічних даних (рис. Ґ.8);
3. Документ key.pdf у якості ключа розміром 446464 байт;
4. Логічна функція не задана у явному вигляді (за замовчуванням використовується сума за модулем 2);
5. Довільне ціле невід'ємне число 56370256, що виступає як "зерно" для першого генератора псевдовипадкових чисел.
6. Довільне ціле невід'ємне число, що виступає як "зерно" для другого генератора псевдовипадкових чисел на задано, отже використовується число з п. 5.

Спочатку зчитується усі метадані контейнера та визначається загальна кількість семплів у заповненому WAV-файлі. У даному випадку контейнер має 24092712 семплів, кожен з яких займає 2 байти, у двох каналах. Розмір конфіденційного файлу – 182108 байт.

У даному випадку кількість змінюваних біт дорівнює кількості байт на один семпл, а отже $n = 2$. За нерівністю (2.12) визначається можливість вбудовування даного файлу у контейнер.

$$\frac{8 \cdot s}{n} \leq c_1$$

$$\frac{8 \cdot 182108}{2} \leq 24092712$$

$$728432 \leq 24092712$$

Отже, конфіденційне зображення можна вбудувати у обраний аудіоконтейнер. Оскільки $n = 2$, то масив байтів конфіденційного зображення ділимо на блоки по 2 біти. Таким чином, розмір u масиву блоків U буде дорівнювати за формулою (2.10) 728432 (вище з нерівності).

Наступним кроком заповнюється масив U індексами елементів. Використовуючи генератор псевдовипадкових чисел вихор Мерсенна з

початковим значенням зазначеним у ключі, а саме 56370256, перемішується масив *U* (лістинг 3.8).

Лістинг 3.8. Заповнення та перемішування масиву чисел *U*.

```
u.reserve(secretSize);
for(quint32 i = 0; i < secretSize; ++i)
{
    u.emplace_back(i);
}
std::shuffle(u.begin(), u.end(),
std::mt19937(startValue));
```

Далі починається вбудовування самого конфіденційного зображення за допомогою ітерування по всім семплам контейнера незалежно від того, якому каналу належить поточний семпл. Оскільки зерно для другого генератора псевдовипадкових чисел не задано, то використовується зерно з першого генератора, що виступає як один з ключів методу. На першій ітерації за формулою (2.13) визначається, чи будуть у даний семпл вбудовуватись стегодані або шум.

$$\frac{randTo(c_1)}{c_1} < \frac{u-j}{c_1-i}$$

$$\frac{4695748}{24092712} < \frac{728432-0}{24092712-0}$$

$$0.1949 < 0.0302$$

У даному випадку у контейнер вбудовується шум. На перших 60 ітераціях у контейнер вбудовується шум. На 61-й ітерації значення змінних у формулі (2.13) наступні:

$$\frac{552199}{24092712} < \frac{728432-0}{24092712-60}$$

$$0.0229 < 0.0302$$

При даних значеннях змінних у семпл на позиції 60 буде вбудовано модифіковані 2 біти стегоданих. У даному випадку, кожен семпл описується 2

байтами. Перший байт відповідає за старші біти, другий – за молодші. Оскільки у семплі змінюються лише 2 молодші біти, для оптимізації зчитується лише другий байт, у першому немає необхідності у даному випадку. У другому байті одразу занулюються 2 молодші біти. Значення другого байту дорівнює 0.

Наступним кроком зчитуються 2 біти інформації з масиву блоків *U* на позиції *j*. У даному випадку, слід пам'ятати, що блоки у масиві *U* перемішані у псевдовипадковому порядку. Зчитані 2 біти стегоданих дорівнюють $2_{(10)} = 10_{(2)}$.

Далі зчитуються перші 2 біти з файлу, що виступає як ключ. Значення цих 2-х біт дорівнює $0_{(10)} = 00_{(2)}$. Оскільки ключ, що визначає логічну функцію не заданий, то використовується логічна функція за замовчуванням – сума за модулем 2. Отже, застосовуємо логічну функцію до пари бітів стегоданих та пари бітів файлу-ключа:

$$2_{(10)} \oplus 0_{(10)} = 10_{(2)} \oplus 00_{(2)} = 10_{(2)} = 2_{(10)}$$

Таким чином молодші 2 біти поточного семплу дорівнюватимуть $10_{(2)}$. Модифіковане значення семплу записується до контейнеру. За аналогією ітерування продовжується по всьому контейнеру (лістинг 3.9).

Лістинг 3.9. Запис модифікованих молодших бітів до семплів контейнера.

```
for(quint32 i = 0; i < ContSize; ++i)
{
if(gen.bounded(ContSize) / (qreal)ContSize <
((qreal)secretSize - j) / (ContSize - i))
{
    qint8 val = _conteinerWithData->getDataAt(i) & -4;
    quint8 secret2Bits = _secret-
>get2BitsByIndex(_usedPosition[j]);
    if((secret2Bits & 2) ^ (_key->readBit() << 1))
        val |= 2;
    if((secret2Bits & 1) ^ _key->readBit())
        val |= 1;
```

```
          _conteinerWithData->setDataAt(i, val);
          ++j;
        }
        else
        {
          _conteinerWithData->setDataAt(i,
_conteinerWithData->getDataAt(i) | noiseGen.bounded(4));
        }
}
```

Після обходу усіх семплів, заповнений WAV-контейнер зберігається на носій та готовий до передачі отримувачу. При прослуховування заповненого контейнера підозри про наявність допоміжної інформації у контейнері не виникає. При декодуванні, першим кроком підраховується кількість блоків по 2 біти в залежності від розміру стегоданих в байтах (один з ключів для коректного декодування) за формулою (2.14):

$$u_1 = \frac{8 \cdot s_1}{n} = \frac{8 \cdot 182108}{2} = 728432$$

Наступним кроком створюється масив U_1 розміром $u_1 = 728432$елементів та заповнюється індексами елементів. За допомогою зерна для першого генератора псевдовипадкових чисел (другий з ключів), що дорівнює 56370256, масив U_1перемішується. Далі створюється бітовий масив розміром $8 \cdot c_1 = 8 \cdot 182108 = 1456864$елементів. Даний результуючий масив далі буде перетворений у масив байтів, а той у свою чергу, і буде конфіденційним файлом.

За формулою (2.15) та другого генератора псевдовипадкових чисел визначаються семпли у заповненому контейнері з корисною інформацією. Оскільки зерно для другого ГПВЧ не задано, то використовується зерно з першого. На перших 60 ітераціях, як і на етапі кодування, вбудовано шум, тому перші 60 семплів пропускаються. На 61-у вбудовані перші 2 біти конфіденційного файлу. Молодший байт семплу дорівнює $10_{(2)} = 2_{(10)}$. Перший

біт файлу-ключа дорівнює *false*. Оскільки старший біт дорівнює *true*, а у якості логічної функції використовується сума за модулем 2, то старший біт у даному випадку залишається без змін:

$$1 \oplus 0 = 1$$

Другий біт файлу-ключа також дорівнює *false*. Молодший біт молодшого байту дорівнює *false*.

$$0 \oplus 0 = 0$$

Таким чином, результуючими першими двома бітами будуть $10_{(2)} = 2_{(10)}$. Даний результат записується у результуючий бітовий масив на позицію, що зазначена у масиві U_1 на нульовій позиції, оскільки це нульові 2 біти. У даному випадку позиція блоку дорівнює 346047.

Після ітерування по всім семплам, результуючий бітовий масив буде повністю заповнений парами бітів. Дана бітова послідовність зберігається як файл на накопичувач. Отриманий файл і є конфіденційною інформацією.

3.5 Висновки до розділу 3

У цьому розділі розроблено універсальну архітектуру програмної системи захисту мультимедійних даних, визначальними рисами якої є можливість використання довільних методів крипто-стеганографічного захисту, легкість масштабування та адаптації до використання у довільних програмних середовищах.

При розробленні програмного забезпечення, що реалізує запропоновану архітектуру та розроблені алгоритмічно-програмні методи, використовувалась методологія об’єктно-орієнтованого програмування з використанням основних принципів: абстракція, поліморфізм, наслідування, інкапсуляція.

Реалізовано наступні методи:

- класичний LSB-метод;

- метод на основі схеми відповідності бітів;
- метод на основі дерева Хаффмана;
- метод на основі процедури псевдовипадкового вбудовування.

Усі розробленні методи представленні як окремі класи, що успадковуються один від одного. Деякі методи мають дві імплементації залежно від типу контейнера (аудіо або графічні дані). Імплементації розділено на дві окремі групи: для вбудовування та декодування стегоданих.

Розроблені алгоритмічно-програмні методи забезпечують захист мультимедійних даних від несанкціонованого доступу. В результаті програмної реалізації запропонованих методів можна відзначити наступне:

1. В програмних реалізаціях усіх запропонованих методів приховані стегодані можуть бути відновленні без втрат.
2. Програмна реалізація методу на основі схеми відповідності бітів є найшвидшою серед програмних реалізацій інших розроблених алгоритмічно-програмних методів.
3. Програмна реалізація методу на основі дерева Хаффмана є найстійкішою до стегоатак з точки зору підбору ключа у випадку якщо контейнером виступають графічні дані.
4. Програмна реалізація методу на основі процедури псевдовипадкового вбудовування є найстійкішою до стегоатак з точки зору підбору ключа у випадку якщо контейнером виступають аудіодані.

Результати свідчать про правильність розробленої архітектури програмної системи та методів вирішення поставленої задачі. Крім того, обрана мова програмування, алгоритми та структури даних дозволили створити програмний продукт, що дозволяє забезпечити реалізацію розроблюваних методів стеганографічного захисту мультимедійних даних при мінімізації часових витрат.

РОЗДІЛ 4. АНАЛІЗ РОЗРОБЛЕНИХ АЛГОРИТМІЧНО-ПРОГРАМНИХ МЕТОДІВ КРИПТО-СТЕГАНОГРАФІЧНОГО ЗАХИСТУ МУЛЬТИМЕДІЙНИХ ДАНИХ

4.1 Аналіз алгоритмічно-програмного методу захисту графічних даних на основі схеми відповідності бітів

Для аналізу методу на основі схеми відповідності бітів пропонується порівняти розроблений метод з існуючими аналогами за наступними критеріями [23]:

- візуальна стійкість;
- часова складність методу;
- ймовірність декодування даних.

Критерій швидкодії методу є особливо важливим для систем захисту даних. Серед основних задач вимірювальних експериментів можна виділити наступні:

1. Виявити закономірності між часом роботи методу та розміром контейнерів.
2. Виявити закономірності між часом роботи методу та розміром стегоданих.
3. Виявити закономірності між часом роботи методу та кількістю змінюваних молодших біт.
4. Порівняти час роботи існуючих та розробленого методу на основі схеми відповідності бітів.

Для порівняння були обрані наступні методи:

- метод на основі схеми відповідності бітів;
- метод LSB-стеганографії з фрагментацією стегоданих [55, 58];
- класичний LSB-метод.

Дані методи було обрано оскільки у них є багато спільного:

- LSB-модифікація;
- можна змінювати кількість змінюваних біт від 1 до 4;
- можуть використовувати ключ;
- при вбудовуванні змінюється однакова кількість байт;
- однаковий максимальний розмір вбудованого зображення;
- чутливі до стиснення;
- чутливі до статистичних атак.

Кожен метод було запущено 100 разів з метою зниження похибки. Час роботи методу – час обробки повідомлення, ключів та контейнера без урахування зчитування та запису даних на накопичувач. Для вимірювання часу виконання кожного з методів було забезпечено виконання наступних умов:

- ідентичне апаратне середовище;
- мінімізація впливу сторонніх процесів.

Вимірювання проводились на одному і тому ж комп'ютері під операційною системою Windows 10 (Home edition) з наступними характеристиками:

1. Процесор – Intel Core i7-3610QM.
2. Оперативна пам'ять – 20.0 ГБ.
3. Тип ОС – x64.
4. Відеокарта – NVIDIA GeForce GTX 670M (3 ГБ).
5. Накопичувач – SSD 256 ГБ.

Операційна система Windows мультизадачна, тому всі заміри експериментів за умов вимкнення усіх процесів і програм, що не є обов'язковими для повноцінного функціонування ОС.

Було використано декілька наборів пар "контейнер" - "стегодані" та різні налаштування кількості змінюваних бітів.

При порівнянні часових характеристик був проведений аналіз методів на основі схеми відповідності бітів та методу з фрагментацією стегоданих.

Класичний LSB-метод не було включено до таблиць з часовими замірами роботи методів, оскільки в ньому відсутні додаткові процедури шифрування стегоданих.

Таблиця 4.1 – Час кодування та декодування даних у мс. при зміні одного молодшого біту у контейнер з роздільністю 2040x2040

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
800x500	Вбудовування	64,65	301,7	252,7
	Декодування	119,36	21,8	19,85
225x225	Вбудовування	9,21	139,75	136,55
	Декодування	14,78	23,2	21,15
150x150	Вбудовування	3,98	125,65	126,75
	Декодування	7,11	22,1	20,9
22x40	Вбудовування	<1	122,25	124
	Декодування	1,5	21,8	20,5

Метод на основі схеми відповідності бітів та LSB-метод з фрагментацією стегоданих мають додаткові операції, через що час виконання даних методів і більший ніж стандартного LSB. У табл. 4.1 порівнюючи метод на основі схеми відповідності бітів та метод з фрагментацією стегоданих, можна зробити висновок, що вбудовування стегоданих методом на основі схеми відповідності відбувається швидше у [3,91; 122] разів в залежності від розміру повідомлення. При зменшенні розміру стегоданих, прискорення відносно методу з фрагментацією стегоданих збільшується.

Стосовно декодування, при великих розмірах стегоданих, метод на основі схеми відповідності бітів показує гірші часові показники порівняно з метод фрагментації (приблизно у 6 разів). Але при зменшенні розміру стегоданих, метод на основі схеми відповідності показує кращі часові показники до 20 разів.

У наступному експерименті було зменшено розмір контейнеру та відбувалось вбудовування стегоданих зі зміною одного молодшого біту. Результати зображено у табл. 4.2.

Таблиця 4.2 – Час кодування та декодування даних у мс. при зміні одного молодшого біту у контейнер з роздільністю 1024x1024

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
225x225	Вбудовування	7,88	63,25	56
	Декодування	15,61	18	123,4
150x150	Вбудовування	4,7	49,25	47,3
	Декодування	7,55	15,5	135,85
22x40	Вбудовування	<1	40,05	40,2
	Декодування	<1	15,95	123,35

З табл. 4.2 видно, що метод на основі схеми відповідності бітів показує кращі результати як при вбудовуванні, так і при декодуванні відносно методу з використанням фрагментації.

З табл. 4.3 можна зробити висновок, що розпаралелений метод з фрагментацією стегоданих майже завжди дає більш кращі результати при заміні останніх двох біт ніж реалізація в одному потоці.

Таблиця 4.3 – Час кодування та декодування даних у мс. при зміні двох молодших біт у контейнер з роздільністю 2040x2040

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
900x900	Вбудовування	262,71	313,45	266,95
	Декодування	276,3	22,25	20,7
800x500	Вбудовування	132,44	219,15	197,8
	Декодування	131,73	21,8	19,95
225x225	Вбудовування	15,66	131,05	131,85
	Декодування	16,4	26,25	21
150x150	Вбудовування	7,65	120,75	23,5
	Декодування	7,8	22,25	20,95

Порівнюючи розпаралелений метод з фрагментацією стегоданих з методом на основі схеми відповідності бітів, можна зробити висновок, що при вбудовуванні повідомлення, метод на основі схеми відповідності дає завжди кращі результати.

При декодуванні великого об’єму стегоданих, він показує найгірші показники серед усіх методів.

Що стосується маленького об’єму відносно контейнера, то при декодуванні метод на основі схеми відповідності виграє у обох реалізацій методу з фрагментацією стегоданих.

Результати експерименту зі зменшеним контейнером, що наведено у табл. 4.4, свідчать, що найшвидшим є метод на основі схеми відповідності бітів, далі метод з фрагментацією стегоданих в одному потоці і останнім – його розпаралелена реалізація.

Таблиця 4.4 – Час кодування та декодування даних у мс. при зміні двох молодших біт у контейнер з роздільністю 1024x1024

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
225x225	Вбудовування	16,79	49,7	47,9
	Декодування	16,99	17,05	125,65
150x150	Вбудовування	7,88	43,1	42,7
	Декодування	7,93	16,65	125,3
22x40	Вбудовування	<1	40,45	39,3
	Декодування	<1	16,05	126,25

З табл. 4.5 можна зробити висновок, що метод з фрагментацією стегоданих дає найкращі результати при декодуванні великих зображень. Однак на етапі вбудовування метод на основі схеми відповідності бітів найшвидший.

Таблиця 4.5 – Час кодування та декодування даних у мс. при зміні трьох молодших біт у контейнер з роздільністю 2040x2040

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
900x900	Вбудовування	252,15	283,5	253,85
	Декодування	259,63	23,05	20,5
800x500	Вбудовування	125,21	202,55	188,2
	Декодування	131,72	21,9	20,6
225x225	Вбудовування	16,45	128,05	130,7
	Декодування	16,96	24,3	21,55
150x150	Вбудовування	7,44	124,05	124,75
	Декодування	7,98	23,05	21,7

При зменшенні роздільності контейнера з 2040x2040 на 1024x1024 з табл. 4.6 видно що метод на основі схеми відповідності бітів завжди швидший за метод з фрагментацією стегоданих при вбудовуванні та у 2 з 3 випадків швидший при декодуванні (при маленькому об’ємі стегоданих).

Останнім етапом у замірі часових показників залишилось тестування методів при заміні останніх чотирьох бітів.

З табл. 4.7 можна зробити висновок, що при вбудовуванні великих повідомлень обидві реалізації методу з фрагментацією стегоданих виграють метод на основі схеми відповідності бітів, однак при вбудовуванні середніх та малих конфіденційних зображень відносно розміру контейнера, метод на основі схеми відповідності показує кращі результати. Щодо декодування даних ця тенденція зберігається. При великих розмірах вбудованих графічних даних, розпаралелена реалізація методу з фрагментацією стегоданих є найшвидшою на стадії декодування.

Таблиця 4.6 – Час кодування та декодування даних у мс. при зміні трьох молодших біт у контейнер з роздільністю 1024x1024

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
225x225	Вбудовування	16,3	49,35	50
	Декодування	16,85	16,5	136,5
150x150	Вбудовування	7,4	43,6	44,95
	Декодування	7,77	16,75	131,85
22x40	Вбудовування	1,64	40,1	41,3
	Декодування	1,71	18,1	129,3

При зменшенні розміру контейнера тенденції описані вище значно не змінилися. Таким чином, можна зробити наступні висновки:

1. При зміні 1, 2 або 3 молодших бітів, метод на основі схеми відповідності бітів вбудовує конфіденційні графічні дані швидше, ніж метод з фрагментацією стегоданих та розпаралелений метод з фрагментацією стегоданих.
2. При зміні 4 молодших бітів, при великих розмірах стегоданих відносно розміру контейнера, метод з фрагментацією стегоданих та розпаралелена його реалізація показують кращі часові показники порівняно з методом на основі схеми відповідності.

Таблиця 4.7 – Час кодування та декодування даних у мс. при зміні чотирьох молодших біт у контейнер з роздільністю 2040x2040

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
1024x1024	Вбудовування	284,6	252,3	234,15
	Декодування	297,55	23	20,35
900x900	Вбудовування	234,5	225,25	220,45
	Декодування	250,71	22	20,9
800x500	Вбудовування	119,4	174	164,25
	Декодування	129,88	21,95	20,1
225x225	Вбудовування	14,7	124,35	125,7
	Декодування	15	23,75	21,45
150x150	Вбудовування	7,2	120,4	119,8
	Декодування	7,35	22,7	21,4

3. При зміні 4 молодших бітів, при середніх та малих розмірах стегоданих відносно розміру контейнера, обидві реалізації методу з фрагментацією стегоданих показують гірші результати порівняно з методом на основі схеми відповідності.
4. При малих розмірах стегоданих відносно розміру контейнера на етапі декодування швидкодія методів за спаданням буде наступною: метод на

основі схеми відповідності бітів; метод з фрагментацією стегоданих в одному потоці; розпаралелений метод з фрагментацією стегоданих.

Таблиця 4.8 – Час кодування та декодування даних у мс. при зміні чотирьох молодших біт у контейнер з роздільністю 1024x1024

Розмір стегоданих	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)
700x700	Вбудовування	137,01	104,5	99
	Декодування	137,86	15,55	121,4
225x225	Вбудовування	14,2	44,15	44,1
	Декодування	15,15	16,75	121,9
150x150	Вбудовування	6,85	41,7	49,75
	Декодування	7,63	16,9	121,2
22x40	Вбудовування	1,56	37,8	37,7
	Декодування	1,77	16,25	117,2

5. При великих розмірах стегоданих відносно розміру контейнера на етапі декодування швидкодія методів за спаданням буде наступною: розпаралелений метод з фрагментацією стегоданих; метод з фрагментацією стегоданих в одному потоці; на основі схеми відповідності бітів.
6. Захист стегоданих завжди сповільнює час роботи методів, як при запису, так і при зчитуванні.
7. Не існує універсального методу захисту, який би у всіх випадках працював швидше за інших.

Звідси можна зробити висновок, що у випадку, коли передаються великі об’єми даних та важливий час декодування, а часом вбудовування можна знехтувати, то розпаралелений метод з фрагментацією стегоданих є найкращим варіантом. Якщо необхідно передати дані максимально швидко і користувач

допускає, що за ним не слідкують та конфіденційна інформація може бути перехоплена з мінімальною ймовірністю – класичний LSB-метод. У всіх інших випадках метод на основі схеми відповідності бітів показує кращі результати.

Аналіз часу вбудовування стегоданих у табл. 4.1-4.8 дав змогу зробити висновок, що метод на основі схеми відповідності бітів працює швидше у середньому від 5,13 до 43,53 рази. Для розрахунку було використано середнє арифметичне часу вбудовування по кожній з таблиць. Виходячи з цього, можна зробити висновок, що метод на основі схеми відповідності бітів у абсолютній більшості випадків буде швидшим на етапі вбудовування даних.

З точки зору ефективності захисту, то класичний LSB-метод є найменш ефективний. На це є декілька причин:

- метод жодним чином не модифікує вхідне зображення при вбудовуванні;
- метод вбудовує дані у контейнер послідовно;
- даний метод відомий усім стегоаналітикам.

Метод з фрагментацією стегоданих та розпаралелений метод з фрагментацією стегоданих з точки зору ефективності захисту вбудованого зображення слід розглядати як один метод, оскільки у даному випадку мова йде про дві різні реалізацію одного й того ж методу.

Порівнюючи класичний LSB-метод та метод з фрагментацією стегоданих, останній не має таких недоліків як послідовне вбудовування у контейнер та популярності серед стегоаналітиків, тому його ефективність захисту буде набагато вище.

При порівнянні методу на основі схеми відповідності бітів з класичним LSB-методом, перший позбавлений таких недоліків як відсутність модифікування вхідного повідомлення та популярності серед стегоаналітиків.

Отже, можна зробити висновок, що і метод на основі схеми відповідності бітів, і метод з фрагментацією стегоданих є набагато ефективнішими за стандартний LSB-метод.

Порівняємо метод на основі схеми відповідності бітів та метод з фрагментацією стегоданих з точки зору ймовірності підбору ключів для декодування стегоданих стегоаналітиком при наступних умовах:

- стегоаналітик знає яким методом вбудовані стегодані;
- стегоаналітик знає розмір конфіденційних графічних даних;
- стегоаналітик не знає ключів для декодування.

Нехай подія *F* – підбір ключів з першої спроби.

Оцінимо ймовірність підбору ключів для класичного LSB-методу.

Оскільки у класичному LSB-методі ключем є кількість змінюваних бітів (1, 2, 3 або 4), тоді ймовірність підбору ключів визначається за формулою:

$$P(F) = \frac{1}{4},$$

Оцінимо ймовірність підбору ключів для методу на основі схеми відповідності бітів. Для коректного декодування стегоданих методом на основі схеми відповідності бітів, треба знати наступні ключі:

- кількість змінюваних біт (4 варіанти);
- логічна функція (4 варіанти);
- схема відповідності бітів.

Ймовірність підбору схеми відповідності при зміні одного молодшого біту для одного пікселя дорівнює $\left(\frac{1}{21}\right)^3$. Нехай подія H_1– підбір ключів з першої спроби у випадку, що змінено 1 молодший біт. Оскільки існує 4 логічні функції, то ймовірність підбору ключів дорівнює:

$$P_{H_1}(F) = \frac{1}{4 \cdot 21^3}$$

Ймовірність підбору схеми відповідності при зміні двох молодших біт для одного пікселя дорівнює $\left(\frac{1}{18}\right)^6$. Нехай подія H_2– підбір ключів з першої спроби у випадку, що змінено 2 молодших біти.

$$P_{H_2}(F) = \frac{1}{4 \cdot 18^6}$$

Ймовірність підбору схеми відповідності при зміні трьох молодших біт для одного пікселя дорівнює $\left(\frac{1}{15}\right)^9$. Нехай подія H_3– підбір ключів з першої спроби у випадку, що змінено 3 молодших біти.

$$P_{H_3}(F) = \frac{1}{4 \cdot 15^9}$$

Ймовірність підбору схеми відповідності при зміні чотирьох молодших біт для одного пікселя дорівнює $\left(\frac{1}{12}\right)^{12}$. Нехай подія H_4– підбір ключів з першої спроби у випадку, що змінено 4 молодших біти.

$$P_{H_4}(F) = \frac{1}{4 \cdot 12^{12}}$$

Отже, загальну ймовірність підбору ключів можна визначити за формулою:

$$P_{H_1H_2H_3H_4}(F) = \frac{1}{\sum_{i=1}^{b=4}(4 \cdot (24 - 3 \cdot i)^{3 \cdot i})} = \frac{1}{4 \cdot 21^3 + 4 \cdot 18^6 + 4 \cdot 15^9 + 4 \cdot 12^{12}} =$$

$$\frac{1}{37044 + 136048896 + 153773437500 + 35664401793024} \approx 2{,}79 \cdot 10^{-14}$$

Оцінимо ймовірність підбору ключів для методу з фрагментацією стегоданих. Для коректного декодування стегоданих методом з фрагментацією стегоданих, треба знати наступні ключі:

- кількість змінюваних біт (4 варіанти);
- кількість блоків;
- довжини блоків.

Оскільки розмір стегоданих стегоаналітику відомі, то максимальну кількість блоків для вбудовування можна визначити за формулою:

$$m = \frac{w \cdot h \cdot 3 \cdot 8}{b} = \frac{24 \cdot w \cdot h}{b}, \quad (4.1)$$

де m – максимальна кількість блоків;

w – ширина конфіденційного зображення у пікселях;

h – висота конфіденційного зображення у пікселях;

b – кількість змінюваних біт.

Кількість блоків у які можна вбудовувати, по суті, є масивом байт контейнера, і тому визначається за формулою:

$$n = 3 \cdot w_1 \cdot h_1, \tag{4.2}$$

де n – кількість блоків, я які вбудовуються стегодані;

w_1 – ширина контейнера у пікселях;

h_1 – висота контейнера у пікселях.

Ймовірність підбору ключів з першої спроби можна визначити за допомоги формули розміщень з комбінаторики:

$$P(F) = \frac{1}{\sum_{i=1}^{b=4} A_{3 \cdot w_1 \cdot h_1}^{\frac{24 \cdot w \cdot h}{i}}} = \frac{1}{\sum_{i=1}^{b=4} \left(\frac{(3 \cdot w_1 \cdot h_1)!}{\left(\frac{3 \cdot (w_1 \cdot h_1 \cdot i - 8 \cdot w \cdot h)}{i} \right)!} \right)}$$

У загальному випадку, у методу з фрагментацією стегоданих ймовірність підбору ключів стегоаналітиком набагато менша, ніж у методу на основі схеми відповідності бітів, однак останній має суттєві переваги:

- може працювати без ключа;
- легко модифікується з багатьма іншими методами.

Умовно, переважна більшість відомих методів на основі LSB-вбудовування складається з трьох етапів:

1. Обробка вхідного повідомлення, можливо його кодування.
2. Визначення певного порядку позицій байтів контейнера у які буде вбудоване повідомлення.
3. Метод вбудовування у байт.

Більшість модифікацій LSB-методу проводять певні операції лиш на одному з цих етапів, інші залишаючи без змін. Найпопулярніші LSB-модифікації конвертують вхідний масив бітів у інший масив бітів (1 етап) та/або вбудовують

повідомлення на псевдовипадкові позиції (2 етап), як, наприклад, метод з фрагментацією стегоданих. Однак вбудовується дана інформація у більшості випадків простою заміною останніх *n* бітів на біти (можливо оброблені) повідомлення.

Метод на основі схеми відповідності бітів виконується на третьому етапі, тому перші два залишаються за замовчуванням. Тож він легко об'єднується з методом з фрагментацією стегоданих та багатьма іншими LSB-модифікаціями, чим підвищує їх стеганографічну стійкість.

4.2 Аналіз алгоритмічно-програмного методу захисту графічних даних на основі дерева Хаффмана

Метод на основі дерева Хаффмана будемо аналізувати за наступними показниками [18, 44, 53]:

- візуальна стійкість;
- часова складність методу;
- ймовірність декодування конфіденційного зображення;
- візуальна стійкість LSB (найменш значущий біт);
- стійкість до RS-методу.

Оскільки у даному методі змінюється лише один молодший біт, то при вбудовуванні конфіденційних графічних даних, візуально неозброєним оком відрізнити порожній контейнер від заповненого неможливо.

Щодо замірів часу, то експерименти проводились у тих же апаратних та програмних умовах, як при тестуванні методу на основі схеми відповідності бітів у п. 4.1.

Для порівняння були обрані наступні методи:

- класичний LSB-метод;
- метод на основі схеми відповідності бітів;

- метод з фрагментацією стегоданих;
- розпаралелена реалізація методу з фрагментацією стегоданих;
- метод на основі дерева Хаффмана.

Як і в п. 4.1, класичний LSB-метод не було включено до таблиць з часовими замірами роботи методів, оскільки він завжди буде швидшим за модифікації, так як у ньому відсутні додаткові операції шифрування конфіденційних даних безпосередньо перед вбудовуванням та операції для підвищення стеганографічної стійкості від статистичних атак.

У даному випадку також було використано декілька наборів пар “контейнер” – “стегодані”.

Часові показники вбудовування та декодування конфіденційних графічних даних у контейнер з роздільністю 2040x2040 усіх методів, що порівнюються, представленні у табл. 4.9. У дужках у методі на основі дерева Хаффмана показано час побудови дерева Хаффмана, що не включено до основного часу вбудовування.

Як видно з табл. 4.9, метод на основі дерева Хаффмана при вбудовуванні майже завжди показує кращі результати без побудови дерева Хаффмана, аніж обидві реалізації методу з фрагментацією стегоданих, але у всіх випадках програє методу на основі схеми відповідності бітів. Це досить очікуваний результат, оскільки метод на основі дерева Хаффмана виконує ті ж операції, що і метод на основі схеми відповідності бітів плюс додаткове зчитування, пошук та запис кодів Хаффмана.

При відсутності дерева Хаффмана, якщо відправник та отримувач вперше обмінюються стегоданими, або змінили ключі, то на етапі вбудовування метод на основі дерева Хаффмана показує найгірші показники, оскільки потребує перебудови дерева.

Що стосується етапу декодування, то з табл. 4.9 видно, що метод на основі дерева Хаффмана показує найгірші показники.

Таблиця 4.9 – Час кодування та декодування даних у мс. при зміні одного молодшого біту у контейнер з роздільністю 2040x2040

Розмір стегода-них	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)	Метод на основі дерева Хаффмана (мс.)
800x500	Вбудову-вання	64,65	301,7	252,7	264,20 (567,5)
	Декоду-вання	119,36	21,8	19,85	8526,3 (572,02)
225x225	Вбудову-вання	9,21	139,75	136,55	43,11 (406,6)
	Декоду-вання	14,78	23,2	21,15	1164,74 (408,72)
150x150	Вбудову-вання	3,98	125,65	126,75	24,49 (367,36)
	Декоду-вання	7,11	22,1	20,9	512,82 (370,71
22x40	Вбудову-вання	<1	122,25	124	7,1 (350,82)
	Декоду-вання	1,5	21,8	20,5	25,57 (355,39)

Це пов’язано з тим, що при кожному декодуванні одного біту з контейнера він додається до поточного коду Хаффмана. Після цього, код Хаффмана відшукується у дереві Хаффмана. Якщо код знайдено – починається пошук нового коду. Якщо ні – зчитується наступний біт та додається до поточного коду Хаффмана. Для оптимізації, після побудови дерева Хаффмана відбувається пошук найкоротшого коду. Часто він складається з 3 бітів. Після додавання нового біту до поточного коду, перевіряється розмір коду. Якщо він менший за найменший розмір коду Хаффмана у дереві, пошук поточного коду у дереві не відбувається. Тож, у загальному випадку, якщо, наприклад, найкоротший код Хаффмана складається з 3 бітів, а поточний код з 7, то пошук даного коду буде відбуватися 4 рази. Пошук у даному випадку досить обчислювальнозатратна операція. Тож часові показники на етапі декодування методу на основі дерева Хаффмана гірші ніж показники усіх інших методів зазначених у таблиці.

У табл. 4.10 представлено часові показники вбудовування та декодування конфіденційних графічних даних у менший контейнер з роздільністю 1024x1024 методом на основі схеми відповідності бітів, методом з фрагментацією стегоданих (обидві реалізації) та методом на основі дерева Хаффмана. У дужках у методі на основі дерева Хаффмана показано час побудови дерева.

З табл. 4.10 видно, що на етапі вбудовування обидві реалізації методу з фрагментацією стегоданих є найповільнішими серед усіх представлених. Метод на основі схеми відповідності бітів, як і очікувалось, є швидший за метод на основі дерева Хаффмана. Однак у випадку коли відправник та отримувач змінили зображення-ключ або вперше обмінюються повідомленнями, метод на основі дерева Хаффмана є найповільнішим.

Таблиця 4.10 – Час кодування та декодування даних у мс. при зміні одного молодшого біту у контейнер з роздільністю 1024x1024

Розмір стегода-них	Тип операції	Метод на основі схеми відповідності бітів (мс.)	Метод з фрагментацією стегоданих (мс.)	Розпаралелений метод з фрагментацією стегоданих (мс.)	Метод на основі дерева Хаффмана (мс.)
225x225	Вбудо-вування	7,88	63,25	56	37,21 (377,98)
	Декоду-вання	15,61	18	123,4	1155,88 (376,45)
150x150	Вбудо-вування	4,7	49,25	47,3	17,61 (362,52)
	Декоду-вання	7,55	15,5	135,85	482,03 (362,61)
22x40	Вбудо-вування	<1	40,05	40,2	3,21 (353,01)
	Декоду-вання	<1	15,95	123,35	20,71 (352,66)

Що стосується декодування конфіденційного зображення, то метод на основі дерева Хаффмана є найповільнішим за винятком передачі досить малого зображення. Найшвидшим є метод на основі схеми відповідності бітів.

У випадку стегоатак на заповнений контейнер методом на основі дерева Хаффмана загроза підбору ключа шляхом повного перебору є виключеною, оскільки мінімальна довжина ключа у даному методі – 256 байт.

Оцінимо ймовірність декодування для методу на основі дерева Хаффмана. Для коректного декодування стегоданих на основі дерева Хаффмана, треба знати наступні ключі:

- логічна функція (4 варіанти);
- схема відповідності бітів;
- зображення-ключ.

По суті, коди Хаффмана при побудові дерева для 256 елементів завжди будуть однакові. У даному випадку, різниця полягає лиш у тому, який код Хаффмана буде відповідати якому значенню байту. Це залежить від кількості повторень значення байту у зображенні-ключі. Отже, кількість варіантів дорівнює кількості перестановок для 256 елементів:

$$P_n = n! = 256!$$

Нехай подія F – підбір ключів з першої спроби. Тоді ймовірність підбору ключів для декодування повідомлення дорівнює:

$$P(F) = \frac{1}{4 \cdot 21^3 \cdot 256!} \approx 3{,}147 \cdot 10^{-512}$$

Порівняємо ймовірність підбору ключів для класичного LSB-методу, методу на основі схеми відповідності бітів, методу з фрагментацією стегоданих та методу на основі дерева Хаффмана.

З табл. 4.11 можна зробити висновок, що ймовірність підбору ключів до методу на основі дерева Хаффмана набагато менша, ніж до класичного LSB-методу або методу на основі схеми відповідності бітів, але у переважній більшості випадків більшій, ніж до методу з фрагментацією стегоданих. Чим більший контейнер, тим більша різниця ймовірностей, оскільки у методу на основі дерева Хаффмана вона константна.

Таблиця 4.11 – Ймовірності підбору ключів до методів з першої спроби

Назва методу	Ймовірність підбору ключів
LSB-метод	$P(F) = \frac{1}{4}$
Метод на основі схеми відповідності бітів	$P(F) \approx 2{,}79 \cdot 10^{-14}$
Метод з фрагментацією стегоданих	$P(F) = \frac{1}{\sum_{i=1}^{b=4} A_{3 \cdot w_1 \cdot h_1}^{\frac{24 \cdot w \cdot h}{i}}} = \frac{1}{\sum_{i=1}^{b=4} \left(\frac{(3 \cdot w_1 \cdot h_1)!}{\left(\frac{3 \cdot (w_1 \cdot h_1 \cdot i - 8 \cdot w \cdot h)}{i} \right)!} \right)}$
Метод на основі дерева Хаффмана	$P(F) \approx 3{,}147 \cdot 10^{-512}$

У табл. 4.11 наведено ймовірність підбору ключів до методів, але треба прийняти до уваги, що ймовірність декодування даних вбудованих методом на основі дерева Хаффмана буде ще нижчою, оскільки вбудовування відбувається не послідовно, а через певні інтервали.

Проаналізувавши реалізацію кожного з методів, можна зробити висновок, що у методі з фрагментацією стегоданих адреси та розмір фрагментів генеруються за допомогою одного ГПВЧ. Таким чином, підібравши зерно для нього, можна отримати всю послідовність адрес та розмірів фрагментів. У більшості випадків, для зерна існує 2^{32}варіантів. Таким чином, після аналізу програмного коду, ймовірність декодування стегоданих методом з фрагментацією стегоданих зводиться до наступної ймовірності:

$$P(F) = \frac{1}{2^{32}}$$

Після аналізу програмної реалізації методу на основі дерева Хаффмана, ймовірність підбору ключів залишиться незмінною. Отже, можна зробити висновок, що з точки зору ймовірності підбору ключів метод на основі дерева Хаффмана є найстійкішим.

Одним з найпопулярніших методів стегоатак є оцінка візуальної стійкості за допомогою найменш значущого біту [78]. Суть методу полягає у наступному: зображення, що виступає у ролі заповненого стегоданими контейнера, розділяється на 3 канали R, G та B. У кожному каналі визначається найменш значущий біт. Далі створюється 3 нових зображення, де кожен піксель описується лише одним наймолодшим бітом кожної компоненти R, G та B. Таким чином отримається 3 бінарних зображення.

Наступним етапом створюється кольорове зображення, де кожен піксель описується трьома байтами R, G та B, кожен з яких може приймати значення або 0, або 255. Значення байтів визначається виходячи з найменш значущого біту компонент заповненого контейнера. Наприклад, перший піксель заповненого контейнера має наступні значення:

$$R = 210_{(10)} = 11010010_{(2)}$$

$$G = 55_{(10)} = 00110111_{(2)}$$

$$B = 147_{(10)} = 10010011_{(2)}$$

Таким чином, обираючи найменш значущий біт кожної компоненти, перший піксель нового кольорового зображення матиме наступні значення компонент:

$$R = 0_{(10)} = 00000000_{(2)}$$

$$G = 255_{(10)} = 11111111_{(2)}$$

$$B = 255_{(10)} = 11111111_{(2)}$$

Щоб продемонструвати роботу методу наглядніше, пропонується обрати за контейнер наступне зображення розміром 2040x2040: рис. Ґ.9.

Наступним кроком конфіденційні графічні дані розміром 225x255 вбудовуються у контейнер, зображений на рис. Ґ.9, класичним LSB-методом замінюючи 1 найменш значущий біт. Далі із порожнього та заповненого контейнера, описаним вище способом, створюється по 4 зображення 3 з яких бінарні (для кожної компоненти R, G та B) і 1 кольорове (значення кожної компоненти або 0, або 255). Отримані бінарні графічні дані для компонент R зображено на рис. Ґ.10.

З рис. Ґ.10 видно, що у порожньому контейнері можна розгледіти силует птаха. Оскільки класичним LSB-методом вбудовування повідомлення відбувається послідовно у кожен піксель контейнера (зліва-направо), то у заповненому контейнері можна побачити на початку зашумлену прямокутну ділянку. У цю ділянку і було вбудовано конфіденційні графічні дані.

З рис. Ґ.10-Ґ.12 видно, що область зображення у яку вбудовано повідомлення можна побачити аналізуючи будь-яку з трьох компонент. Чим більший розмір конфіденційних графічних даних, тим більша область вбудовування. Об’єднавши 3 бінарних зображення вище, отримується кольорове зображення на рис. Ґ.13.

На даному прикладі можна переконатися, що класичний LSB-метод можна виявити за допомогою візуальної атаки найменш значущих бітів.

Далі у обраний контейнер вбудовуються ті ж самі конфіденційні графічні дані розміром 225x225 методом на основі дерева Хаффмана та проводиться аналіз заповненого контейнера. Результати візуальної атаки зображено на рис. Ґ.14.

З рис. Ґ.14 можна зробити висновок, що після стегоатаки LSB різницю між порожнім та заповненим контейнером методом на основі дерева Хаффмана майже не видно.

При вбудовування досить великого повідомлення відносно розміру контейнера розробленим методом, після стегоатаки LSB на результуючих зображеннях (3 бінарних і 1 кольорове) може з’явитися шум, тому

рекомендується обирати якомога більші контейнери. При виконанні цієї рекомендації можна зробити висновок, що метод на основі дерева Хаффмана є стійким до візуальних стегоатак LSB.

Одним з найпопулярнішим статистичним методом стегоатак на графічні дані є RS-метод (Regular-Singular) [80, 82, 120]. У даному методі кожен канал R, G або B аналізується окремо. Суть методу полягає у наступному.

Зображення, що виступає у ролі заповненого контейнера, розбивається на групи по *n* пікселів.

$$G(x_1, x_2, \ldots, x_n), \tag{4.3}$$

де n – деяке парне натуральне число, наприклад, по 2 пікселя, що знаходяться поруч по горизонталі [11].

Для даної групи пікселів визначається функція регулярності (дискримінант-функція або функція гладкості), що зіставляє кожній групі пікселів дійсне число. Ціль даної функції полягає у тому, щоб зафіксувати гладкість або регулярність групи пікселів *G*. Найчастіше обирають суму різниць значень сусідніх пікселів у групі. Під значенням пікселя у даному випадку мається на увазі ціле число у проміжку [0; 255], оскільки кожен канал аналізується окремо.

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n-1} |x_{i+1} - x_i| \tag{4.4}$$

Наступним кроком визначається клас функції *F* фліпінга від одного пікселя, що повинна мати наступну властивість:

$$F\big(F(x)\big) = x \tag{4.5}$$

При даному аналізі використовується 3 функції фліпінга:

- F_1– інверсія молодшого біту у байті кольору в контейнері;
- F_0– байт залишається без змін;

- F_{-1}– інверсія молодшого біту у байті кольору в контейнері з переносом у старший біт (наприклад $255 \leftrightarrow 0{,}1 \leftrightarrow 2$і т.д.).

Приклад функції F_1 наведено у лістингу 4.1.

Лістинг 4.1. Приклад функції F_1.

```
uint8_t flip(uint8_t value)
{
   if(value & 1)
   {
       return value - 1;
   }
   return value + 1;
}
```

Приклад функції F_{-1} наведено у лістингу 4.2.

Лістинг 4.2. Приклад функції F_{-1}.

```
uint8_t invertFilp(uint8_t value)
{
   if(value & 1)
   {
       return value + 1;
   }
   return value - 1;
}
```

Приклад функції F_0 наведено у лістингу 4.3.

Лістинг 4.3. Приклад функції F_0.

```
uint8_t noFlip(uint8_t value)
{
   return value;
}
```

До кожної групи пікселів застосовується одна з функцій фліпінга. Далі на основі значення функції гладкості до і після фліпінга, визначається тип групи пікселів. Їх усього три:

- звичайний (regular);
- одиничний (singular);
- непридатний (unusable).

Так як останній тип груп “непридатний” не використовується у наступних операціях, то метод був названий по першим буквам перших двох типів груп.

$$Regulargroups: G\epsilon R \Leftrightarrow f\big(F(G)\big) > f(G)$$

$$Singulargroups: G\epsilon S \Leftrightarrow f\big(F(G)\big) < f(G)$$

$$Unusablegroups: G\epsilon U \Leftrightarrow f\big(F(G)\big) = f(G)$$

У рамках однієї групи можна застосовувати різні функції фліпінга для різних значень пікселів. Для цього створюється маска *M* (вектор розмірністю *n*), кожен елемент якого приймає значення -1, 0 або 1 (індекс функції фліпінга). Маска вказує яка функція фліпінга відповідає певному значенню пікселя.

$$F_M(G) = \Big(F_{M(1)}(x_1), F_{M(2)}(x_2), \ldots, F_{M(n)}(x_n)\Big) \tag{4.6}$$

Шляхом експериментів, було виявлено, що для даного методу рекомендується використовувати симетричні маски, що не використовують функцію F_{-1}, наприклад [0, 1, 0, 1], [1, 0, 1, 0], [1, 0, 0, 1] і т.д.

Нехай кількість звичайних типів груп у процентному відношенні відносно всіх типів груп для маски M дорівнює R_M, S_M - кількість одиничних груп, U_M– кількість непридатних груп.

$$R_M + S_M + U_M = 1 \tag{4.7}$$

У даному випадку U_M може дорівнювати 0. Звідси випливає:

$$R_M + S_M \leq 1 \tag{4.8}$$

$$R_{-M} + S_{-M} \leq 1, \tag{4.9}$$

де R_{-M}– кількість звичайних типів груп для від'ємної маски (усі компоненти маски помножені на -1);

S_{-M}– кількість одиничних типів груп для від'ємної маски.

Метод ґрунтується на статистичному припущенні, що для порожнього контейнера характерно наступне:

$$R_M \simeq R_{-M}, S_M \simeq S_{-M}$$

Припущення базується на тому, що застосування F_1 дасть той же розподіл, що і F_{-1} на зображенні, значення пікселів якого зсунуто на одиницю. Для звичайного зображення співвідношення між групами не повинно істотно змінюватися. Значна розбіжність між значеннями свідчить про застосування LSB-стеганографії для молодших біт зображення.

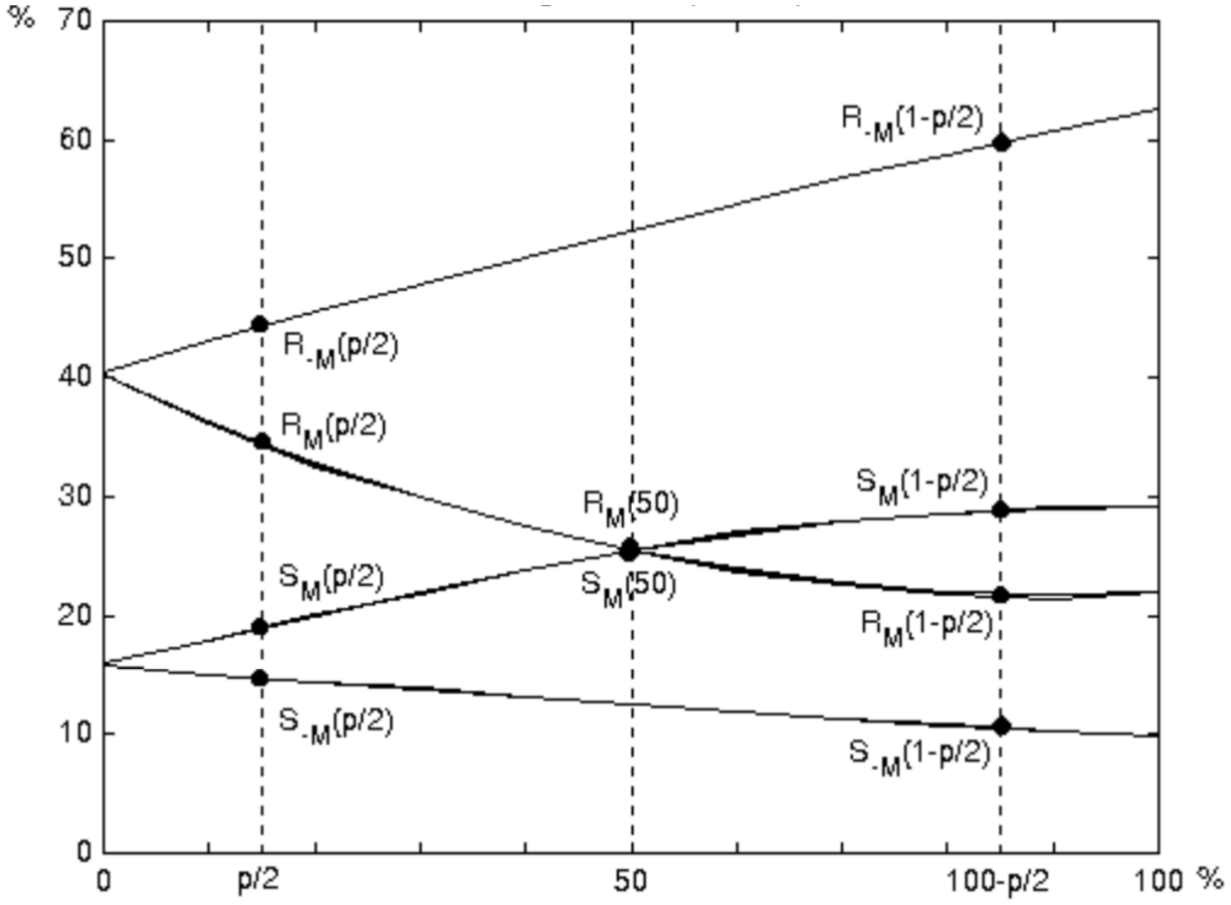

Рис. 4.1 RS-діаграма типового зображення [80]

На рис. 4.1 вісь абсцис представляє собою відсоток пікселів с інвертованим LSB, а вісь ординат – відносне число звичайних та одиничних типів груп з масками *M* та *-M*, *M=[0, 1, 1, 0]*.

Сутність RS-методу полягає у оцінці чотирьох кривих на RS-діаграмі та обчисленні їх точки перетину з використанням екстраполяції. Нехай у контейнер вбудовано повідомлення невідомого розміру p (у відсотках від пікселів). Початкові заміри числа звичайних груп R та одиничних груп S відповідають точкам $R_M(p/2)$, $S_M(p/2)$, $R_{-M}(p/2)$ та $S_{-M}(p/2)$. Точки обираються саме від половини довжини повідомлення, оскільки у середньому тільки одна половина пікселів буде змінена після вбудовування повідомлення. У випадку якщо інвертуються молодші біти усіх пікселів в контейнері та обчислюється R та S, отримується 4 точки $R_M(1-p/2)$, $S_M(1-p/2)$, $R_{-M}1-(p/2)$ та $S_{-M}(1-p/2)$. В залежності від рандомізації вбудовування, визначаються середні точки $R_M(1/2)$, $S_M(1/2)$. Процес треба повторювати багатократно та обчислювати точки $R_M(1/2)$ і $S_M(1/2)$ з статистичних вибірок.

Через точки $R_{-M}(p/2)$, $R_{-M}(1-p/2)$ та $S_{-M}(p/2)$, $S_{-M}(1-p/2)$ проведемо 2 прямі. Точки $R_M(p/2)$, $R_M(1/2)$, $R_M(1-p/2)$ та $S_M(p/2)$, $S_M(1/2)$, $S_M(1-p/2)$визначають 2 параболи. Кожна парабола та відповідна лінія перетинаються зліва. Середнє арифметичне *x*-координат обох перетинів дозволяє оцінити невідому довжину повідомлення.

Зробимо 2 припущення:

1. Точка перетину кривих R_Mта R_{-M} має ту саму координату *x*, що і точка перетину кривих S_Mта S_{-M}.
2. Криві R_M та S_M перетинаються при *m* = 50% або $R_M(1/2) = S_M(1/2)$.

Дані припущення дозволяють отримати просту формулу для довжини повідомлення *p*. Після масштабування осі *x* таким чином, що *p* / 2 стає 0, а (1 - *p* / 2) дорівнює 1, *x*-координата точки перетину є коренем з наступного квадратного рівняння:

$$2(d_1 + d_0)x^2 + (d_{-0} - d_{-1} - d_1 - 3d_0)x + d_0 - d_{-0} = 0, \quad (4.10)$$

де $d_0 = R_M(p/2) - S_M(p/2)$,

$d_{-0} = R_{-M}(p/2) - S_{-M}(p/2)$,

$d_1 = R_M(1 - p/2) - S_M(1 - p/2)$,

$d_{-1} = R_{-M}(1 - p/2) - S_{-M}(1 - p/2)$.

Тоді довжину повідомлення можна обчислити за формулою:

$$p = \frac{x}{x - 0.5}, \quad (4.11)$$

Розглянемо результати роботи методу у табл. 4.12.

Таблиця 4.12 – Результати роботи RS-методу

Розмір контейнера	Розмір конфіденційного зображення	Фактична заповненість контейнера (%)	RS для порожнього контейнера (%)	RS для контейнера заповненим LSB (%)	RS для контейнера заповненим методом на основі Хаффмана (%)
1024x1024	150x150	17,17	3,12	36,66	4,06
1024x1024	340x340	88,16	3,12	88,79	5,7
1300x1300	225x225	23,96	0,01	61,66	1,38
1300x1300	440x440	91,68	0,01	29,78	3,44
2000x1300	350x350	37,69	0,18	50,84	1,64
2000x1300	540x540	89,76	0,18	64,33	3,97
2400x1800	540x540	54	7,46	61,34	8,71
2400x1800	700x700	90,72	7,46	59,77	8,4

RS-метод має одну велику перевагу – він працює, навіть якщо вбудовування у контейнер було не послідовне як у класичному LSB-методі, а псевдовипадкове, як у методу з фрагментацією стегоданих та методі на основі

дерева Хаффмана. Але оскільки у методі на основі Хаффмана була не проста заміна останнього біту на секретний, то з табл. 4.12 видно, що RS-метод не працює з контейнерами заповненими методом на основі дерева Хаффмана. А от наявність прихованих даних у контейнері заповненим класичним LSB-методом при зміні всього одного біту метод розпізнає непогано, хоч і з великою похибкою. Незважаючи на це, виходячи з отриманих результатів, стегоаналітик запідозрить наявність стегоданих у контейнері, оскільки результат RS-методу для класичного LSB-методу при будь-яких випадках не менш як майже 30%.

Що стосується методу на основі дерева Хаффмана, то з табл. 4.12 найвищий показник довжини стегоданих – 8,71%. Дане значення хоч і невелике, але все одно може викликати підозру у стегоаналітика. У даному випадку, слід звернути увагу на те, що для порожнього контейнеру показник RS-методу 7,46%, що всього на 1,25% нижче, ніж для заповненого. Тож можна зробити висновок, що контейнер з розміром 2400x1800 не є хорошим варіантом для вбудовування туди стегоданих.

Отже, проаналізувавши дані у табл. 4.12, можна зробити висновок, що метод на основі дерева Хаффмана є стійкішим від стегоатаки RS-методом у середньому у 17,4 рази. Виходячи з цього, можна вважати, що метод на основі Хаффмана є стійким до статистичного RS-методу.

Ще одним відомим методом стегоаналізу є метод “Хі-квадрат” [116]. За допомогою нього проводяться стегоатаки на кожен канал R, G або B окремо. Результат кожної атаки можна усереднити та отримати результат для повного зображення.

Метод “Хі-квадрат” базується на припущені, що ймовірність одночасної появи сусідніх кольорів у порожньому контейнері досить мала [117]. У даному контексті під сусідніми кольорами мається на увазі кольори, що відрізняються один від одного найменш значущим бітом. При вбудовуванні стегоданих ймовірність одночасної появи сусідніх кольорів значно зростає.

Даний метод рекомендовано застосовувати не для повного зображення, а для певних його ділянок, наприклад рядків або стовпців.

Метод “Хі-квадрат” досить ефективно виявляє послідовне вбудовування у найменш значущі біти. При вбудовуванні у псевдовипадкові пікселі та при умові, що стегодані досить малого розміру відносно розміру контейнера, даний метод не є ефективним. Виходячи з цього, класичний LSB-метод та метод на основі схеми відповідності бітів є вразливими для “Хі-квадрат”. Що стосується методу з фрагментацією стегоданих, то він є більш стійким. Але у випадку, якщо псевдовипадково згенерована довжина блоку буде досить великою, “Хі-квадрат” може розпізнати наявність вбудованих даних у даному блоку. Метод на основі дерева Хаффмана вбудовує інформацію у псевдовипадкові пікселі короткими блоками. Тож, теоретично, він є стійким до стегоатак “Хі-квадрат” методом.

4.3 Аналіз алгоритмічно-програмного методу захисту мультимедійних даних на основі процедури псевдовипадкового вбудовування

Для аналізу методу на основі процедури псевдовипадкового вбудовування порівняємо розроблений метод з аналогами за такими критеріями [31, 70, 92]:

- непомітність вбудовування повідомлення при прослуховуванні аудіофайлу;
- часова складність методу;
- ймовірність декодування даних.

Для порівняння були обрані наступні методи:

- класичний LSB метод;
- метод на основі схеми відповідності бітів;
- метод LSB-стеганографії з фрагментацією стегоданих.

Дані методи було обрано оскільки у них є багато спільного:

- LSB або його модифікація;

- можна змінювати кількість змінюваних біт;
- можуть використовувати ключ (для стандартного LSB – кількість змінюваних біт, розмір стегоданих);
- при вбудовуванні змінюється однакова кількість байт;
- однаковий максимальний розмір повідомлення;
- чутливі до стиснення;
- чутливі до статистичних атак.

Кожен метод було запущено 100 разів з метою зниження похибки. Час роботи методу – час обробки повідомлення, ключів та контейнера без урахування зчитування та запису даних на накопичувач. Апаратне середовищу та інші умови ідентичні як у експериментальних вимірювань у п. 4.1.

Таблиця 4.13 – Час кодування та декодування зображень у мс. при зміні двох молодших біт у контейнер WAV розміром 48185320 байт

Розмір конфіден-ційного зображення	Тип операції	Метод на основі схеми відповід-ності бітів (мс.)	Метод з фрагмен-тацією стегоданих (мс.)	Метод з фрагмен-тацією стегоданих розпарале-лений (мс.)	Метод на основі псевдови-падкового вбудову-вання (мс.)
1600x1600	Вбудову-вання	889,25	338,1	792,8	2916,57
	Декоду-вання	878,47	264	89,6	3057,14
900x900	Вбудову-вання	286,12	118,6	227	1468,87
	Декоду-вання	177,8	82,8	56,6	1182,45
350x350	Вбудову-вання	31,55	22	34,6	874,2
	Декоду-вання	31,72	9	5	444,76

На першому кроці пропонується вбудувати графічні дані у контейнер WAV та порівняти час роботи методів. Як контейнер виступає стерео аудіофайл у WAV форматі cover.wav наступними параметрами з частотою дискретизації 44 100 Герц, двома байтами на один семпл та загальним розміром 48185620 байт. Як повідомлення виступають три зображення з роздільністю 350x350, 900x900 та 1600x1600 пікселів. Документ key.pdf – ключ загальним розміром 446464 байт.

Таблиця 4.14 – Час кодування та декодування мультимедійних даних при зміні двох молодших біт у контейнер WAV розміром 48185320 байт

Розмір конфіден-ційних даних	Формат	Тип операції	Метод на основі схеми відповід-ності бітів (мсек)	Метод з фрагмен-тацією стегода-них (мсек)	Метод з фрагмен-тацією стегода-них розпара-лелений (мсек)	Метод на основі псевдови-падко-вого вбудову-вання (мсек)
3312453	MP3	Вбудову-вання	560,45	215,3	424,8	2596,41
		Декоду-вання	608,71	164,1	129	2504,19
5718710	JPG	Вбудову-вання	1296,53	373,8	721,5	2944,1
		Декоду-вання	957,38	285,7	166,5	3254,72
5354473	MOV	Вбудову-вання	898,31	352,6	702,7	2895,41
		Декоду-вання	890,1	259,2	145,3	3085,9

З табл. 4.13 можна зробити висновок, що метод на основі процедури псевдовипадкового вбудовування є найповільнішим як при вбудовуванні, так і при декодуванні повідомлення. Це пов’язано з тим, що розроблений метод ітерується по всім семплам контейнера незалежно від того, вбудовується туди

конфіденційна інформація, або ні. Як повідомлення виступали зображення у форматі PNG. Метод на основі процедури псевдовипадкового вбудовування дозволяє передавати не тільки зображення, а і інші мультимедійні файли.

З табл. 4.14 видно, що метод на основі процедури псевдовипадкового вбудовування є найповільнішим незалежно від формату конфіденційного файлу.

З точки зору непомітності вбудовування повідомлення при прослуховуванні аудіофайлу, усі методи показують хороший результат – не чутно шумів або інших звуків, що вибиваються з загальної мелодії.

Порівняймо дані методи з точки зору ймовірності підбору ключів для випадку зазначеному у п. 3.4 для декодування стегоданих стегоаналітиком при наступних умовах:

- стегоаналітик знає яким методом вбудовані стегодані;
- стегоаналітик знає розмір конфіденційних графічних даних;
- стегоаналітик знає, що кількість змінюваних біт варіюється від 1 до 4 включно;
- стегоаналітик не знає ключів для декодування (крім розміру, оскільки він виступає як один з ключів для методу на основі процедури псевдовипадкового вбудовування).

Нехай подія F – підбір ключів з першої спроби.

Для класичного LSB-методу ймовірність підбору ключів визначалась у п. 4.1. Вона не змінилась і дорівнює:

$$P(F) = \frac{1}{4}$$

Порахуємо ймовірність підбору ключів для методу на основі схеми відповідності бітів. Вона буде відрізнятися від ймовірності порахованої для графічного контейнера. Для коректного декодування стегоданих методом на основі схеми відповідності бітів, як і у разі графічного контейнеру, треба знати наступні ключі:

- кількість змінюваних біт (4 варіанти);

- логічна функція (4 варіанти);
- схема відповідності бітів.

Єдиний ключ, що буде відрізнятись – схема відповідності бітів. У графічному контейнері у схемі могли використовуватись від 12 до 21 старших біт для використання у логічній функції в залежності від кількості змінюваних біт. У разі контейнера у форматі WAV, старшими бітами можуть виступати незмінювані біти поточного семплу. Оскільки у даному випадку кожне значення семплу описується двома байтами, то можна зробити висновок, що у семплі 16 бітів.

Нехай події H_1, H_2, H_3, H_4– підбір ключів з першої спроби у випадку, що змінено 1, 2, 3 або 4 молодших бітів відповідно. Тоді ймовірність підбору ключів буде дорівнювати:

$$P_{H_1}(F) = \frac{1}{4 \cdot 15},$$

$$P_{H_2}(F) = \frac{1}{4 \cdot 14^2},$$

$$P_{H_3}(F) = \frac{1}{4 \cdot 13^3},$$

$$P_{H_4}(F) = \frac{1}{4 \cdot 12^4}.$$

Отже, загальна ймовірність підбору ключів можна визначити за формулою:

$$P_{H_1H_2H_3H_4}(F) = \frac{1}{\sum_{i=1}^{b=4}(4 \cdot (16-i)^i)} = \frac{1}{4 \cdot 15 + 4 \cdot 14^2 + 4 \cdot 13^3 + 4 \cdot 12^4} = \frac{1}{60 + 784 + 10976 + 153664} \approx 6{,}04 \cdot 10^{-6}$$

Порахуємо ймовірність підбору ключів для методу з фрагментацією стегоданих для випадку, коли контейнером виступає аудіофайл. Для коректного декодування стегоданих методом з фрагментацією стегоданих, треба знати наступні ключі:

- кількість змінюваних біт (4 варіанти);
- кількість блоків;
- довжини блоків.

Оскільки розмір стегоданих стегоаналітику відомі, то максимальну кількість блоків для вбудовування можна визначити за формулою:

$$m = \frac{8 \cdot y}{b}, \tag{4.12}$$

де m – максимальна кількість блоків;

y – розмір конфіденційного файлу у байтах;

b – кількість змінюваних біт.

Кількість блоків n у які можна вбудовувати, по суті, є масивом семплів аудіоконтейнера.

Ймовірність підбору ключів для методу з фрагментацією стегоданих з першої спроби можна визначити за допомоги формули розміщень з комбінаторики:

$$P(F) = \frac{1}{\sum_{i=1}^{b=4} A_n^{\frac{8 \cdot y}{i}}} = \frac{1}{\sum_{i=1}^{b=4} \left(\frac{n!}{\left(n - \frac{8 \cdot y}{i}\right)!} \right)}$$

Визначимо ймовірність підбору ключів для методу на основі процедури псевдовипадкового вбудовування. Для коректного декодування стегоданих необхідно знати наступні ключі:

- зерно для першого генератора псевдовипадкових чисел (4294967296 варіантів);
- довільний файл;
- розмір повідомлення у байтах;
- логічна функція (4 варіанти);
- зерно для другого генератора псевдовипадкових чисел (4294967296 варіантів);

Нехай один з ключів, а саме розмір стегоданих, стегоаналітику уже відомо. Кількість змінюваних біт у підрахунку ймовірності не враховується, оскільки визначено, що вона дорівнює кількості байтів для одного семплу. Ймовірність

підбору ключів з першого разу для методу на основі процедури псевдовипадкового вбудовування визначається за формулою:

$$P(F) = \frac{1}{4 \cdot 4294967296^2 \cdot 2^{8 \cdot s}} \approx \frac{1{,}36 \cdot 10^{-20}}{2^{8 \cdot y}},$$

де y – розмір конфіденційного файлу у байтах.

Порівняємо ймовірність підбору ключів стегоаналітиком для класичного LSB-методу, методу на основі схеми відповідності бітів, методу з фрагментацією стегоданих та методу на основі процедури псевдовипадкового вбудовування персональних конфіденційних мультимедійних даних користувачів мережі Інтернет.

Таблиця 4.15 – Ймовірності підбору ключів до методів з першої спроби

Назва методу	Ймовірність підбору ключів
Класичний LSB-метод	$P(F) = \frac{1}{4}$
Метод на основі схеми відповідності бітів	$P(F) \approx 6{,}04 \cdot 10^{-6}$
Метод з фрагментацією стегоданих	$P(F) = \frac{1}{\sum_{i=1}^{b=4} \left(\frac{n!}{\left(n - \frac{8 \cdot y}{i}\right)!} \right)}$
Метод на основі процедури псевдовипадкового вбудовування	$P(F) \approx \frac{1{,}36 \cdot 10^{-20}}{2^{8 \cdot y}}$

З табл. 4.15 можна зробити висновок, що ймовірність підбору ключів до методу на основі процедури псевдовипадкового вбудовування набагато менша, ніж до класичного LSB-методу або методу на основі схеми відповідності бітів. Але порівнюючи розроблений метод з методом на основі фрагментації

стегоданих, ймовірність підбору ключів до методу на основі фрагментації стегоданих у переважній більшості випадків буде меншою.

У п. 4.2 було проаналізовано ймовірність підбору ключа до методу з фрагментацією даних з точки зору програмної реалізації. Отже, ймовірність підбору ключі для даного методу дорівнює:

$$P(F) = \frac{1}{2^{32}}$$

Після аналізу програмної реалізації методу на основі процедури псевдовипадкового вбудовування, ймовірність декодування не зміниться, оскільки 4 з 5 (розмір стегоданих відомі за замовчуванням) ключів залишаються невідомими для стегоаналітика.

Таким чином, можна зробити висновок, що метод на основі процедури псевдовипадкового вбудовування є найстійкішим з точки зору підбору ключів для декодування повідомлення.

Порівнюючи метод на основі процедури псевдовипадкового вбудовування та метод з фрагментацією стегоданих, перший має суттєву перевагу, адже кожне згенероване число використовується та загальну кількість ітерацій при роботі з конкретними даними при необхідності можна порахувати. Якщо повідомлення буде дорівнювати максимальному розміру повідомлення, що можна вбудувати у конкретний контейнер, і під кінець процесу вбудовування методом з фрагментацією стегоданих залишиться лише одна вільна адреса у контейнері, то теоретично, ця адреса може бути ніколи не згенерована ГПВЧ, а отже, процес вбудовування може бути нескінченним, так як ГПВЧ мають властивість зациклюватись та видавати одні й ті ж самі числа.

Для стегоаналізу заповненого контейнера часто використовують метод “Хі-квадрат”. Він ефективно виявляє послідовне вбудовування у найменш значущі біти. При вбудовуванні конфіденційної інформації у псевдовипадкові семпли аудіофайлу та при умові, що конфіденційний файл досить малого розміру відносно розміру контейнера, даний метод не є ефективним. Виходячи з цього,

метод на основі процедури псевдовипадкового вбудовування вбудовує інформацію у псевдовипадкові семпли короткими блоками по *n* бітів. Тож, теоретично, він є стійким до стегоатак “Хі-квадрат” методом.

4.4 Порівняльний аналіз методів

У дослідженні було проведено порівняльний аналіз існуючих та розроблених крипто-стеганографічних методів для роботи з графічними та аудіоданими. Результати аналізу методів представлені у табл. 4.16 та табл. 4.17.

Таблиця 4.16 – Порівняння методів

Критерії порівняння	LSB	Метод на основі дерева Хаффман	Метод стеганографії зображень із використанням дерева Хаффмана та матриці міжпіксельних різниць	Метод на основі модифікації колірних параметрів	Метод стеганографії зображень із використанням кодування Хаффмана та чотирьохрівневої процедури зберігання
Можливість працювати без ключів	Так	Ні	Так	Так	Так
Змінна кількість ключів	Ні	Так	Відсутні ключі	Так	Відсутні ключі
Ймовірність підбору ключа	0,25	3,147*10^ (-512)	-	Висока	-
Підозрілість ключів (по місяцях)	1	1	Відсутні ключі	2	Відсутні ключі
Швидкість роботи (по місцях)	1	3	3	2	3
Стійкість до статичного аналізу	Ні	Так	Ні	Ні	Ні
Можливість перевірки вбудовування у обраний контейнер до початку	Так	Ні	Ні	Так	Ні
Можливість працювати з графічними даними та аудіо	Так	Тільки графічні дані	Тільки графічні дані	Тільки графічні дані	Тільки графічні дані
Візуальна/аудіо стійкість	Так	Так	Так	Ні	Так

З табл. 4.16 та 4.17 видно, що метод на основі дерева Хаффмана є найбільш захищеним з точку зору ймовірності підбору ключа. Разом з тим, він є стійким до статистичних атак та має ключ у вигляді зображення, що не є підозрілим при передачі отримувачу. Отже, можна зробити висновок, що у випадку, якщо контейнером виступають графічні дані, метод на основі дерева Хаффмана є найнадійнішим.

Таблиця 4.17 – Порівняння методів

Критерії порівняння	Метод на основі схеми відповідності бітів	Метод з фрагментацією стегоданих	Метод на основі процедури псевдовипадкового вбудовування	Метод на основі алгоритму AES
Можливість працювати без ключів	Так	Ні	Ні	Ні
Змінна кількість ключів	Так	Ні	Так	Ні
Ймовірність підбору ключа	2,79*10^(-14)	2,33*10^(-11)	(1,36*10^(-20))/2^(8*s)	8,63^(-78)
Підозрілість ключів (по місяцях)	3	3	2	3
Швидкість роботи (по місцях)	1	2	3	3
Стійкість до статичного аналізу	Ні	Так/Ні	Так/Ні	Ні
Можливість перевірки вбудовування у обраний контейнер до початку	Так	Так	Так	Так
Можливість працювати з графічними даними та аудіо	Так	Так	Тільки аудіо, але можлива адаптація для графічних даних	Так
Візуальна/аудіо стійкість	Так	Так	Так	Так

Що стосується роботи з аудіоданими, то найбільш захищеними методами з точки зору ймовірності підбору ключів є метод на основі процедури псевдовипадкового вбудовування та метод на основі алгоритму AES. Обидва методи працюють повільніше за метод на основі схеми відповідності та метод з

фрагментацією стегоданих. Метод на основі процедури псевдовипадкового вбудовування має суттєві переваги перед методом на основі алгоритму AES, а саме:

- має менш підозрілий ключ (файл та число);
- може використовувати допоміжні ключі для більш високої надійності;
- є стійким до певних видів статистичних атак;
- має можливість вбудовувати шум.

Метод на основі схеми відповідності є найшвидшим серед розглянутих методів (не враховуючи стандартного LSB) та має можливість легкого об`єднання з іншими LSB-модифікаціями для підвищення їх стеганографічної швидкості.

4.5 Висновки до розділу 4

Аналіз розроблених алгоритмічно-програмних методів, результати якого представлено у цьому розділі, дозволяє зробити загальний висновок про їх ефективність. Крім того, проведене дослідження дозволяє зробити такі висновки.

1. Найбільш ефективним з точки зору швидкодії є алгоритмічно-програмний метод на основі схеми відповідності бітів, що досягається використанням логічних функцій. Особливістю цього методу є можливість його інтеграції з іншими методами LSB-стеганографії, оскільки він виконується на етапі самого вбудовування і не впливає на первинне оброблення стегоданих. Таке поєднання методів дозволяє підвищити стійкість захисту даних. Метод на основі схеми відповідності бітів доцільно використовувати у випадку, коли важливим є час оброблення даних.

2. Особливістю методу на основі дерева Хаффмана є залежність часу виконання процедури побудови дерева від розміру ключа. Якщо протягом певного періоду часу використовується один і той самий ключ, метод на основі

Хаффмана демонструє прийнятні часові показники. При побудові нового дерева алгоритмічно-програмний метод на основі дерева Хаффмана характеризується відносно низькими часовими показниками. Проте, якщо дерево побудовано, то при вбудовуванні даний метод показує задовільні результати. Крім того, цей метод є найстійкішим до підбору ключів. Також цей метод є стійким до візуального стегоаналізу та до статистичного аналізу.

3. Особливостями методу на основі процедури псевдовипадкового вбудовування є застосування двох генераторів псевдовипадкових чисел для визначення позицій непослідовного вбудовування, застосування процедури вбудовування шуму та змінна кількість ключів, що дозволяє забезпечити високий рівень захисту даних (ймовірність підбору ключа складає $\frac{1{,}36 \cdot 10^{-20}}{2^{8 \cdot y}}$, де y – розмір повідомлення у байтах). Проте додаткові процедури захисту сповільнюють процес оброблення даних, тому метод на основі процедури псевдовипадкового вбудовування доцільно використовувати у випадках, коли є висока ймовірність несанкціонованого доступу до даних, а часом оброблення даних можна знехтувати.

ВИСНОВКИ

У дисертаційній роботі вирішено актуальну науково-технічну проблемну задачу підвищення ефективності програмних засобів крипто-стеганографічного захисту мультимедійних даних користувачів мережі Інтернет за рахунок вдосконалення архітектури програмної системи захисту та розроблення нових алгоритмічно-програмних методів крипто-стеганографічного захисту мультимедійних даних, що забезпечують надійність захисту та підвищення швидкодії процедур оброблення мультимедійних даних користувача.

В результаті дисертаційного дослідження отримано такі основні наукові результати:

1. Показано, що для захисту персональних мультимедійних даних користувачів Інтернет доцільно розроблювати програмне забезпечення на основі методів крипто-стеганографічного захисту даних.
2. Запропоновано універсальну архітектуру програмної системи захисту мультимедійних даних користувачів мережі Інтернет, визначальними рисами якої є можливість використання довільних методів крипто-стеганографічного захисту, легкість масштабування та адаптації до отримання вхідних даних з різних програмних середовищ, що дозволяє спростити процес розроблення програмних систем захисту мультимедійних даних.
3. Розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, який відрізняється від відомих застосуванням схеми відповідності бітів і логічної функції та характерною рисою якого є можливість поєднання з існуючими методами LSB-стеганографії, що дозволяє поліпшити стеганографічну стійкість програмного забезпечення та підвищити швидкодію процедури вбудовування стегоданих у понад 5 разів. Показано, що метод на основі

схеми відповідності бітів доцільно використовувати у разі обмежених обчислювальних ресурсів.

4. Розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, особливістю якого є використання процедури побудови дерева Хаффмана на основі зображення-ключа, що дозволяє підвищити рівень захисту конфіденційних графічних даних від підбору ключів і статистичних стегоатак у середньому у 17,4 разів. Показано, що метод на основі дерева Хаффмана доцільно використовувати у випадках, коли важливими є як стеганографічна стійкість, так і швидкодія процедури оброблення мультимедійних даних користувача.
5. Розроблено алгоритмічно-програмний метод крипто-стеганографічного захисту мультимедійних даних, який, на відміну від відомих, ґрунтується на використанні процедури псевдовипадкового вбудовування даних із застосуванням двох генераторів псевдовипадкових чисел, процедури вбудовування шуму та змінної кількості ключів, що дозволяє підвищити рівень захисту мультимедійних даних, зокрема, забезпечити високу стійкість до підбору ключів. Показано, що метод на основі процедури псевдовипадкового вбудовування доцільно використовувати у випадках, коли є висока ймовірність несанкціонованого доступу до даних, проте часом оброблення даних можна знехтувати.
6. Розроблено експериментальну програмну систему для дослідження запропонованої універсальної архітектури програмного забезпечення процесу захисту мультимедійних даних та розроблених алгоритмічно-програмних методів захисту мультимедійних даних користувачів мережі Інтернет.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Абазина Е. С. Цифровая стеганография: состояние и перспективы / Е. С. Абазина, А. А. Урунов. – 2016. – №2. – С. 182.
2. Аграновский А.В. Стеганография, цифровые водяные знаки и стеганоанализ / Аграновский А.В., Балакин А.В., Грибунин В.Г., Сапожников С. // М.: Вузовская книга. – 2009. – С. 220.
3. Белоус А. СВЧ-электроника в системах радиолокации и связи. Техническая энциклопедия. Книга 2 / А. Белоус, С. Шведов, М. Мерданов. – 2019. – С. 1011.
4. Бенкен Е. С. PHP, MySQL, XML: программирование для Интернета. (+CD) (3-е изд.) / Е. С. Бенкен. – Санкт-Петербург: БХВ-Петербург, 2011. – 304 с. – С. 223.
5. Білинський Й. Електронні системи / Й. Білинський, К. Огородник, М. Юкиш., 2017.
6. Бількевич Д. Програмна інженерія та моделювання складних розподілених систем [Електронний ресурс] / Д. Бількевич, Д. Михалик – Режим доступу до ресурсу: http://elartu.tntu.edu.ua/bitstream/lib/27287/2/IMST_2018_Bilkevych_D_A-Rozrobka_krosplatformenoho_94.pdf – [23.03.2020].
7. Боренков А. Ю. Windows XP. Библиотека пользователя / А. Ю. Боренков, Ю. Н. Зозуля., 2006. – 496 с. – С. 266.
8. Вахаб А. Методы цифровой стеганографии на основе модификации цветовых параметров изображения / А. Вахаб, Д. Романенко. // Труды БГТУ. – 2018. – №3. – С. 94–98.

9. Воронюк А. Актуальный интернет-маркетинг. Как привлекать клиентов и повышать продажи через интернет / А. Воронюк., 2018. – 160 с.
10. Гамма Э. Приемы объектно-ориентированного проектирования / Э. Гамма, Р. Хелм, Р. Джонсон., 2013. – 368 с. – С. 183.
11. Гололобов А. В. Стеганографическое скрытие данных в полях битовых плоскостей изображений / Гололобов А. В., 2015. – С. 32.
12. Гребенніков В. Комплексні системи захисту інформації. Проектування, впровадження, супровід / В. Гребенніков., 2019.
13. Гребенніков В. Стеганографія. Історія спецзв'язку / В. Гребенніков., 2018.
14. Грибунин В. Цифровая стеганография / В. Грибунин, И. Оков, И. Туринцев., 2018. – С. 11.
15. Губенко Н. Е. Анализ особенностей методов цифровой стеганографии для защиты информации, передаваемой по открытым каналам [Електронний ресурс] / Н. Е. Губенко, Д. С. Сипаков // Донецкий национальный технический университет (Донецк). – 2015. – Режим доступу до ресурсу: https://www.elibrary.ru/item.asp?id=29357976 – [17.03.2020].
16. Дмитриев А. Е. Обзор современных методов сокрытия информации / А. Е. Дмитриев. – С. 1.
17. Дьяконов В. Вейвлеты. От теории к практике / В. Дьяконов., 2010. – 400 с. – С. 276.
18. Жилкин М. Ю. Стегоанализ графических данных на основе методов сжатия / М. Ю. Жилкин. // Вестник СибГУТИ. – 2008. – №2. – С. 62–66.

19. Зорин В. Что такое геотегинг? [Електронний ресурс] / В. Зорин, Д. Каменев. – 2008. – Режим доступу до ресурсу: https://web.archive.org/web/20140328095445/http://gps-club.ru/detail.php?ID=16234 – [23.03.2020].

20. Клетте Р. Компьютерное зрение. Теория и алгоритмы / Р. Клетте., 2019. – С. 62.

21. Кнут Д. Искусство программирования. Том 2. Получисленные алгоритмы / Д. Кнут., 2018. – С. 29-34.

22. Коробейников А. Г. Выбор коэффициентов матрицы дискретно-косинусного преобразования при построении стеганографических систем. / А. Г. Коробейников, Н. Н. Прохожев, О. В. Михайличенко. // Вестник компьютерных и информационных технологий.. – 2008. – №11. – С. 12–17.

23. Кошкина Н. В. Обзор и классификация методов стеганоанализа / Н. В. Кошкина. // Управляющие системы и машины. – 2015. – №3. – С. 3 – 12.

24. Лафоре Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – Санкт-Петербург: Питер, 2004. – С. 33 – 37.

25. Леоненков А. В. Самоучитель UML (2 изд.) / А. В. Леоненков., 2006. – 432 с. – С. 133.

26. Лужецький В. А. Основи інформаційної безпеки. Навчальний посібник. / В. А. Лужецький, А. Д. Кожухівський, О. П. Войтович. – Вінниця: ВНТУ, 2009. – 268 с. – С. 194.

27. Мартинюк О. М. Основи дискретної математики / О. М. Мартинюк., 2008. – С. 163.

28. Молдовян Н. А. Криптография: от примитивов к синтезу алгоритмов

/ Н. А. Молдовян., 2004. – 448 с. – С. 115.

29. Музыкантский А. И. Лекции по криптографии / А. И. Музыкантский, В. В. Фурин. – Москва, 2014. – 66 с. – С. 4.

30. Орлов С. А. Программная инженерия. Учебник для вузов. 5-е издание обновленное и дополненное. Стандарт третьего поколения / С. А. Орлов., 2020. – 640 с. – С. 307.

31. Очимов С. Ю. Стегоанализ аудиофайлов, базирующийся на алгоритмах сжатия / С. Ю. Очимов. // Вестник СибГУТИ. – 2010. – №1. – С. 33–40.

32. Павлов С. В. Алгоритми ущільнення цифрових зображень [Електронний ресурс] / С. В. Павлов, Р. Ю. Довгалюк // Оптико-електронні інформаційно-енергетичні технології. – 2010. – № 2. – С. 194-198. – Режим доступу: http://nbuv.gov.ua/UJRN/oeiet_2010_2_32 – [17.03.2020].

33. Панасенко С. Смарт-карты и информационная безопасность / С. Панасенко, К. Мытник., 2019. – С. 180.

34. Парфенов В. Представлена камера с самым большим сенсором и разрешением 102 МП [Електронний ресурс] / В. Парфенов. – 2019. – Режим доступу до ресурсу: https://www.popmech.ru/gadgets/news-482831-predstavlena-kamera-s-samym-bolshim-sensorom-i-razresheniem-102-mp/ – [23.03.2020].

35. Перов Д. Ю. Анализ модификаций стеганографического метода LSB [Електронний ресурс] / Д. Ю. Перов, А. Л. Исаев. – 2015. – Режим доступу до ресурсу: http://ainsnt.ru/doc/799240.html – [17.03.2020].

36. Пикус Ф. Г. Идиомы и паттерны проектирования в современном C++

/ Ф. Г. Пикус., 2019. – С. 292.

37. Привалов А. Внутри MP3. А как оно всё устроено? [Електронний ресурс] / А. Привалов. – 2010. – Режим доступу до ресурсу: https://habr.com/ru/post/103635/ – [23.03.2020].

38. Пузиренко О. Ю. Комп'ютерна стеганографічна обробка й аналіз мультимедійних даних: Підручник / О. Ю. Пузиренко, Д. О. Прогонов, Г. Ф. Конахович., 2018. – 558 с. – С. 12-15.

39. Радченко Є. О. Метод стеганографічного захисту WAV-файлів / Є. О. Радченко, Є. С. Сулема // Збірник тез доповідей 12-ї наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2019). – НТУУ «КПІ». – 2019. – С. 99-104.

40. Радченко Є. О. Спосіб стеганографічного захисту графічних даних на основі схеми відповідності бітів та аналізу візуальних властивостей контейнера / Є. О. Радченко, Є. С. Сулема // Тези доповідей шостої міжнародної науково-практичної конференції "Методи та засоби кодування, захисту й ущільнення інформації". – Вінниця. – 2017. – С. 51-53.

41. Райтман М. А. Искусство легального анонимного и безопасного доступа к ресурсам Интернета / М. А. Райтман., 2017. – 624 с. – С. – 51.

42. Рева И. Исследования генератора акустического и виброакустического шума с перенастраиваемой огибающей спектра / И. Рева., 2018. – С. – 29.

43. Скляр Б. Цифровая связь: Теоретические основы и практическое применение / Б. Скляр., 2004. – 1104 с. – С. – 879.

44. Смірнов, О. А. Дослідження методів стегоаналізу цифрових зображень / О. А. Смірнов, Є. В. Мелешко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків.: ХУПС, 2012. – Вип. 2 (8). – С. 92–99.

45. Страуструп Б. Программирование. Принципы и практика использования C++ / Б. Страуструп., 2019.

46. Сулема Є. С. Метод стеганографічного захисту мультимедійних даних на основі процедури псевдовипадкового вбудовування / Є. С. Сулема, Є. О. Радченко. // Наукові вісті КПІ ім. Ігоря Сікорського. – 2020. – №1. – С. 40–47.

47. Сулема Є. С. Algorithm of graphical data stegonagraphic protection based on bits difference transform / Є. С. Сулема, Є. О. Радченко // Збірник тез доповідей 8-ї наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» (ПМК-2016). – НТУУ «КПІ». – 2016. – С. 254-258.

48. Теґмарк М. Життя 3.0. Доба штучного інтелекту / М. Теґмарк., 2019. – 432 с.

49. Федоренко Ю. Алгоритмы и программы на C++ Builder / Ю. Федоренко., 2017. – С. 429.

50. Фергюсон Н. Практическая криптография / Н. Фергюсон, Б. Шнайер. – Киев: Издательский дом Вильямс, 2004. – 432 с. – С. 39.

51. Филиппов М. В. Метод сокрытия информации в квантованных коэффициентах дискретного косинус преобразования [Електронний ресурс] / М. В. Филиппов, С. А. Балашtions

52. Шаханова М. В. Современные технологии информационной безопасности. Учебно-методический комплекс / М. В. Шаханова., 2015. – 205 с.

53. Швидченко И. В. Методы стеганоанализа для графических файлов / И. В. Швидченко. // Штучний інтелект. – 2010. – №4. – С. 697–705.

54. Шеховцов В. А. Операційні системи / В. А. Шеховцов. – Київ: BHV Publishing Group, 2005. – С. – 19.

55. Широчин С.С. Алгоритм фрагментації стегоданих у стеганографії зображень. / С.С. Широчин, Є.С. Сулема // Збірник тез доповідей четвертої наукової конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг (ПМК-2012)». – Київ. – 2012. – С. 299-303.

56. Широчин С.С. Метод захисту зображень на основі шифрування палітри / Сулема Є.С., Широчин С.С. // Вісник Хмельницького національного університету. – Хмельницький : ХНУ. – 2014.– №3. – С. 114-119.

57. Широчин С. С. Методи комбінованого стеганографічного захисту мультимедійних даних в хмарних сховищах : дис. канд. техн. наук : 05.13.05 / Широчин С. С. – Київ, 2015.

58. Широчин, С.С. Спосіб стеганографії зображень з фрагментацією стегоданих та розділенням закритого ключа / Є.С. Сулема, С.С. Широчин // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. – 2012. – Вип. 1 (22). – С. 64-68.

59. Эклер Ю. Прогрессивный самоучитель работы на компьютере / Ю. Эклер., 2017. – С. 238.

60. Adak C. Robust Steganography Using LSB-XOR and Image Sharing / C. Adak. // International Conference on Computation and Communication Advancement. – 2013. – P. 99–102.

61. Adams C. Understanding PKI: Concepts, Standards, and Deployment Considerations / C. Adams, S. Lloyd., 2003. – 322 p. – P. 18.

62. Adebayo J. K. Issues on E-health Adoption in Nigeria / J. K. Adebayo, E. O. Ofoegbu. // International Journal Modern Education and Computer Science. – 2014. – №6. – P. 36–46.

63. Alfonse M. An Ontology-Based System for Cancer Diseases Knowledge Management / M. Alfonse, M. M. Aref, A. M. Salem. // International Journal Information Engineering and Electronic Business. – 2014. – №6. – P. 55–63.

64. Andersen R. J. On the limits of steganography / R. J. Andersen, F. Petitcolas. // IEEE Journal of Selected Areas in Communications, Special Issue on Copyright and Privacy Protection 16. – 1998. – №4. – P. 474–481.

65. Anderson R. Serpent: A Proposal for the Advanced Encryption Standard [Електронний ресурс] / R. Anderson, E. Biham, L. Knudsen – Режим доступу до ресурсу: http://cryptosoft.net/docs/Serpent.pdf. – [25.03.2020].

66. Baritha Begum M. LSB Based Audio Steganography Based On Text Compression / M. Baritha Begum. // Procedia Engineering. – 2012. – №30. – P. 703–710.

67. Bhattacharjee K. A Search for Good Pseudo-random Number Generators: Survey and Empirical Studies [Електронний ресурс] / K. Bhattacharjee, K. Maity, S. Das // Department of Information Technology, Indian Institute of Engineering Science and Technology, Shibpur, West Bengal, India. – 2018. –

Режим доступу до ресурсу: https://arxiv.org/abs/1811.04035 – [17.03.2020].

68. Boyd C. Protocols for Authentication and Key Establishment / C. Boyd, A. Mathuria, D. Stebila., 2020. – P. 448.

69. Cummins, J. Steganography And Digital Watermarking / Jonathan Cummins, Patrick Diskin, Samuel Lau and Robert Parlett. // – School of Computer Science, The University of Birmingham., 2004.

70. Cvejic N. Increasing the capacity of LSB based audio steganography / N. Cvejic, T. Seppänen. // Proc. 5th IEEE International Workshop on Multimedia Signal Processing. – 2002. – P. 336–338.

71. Daemen J. The Design of Rijndael: AES – The Advanced Encryption Standard / J. Daemen, V. Rijmen. // Springer Science & Business Media. – 2013. – P. 2.

72. Das R. A Novel Steganography Method for Image Based on Huffman Encoding / R. Das, T. Tuithung. // IEEE. – 2012.

73. De Paz J. F. Ambient Intelligence – Software and Applications – 8th International Symposium on Ambient Intelligence (ISAmI 2017) / J. F. De Paz, V. Julián, G. Villarrubia. // Springer. – 2017. – P. 42.

74. Dhotre I. A. Cryptography And Network Security / I. A. Dhotre, V. S. Bagad. – 2008. – P. 44.

75. Dixon W. J. Introduction to Statistical Analysis / W. J. Dixon, F. J. Massey. // McGrawHill Book Company, Inc.. – 1957.

76. Donahoo M. J. TCP/IP Sockets in C: Practical Guide for Programmers / M. J. Donahoo, L. C. Kenneth. – Boston, 2009. – 216 p. – Morgan Kaupfmann Publisher. – C. 2.

77. Dongarra J. High Performance Computing and Communications / J. Dongarra, L. T. Yang, O. F. Rana. // Springer Science & Business Media. – 2005. – P. 368.

78. Dumitrescu S. Detection of LSB Steganography via Sample Pair Analysis / S. Dumitrescu, X. Wu, Z. Wang. // IEEE Transactions on Signal Processing. – 2003. – №51.

79. Dychka I. Analysis of Parallel Computations Efficiency for User's Private Multimedia Data Protection in Clouds / I. Dychka, S. Shyrochyn, Y. Sulema. // Research Bulletin of the National Technical University of Ukraine "Kyiv Polytechnic Institute". – 2016. – №1. – P. 40–46.

80. Fridrich J. Reliable Detection of LSB Steganography in Color and Grayscale Images / J. Fridrich, M. Goljan, R. Du. – 2002.

81. Fridrich J. Steganalysis based on JPEG compatibility / J. Fridrich, M. Goljan, R. Du. // SPIE Multimedia Systems and Applications IV. – 2001.

82. Fridrich J. Steganalysis of LSB Encoding in Color Images / J. Fridrich. // Proceedings of the IEEE International Conference on Multimedia. – 2000. – P. 1279–1282.

83. Hillar G. MQTT Essentials – A Lightweight Experience from the following is meriting:t IoT Protocol / C. Hillar., 2017. – 280 p. – Packt Publishing Ltd. - P. 9.

84. Hu Z. Graphical Data Steganographic Protection Method Based on Bits Correspondence Scheme / Z. Hu, I. Dychka, Y. Sulema, Y. Radchenko. // China : Hong Kong MECS Press.. – 2017. – P. 34–40.

85. Huffman D. A. A Method for the Construction of Minimum-Redundancy

Codes / D. A. Huffman., 1952. – P. 1098-1101.

86. Jayant A. C++ Algorithm Series: Binary Trees and Binary Search Trees / A. Jayant., 2019.

87. Johnson N. F. Steganography: Seeing the Unseen / N. F. Johnson, S. Jajodia. // IEEE Computer. – 1998. – P. 26–34.

88. Johnson N. F. Steganalysis of Images Created Using Current Steganography Software / N. F. Johnson, S. Jajodia. // Information Hiding, LNCS 1525, Springer-Verlag Berlin Heidelberg. – 1998. – P. 32–47.

89. Kaur H. A Survey on different techniques of steganography / H. Kaur, J. Rani. // MATEC Web of Conferences. – 2016. – №57.

90. L'Ecuyer P. Fast random number generators based on linear recurrences modulo 2: Overview and comparison / P. L'Ecuyer, F. Panneton. // Proceedings of the 37th Conference on Winter Simulation, WSC '05, Winter Simulation Conference. – 2005. – P. 110–119.

91. Lampkin V. Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry / V. Lampkin, W. T. Leong, L. Olivera. // IBM Redbooks. – 2012. – P. 2.

92. Lee Y. K. High Capacity Image Steganographic Model / Y. K. Lee, L. H. Chen. // IEEE Proceedings Vision, Image and Signal Processing. – 2000. – P. 288–294.

93. Malik A. A Modified Pixel-Value Differencing Image Steganographic Scheme with Least Significant Bit Substitution Method / A. Malik, G. Sikka, H. K. Verma. // International Journal Image, Graphics and Signal Processing. – 2015. – №7. – P. 68–74.

94. Matsuura K. Advances in Information and Computer Security: Third International Workshop on Security / K. Matsuura, E. Fujisaki. – Kagawa, Japan: Springer, 2008. P. 266.

95. Nath A. The Modern Cryptography Cookbook: Learn Crypto Principle to Applied Cryptography with example / A. Nath., 2018. – 240 p. – P. – 46.

96. Morkel T. An overview of image steganography / T. Morkel, J. Eloff, M. S. Olivier. // The Fifth Annual Information Security South Africa Conference. – 2005.

97. Mwammenywa I. A. Towards Enhancing Access of HIV/AIDS Healthcare Information in Tanzania: Is a Mobile Application Platform a Way Forward? / I. A. Mwammenywa, S. F. Kaijage. // International Journal Information Technology and Computer Science. – 2018. – №10. – P. 31–38.

98. Oppliger R. SSL and TLS: Theory and Practice, Second Edition Artech House information security and privacy series / R. Oppliger., 2016. – 304 p. – Artech House. – P. 21.

99. Oppliger R. SSL and TLS: Theory and Practice, Second Edition Artech House information security and privacy series / R. Oppliger., 2016. – 304 p. – Artech House. – P. 91.

100. Pal S. K. The Future of Audio Steganography / S. K. Pal, P. K. Saxena, S. K. Mutto. // Pacific Rim Workshop on Digital Steganography. – 2002.

101. Pandian N. An Image Steganography Algorithm Using Huffman and Interpixel Difference Encoding / Nithyanandam Pandian. // International Journal of Computer Science & Security. – 2014. – №8. – P. 202–215.

102. Provos N. Statistical steganalysis / N.Provos // University of Michigan. –

2003.

103. Richardson I. The H.264 Advanced Video Compression Standard / I. E. Richardson. – 2011. – 346 p.

104. Ristic I. Bulletproof SSL and TLS: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications / I. Ristic. – 2013. – Feisty Duck. – P. 1.

105. Saint-Andre P. XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies / P. Saint-Andre, K. Smith, R. Troncon., 2009. – 310 p. – O'Reilly Media, Inc. – P. 3.

106. Saint-Andre P. XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies / P. Saint-Andre, K. Smith, R. Tronçon., 2009. – 310 p. – O'Reilly Media, Inc. – P. 12.

107. Saito M. SIMD-Oriented Fast Mersenne Twister: a 128- bit Pseudorandom Number Generator / M. Saito, M. Matsumoto. // Springer Berlin Heidelberg. – 2008. – P. 607–622.

108. Sethi N. Steganography Technique with Huffman Code / N. Sethi. // International Journal of Recent Technology and Engineering. – 2019. – №8. – P. 866–870.

109. Seyyedi S. A. Statistical Image Classification for Image Steganographic Techniques / S. A. Seyyedi, N. Ivanov. // International Journal Image, Graphics and Signal Processing. – 2014. – №6. – P. 19–24.

110. Stanislav M. Two-Factor Authentication / M. Stanislav., – 2005. – 104 p. – IT Governance Ltd. – P. 27.

111. Sulema Y. Criteria of Search of Optimal Placement of Stegodata Blocks

in Container / Y. Sulema, S. Shyrochyn. // Proceedings of the International Conference “System Analysis and Information Technologies” (SAIT 2011). – 2011. – P. 516.

112. Sulema Y. Image Protection Method Based on Binary Operations / Y. Sulema. // Proceedings of the 23rd IEEE International Conference on Systems, Signals and Image Processing IWSSIP2016. – 2016. – P. 295–298.

113. Sulema Y. Information System for Archival Medical Images Automated Processing / Y. Sulema, O. Shkurat. // The 3rd International Conference Health Technology Management. – 2016. – P. 72.

114. Viega J. Network Security with OpenSSL: Cryptography for Secure Communications / J. Viega, M. Messier, P. Chandra. – 2002. – 386 p. – P. 175.

115. Waggener W. Pulse Code Modulation Systems Design Artech House telecommunications library Telecommunications Library / William N. Waggener. – 1999. – 322 p.

116. Westfeld A. Attacks on Steganographic Systems / A. Westfeld, A. Pfitzmann. // Springer-Verlag. – 2000. – №1768. – P. 70.

117. Westfeld A. Attacks on Steganographic Systems / A. Westfeld, A. Pfitzmann. // Springer-Verlag. – 2000. – №1768. – P. 71.

118. Yaremchuk Y. The digital watermarks usage for copyright protection in the images / Y. Yaremchuk, V. Karpinets. // Legal, Normative and Metrological Provision of the Information Protection system in Ukraine. – 2006. – №2. – P. 63–70.

119. Аудио конвертер онлайн [Електронний ресурс] – Режим доступу до ресурсу: https://online-audio-converter.com/ru/ – [23.03.2020].

120. Дослідження кольорових цифрових фотографій методами RS-стегоаналізу та статистики / В. Ю.Корольов, В. В. Поліновський, В. А. Герасименко, М. Л. Горинштейн. // Інформація і право. – 2011. – №3. – С. 102–110.

121. Засоби захисту інформації [Електронний ресурс] – Режим доступу до ресурсу: https://ssbb.com.ua/uk/poshuk-i-vyyavlennya-proslyshky/poshuk-zakladnykh-ustrojstv/sredstva-zashity-informacii/ – [23.03.2020].

122. Защита от хакеров корпоративных сетей, 2017 / Litres, – С. 1283.

123. Ідентифікація і захист мультимедійних даних / [В. Г. Іванов, М. Г. Любарський, В. В. Карасюк та ін.]. // М-во оборони України, Харк. ун-т Повітрян. Сил ім. Івана Кожедуба. – 2012. – №8. – С. 87–91.

124. Колір й світосприйняття [Електронний ресурс] – Режим доступу до ресурсу: https://works.doklad.ru/view/IBQkH3iYXyE.html – [23.03.2020].

125. Первая в мире камера на 40 мегапикселей: компания huawei презентовала смартфон p20 pro [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: http://hdfashion.tv/pervaya-v-mire-kamera-na-40-megapikselei-kompaniya-huawei-prezentovala-smartfon-p20-pro – [23.03.2020].

126. Поняття колірної моделі. Колірна модель HSB. [Електронний ресурс] – Режим доступу до ресурсу: http://um.co.ua/8/8-2/8-205347.html – [23.03.2020].

127. Практическое применение стеганографии [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/253045/. – [23.03.2020].

128. Статистический стегоанализ и противодействие [Електронний ресурс] – Режим доступу до ресурсу: https://tech.wikireading.ru/13239 – [23.03.2020].

129. Тестирование MPEG Layer 3 (MP3) кодеров. [Електронний ресурс] / Режим доступу: https://www.ixbt.com/multimedia/mp3-codec-tst1.html – [16.03.2020].

130. Что такое видеокодеки и аудиокодеки [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://movavi.io/ru/codec/ – [23.03.2020].

131. Що таке пікселі – поняття, визначення простими словами. [Електронний ресурс] – Режим доступу до ресурсу: https://termin.in.ua/piksel-pixel/ – [23.03.2020].

132. A genetic-algorithm-based approach for audio steganography / [Z. Mazdak, A. Manaf, R. Ahmad та ін.]. – 2009.

133. A review of audio based steganography and digital watermarking / M.Mat Kiah, B. Bilal, A. A. Zaidan, M. A. Sameer. // International Journal of Physical Sciences. – 2011. – №6.

134. An overview of steganography techniques applied to the protection of biometric data / M.Douglas, K. Bailey, M. Leeney, K. Curran. // Multimedia Tools and Applications. – 2018. – №77. – P. 17333–17373.

135. Announcing the Advanced Encryption Standard (AES) [Електронний ресурс] / National Institute of Standards and Technology, 2001 / Режим доступу: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf – [25.03.2020].

136. Combination of Steganography and Cryptography: A short Survey / [M.

Taha, M. Rahim, S. Lafta та ін.]. // Information Technology and Communication. – 2019. – №518. – P. 1–13.

137. Curve25519: new Diffie-Hellman speed records / D. Bernstein, M.Yung, Y. Dodis. // Springer. – 2006. – №3958. – P. 207–228.

138. Digital Image Steganography: Survey and Analysis of Current Methods / A.Cheddad, J. Condell, K. Curran, P. Mc Kevitt. // Signal Processing. – 2010. – №90. – P. 727–752.

139. E-Banking Security using Cryptography, Steganography and Data Mining / N.Devadiga, H. Kothari, H. Jain, S. Sankhe. // International Journal of Computer Applications. – 2017. – №164. – P. 26–30.

140. Free Lossless Audio Codec. [Електронний ресурс] / Режим доступу: https://xiph.org/flac/ – [16.03.2020].

141. Head First. Паттерны проектирования. Обновленное юбилейное издание / Э.Фримен, Э. Робсон, К. Сьерра, Б. Бейтс., 2018.

142. Hide4PGP. [Електронний ресурс] / Режим доступу: http://www.heinz-repp.onlinehome.de/Hide4PGP.htm – [15.03.2020].

143. JSteg. [Електронний ресурс] / Режим доступу: https://github.com/lukechampine/jsteg – [15.03.2020].

144. MP3 format. [Електронний ресурс] / Режим доступу: https://techterms.com/definition/mp3 – [16.03.2020].

145. Outguess. [Електронний ресурс] / Режим доступу: https://www.download3k.ru/Bezopasnost/SHifrovanie/Download-Outguess-Rebirth-steganography.html – [15.03.2020].

146. S-Tools. [Електронний ресурс] / Режим доступу:

https://www.chaynikam.info/stools.html – [15.03.2020].

147. Steganographic Protection Method Based on Huffman Tree / [Y. Radchenko, I. Dychka, Y. Sulema та ін.]. // Springer. – 2019. – № 902. – P. 283–292.

148. Steganos. [Електронний ресурс] / Режим доступу: https://www.steganos.com/en/help/18/steganos-safe/hidden-safe – [15.03.2020].

149. The Twofish Encryption Algorithm: A 128-Bit Block Cipher / [B. Schneier, J. Kelsey, D. Whiting та ін.]., 1999. – 186 p. – P. 4.

150. Understanding Color Models: A Review / N. A.Ibraheem, M. M. Hasan, R. Z. Khan, P. K. Mishra. // ARPN Journal of Science and Technology. – 2012. – №2. – P. 265–275.

151. WAVE PCM soundfile format. [Електронний ресурс] / Режим доступу: http://soundfile.sapp.org/doc/WaveFormat/ – [16.03.2020].

ДОДАТОК А. Програмна реалізація методу на основі схеми відповідності

```
#include "ImageBitsDifferenceEmbeder.h"
#include <cmath>
#include <QFile>
#include <bitset>

ImageBitsDifferenceEmbeder::ImageBitsDifferenceEmbeder(std::vector<bool>
secretInBits
                                                       , QSize secretImageSize
                                                       , const QString&
conteinerPath
                                                       , const QString&
conteinerWithDataPath
                                                       , std::vector<uint8_t>*
keyBytes)
    : ImageLSBEmbeder(secretInBits, secretImageSize, conteinerPath,
conteinerWithDataPath, keyBytes)
{
    parseKey();
}

void ImageBitsDifferenceEmbeder::parseKey()
{
    if(!_keyBytes || _keyBytes->empty())
    {
        Logger::instance()->log("ImageBitsDifferenceEmbeder::parseKey: Invalid
_keyBytes.");
        _logicFunctionPtr = &Operations::secondFunction;
        _schema.fillDefaultSchema();

        return;
    }

    uint8_t logicFuncNunber = _keyBytes->at(0);
    _logicFunctionPtr = Operations::getLogicFuncByNumber(logicFuncNunber);

    _schema = Schema(std::vector<uint8_t>(_keyBytes->begin() + 2, _keyBytes-
>end()));
}

bool ImageBitsDifferenceEmbeder::embedInfo()
{
    if(_schema.isEmpty())
    {
        _schema.fillDefaultSchema();
    }

    return ImageLSBEmbeder::embedInfo();
}

bool ImageBitsDifferenceEmbeder::embedInfoInPixel(uint8_t& red, uint8_t& green,
uint8_t& blue)
{
    const std::vector<std::pair<uint8_t, uint8_t> >* bitIndexes =
_schema.getSchemaByBitsCountNumbet(_bitsChangedCount);
```

```
    if(!bitIndexes)
    {
        qWarning() << "LSB Image: Invalid bits count changed.";

        return true;
    }

    uint allrgb = (red << 16) ^ (green << 8) ^ blue;
    std::bitset<24> bitset(allrgb);

    if(_currentSecretBitIndex + 3 * _bitsChangedCount < _secretDataInBitsSize)
    {
        for(int i = 0; i < bitIndexes->size(); ++i)
        {
            bitset.set(bitIndexes->at(i).first,
_logicFunctionPtr(bitset[bitIndexes->at(i).second],

_secretDataInBits[_currentSecretBitIndex++]));
        }

        allrgb = bitset.to_ulong();

        red = allrgb >> 16;
        green = allrgb >> 8;
        blue = allrgb;

        return false;
    }

    for(int i = 0; i < bitIndexes->size(); ++i)
    {
        bitset.set(bitIndexes->at(i).first, _logicFunctionPtr(bitset[bitIndexes-
>at(i).second],

_secretDataInBits[_currentSecretBitIndex++]));
        red = bitset.to_ulong() >> 16;
        green = bitset.to_ulong() >> 8;
        blue = bitset.to_ulong();
        if(_currentSecretBitIndex >= _secretDataInBitsSize)
        {
            return true;
        }
    }

    return false;
}
```

ДОДАТОК Б. Програмна реалізація методу на основі дерева Хаффмана

```
#include "HuffmanEmbeder.h"
#include <bitset>

HuffmanEmbeder::HuffmanEmbeder(std::vector<bool> secretInBits
                               , QSize secretImageSize
                               , const QString& conteinerPath
                               , const QString& conteinerWithDataPath
                               , std::vector<uint8_t>* keyBytes)
    : ImageBitsDifferenceEmbeder(secretInBits, secretImageSize, conteinerPath,
conteinerWithDataPath,
                                 keyBytes)
{
    _bitsChangedCount = 1;
}

bool HuffmanEmbeder::code()
{
    if(_conteinerImage.isNull())
    {
        qWarning() << "Empty conteiner image.";

        return false;
    }

    if(!canEmbedSecretData())
    {
        qWarning() << "Secret data is too big for this conteiner.";

        return false;
    }

    QTime time;
    time.start();

    embedInfo();

    if(_currentSecretBitIndex < _secretDataInBits.size())
    {
        size_t recomended_size = _secretDataInBits.size() / 3;
        size_t add_pixels_to_write_secret_resolution =
static_cast<size_t>(recomended_size * 0.001);

        qDebug() << "Information can not be embedded! Container is too small!";
        qDebug()  << "To embed this secret image, you must use container which
has not less than" <<
                  recomended_size +
                  add_pixels_to_write_secret_resolution + 2 << "pixels";
        qDebug()  << "Min container resolution:" << (int)(sqrt(recomended_size +

add_pixels_to_write_secret_resolution) + 2) << "x" <<
                  (int)(sqrt(recomended_size +
add_pixels_to_write_secret_resolution) + 2);

        return false;
    }
```

```
    qDebug() << "Information was embedded. BitsDifferenceEmbedder code time
(before save): " <<
             time.elapsed();
    save();
    qDebug() << "Information was embedded. After saving: " << time.elapsed();

    return true;
}

bool HuffmanEmbeder::embedInfo()
{
    _currentSecretBitIndex = 0;

    uint bytesCountToEmbedSecretImage = _secretDataInBitsSize / 8;
    if(_secretDataInBitsSize % 8 != 0)
    {
        bytesCountToEmbedSecretImage += 1;
    }

    uint startPosition = writeSecretDataSize();
    uint freePixelInConteiner = _conteinerImage.width() *
_conteinerImage.height() - 2 *
                                (startPosition / 3) - _MaxHolePixelCount;

    uint maxBytesCountCouldBeEmbed = (freePixelInConteiner * 3) / 8;
    if(bytesCountToEmbedSecretImage >= maxBytesCountCouldBeEmbed)
    {
        qWarning() << "That image could not be embed in conteiner.";

        return false;
    }

    if(_schema.isEmpty())
    {
        _schema.fillDefaultSchema();
    }

    uint pixelCountHole = maxBytesCountCouldBeEmbed /
bytesCountToEmbedSecretImage - 1;
    writeHolePixelCount(startPosition, pixelCountHole);
    qDebug() << "Pixel hole count:" << pixelCountHole;

    uint pixelEmbed = 0;
    if(pixelCountHole == 0)
    {
        for(uint i = startPosition + 3 * _MaxHolePixelCount; i <
_rgbValuesOfConteiner.size(); i += 3)
        {
            if(embedInfoInPixel(_rgbValuesOfConteiner[i],
_rgbValuesOfConteiner[i + 1],
                                _rgbValuesOfConteiner[i + 2]))
            {
                return true;
            }
        }
    }
    else
    {
        for(uint i = startPosition + 3 * _MaxHolePixelCount; i <
_rgbValuesOfConteiner.size(); i += 3)
```

```
        {
            if(pixelEmbed == 0) //встраиваем полезную инфу
            {
                if(embedInfoInPixel(_rgbValuesOfConteiner[i],
_rgbValuesOfConteiner[i + 1],
                                    _rgbValuesOfConteiner[i + 2]))
                {
                    return true;
                }
            }
            else if(pixelEmbed ==
                    pixelCountHole) //пропускаем и ничего не встраиваем в теку-
щий пиксель
            {
                pixelEmbed = 0;

                continue;
            }

            ++pixelEmbed;
        }
    }

    return false;
}

bool HuffmanEmbeder::embedInfoInPixel(uint8_t& red, uint8_t& green, uint8_t&
blue)
{
    uint allrgb = (red << 16) ^ (green << 8) ^ blue;
    std::bitset<24> bitset(allrgb);

    if(_currentSecretBitIndex + 3 < _secretDataInBitsSize)
    {
        bool lastRedBitIsTrue = red % 2 == 0;
        bool newRedBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(0).second],

_secretDataInBits[_currentSecretBitIndex++]);
        if(lastRedBitIsTrue == newRedBit)
        {
            if(_randomGenerator.bounded(2) == 0)
            {
                if(red > 0)
                {
                    red -= 1;
                }
                else
                {
                    red += 1;
                }
            }
            else
            {
                if(red < 255)
                {
                    red += 1;
                }
                else
                {
                    red -= 1;
```

```
                }
            }
        }

        //green
        std::bitset<8> greenBitSet(green);
        bool lastGreenBitIsTrue = green % 2 == 0;
        bool newGreenBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(1).second],

_secretDataInBits[_currentSecretBitIndex++]);
        if(lastGreenBitIsTrue == newGreenBit)
        {
            if(_randomGenerator.bounded(2) == 0)
            {
                if(green > 0)
                {
                    green -= 1;
                    std::bitset<8> newGreenBitSet(green);
                    if(newGreenBitSet[6] != greenBitSet[6])
                    {
                        green += 2;
                    }
                }
                else
                {
                    green += 1;
                    std::bitset<8> newGreenBitSet(green);
                    if(newGreenBitSet[6] != greenBitSet[6])
                    {
                        green -= 2;
                    }
                }
            }
            else
            {
                if(green < 255)
                {
                    green += 1;
                    std::bitset<8> newGreenBitSet(green);
                    if(newGreenBitSet[6] != greenBitSet[6])
                    {
                        green -= 2;
                    }
                }
                else
                {
                    green -= 1;
                    std::bitset<8> newGreenBitSet(green);
                    if(newGreenBitSet[6] != greenBitSet[6])
                    {
                        green += 2;
                    }
                }
            }
        }

        //blue
        std::bitset<8> blueBitSet(blue);
        bool lastBlueBitIsTrue = blue % 2 == 0;
```

```
        bool newBlueBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(2).second],

_secretDataInBits[_currentSecretBitIndex++]);
        if(lastBlueBitIsTrue == newBlueBit)
        {
            if(_randomGenerator.bounded(2) == 0)
            {
                if(blue > 0)
                {
                    blue -= 1;
                    std::bitset<8> newBlueBitSet(blue);
                    if(newBlueBitSet[5] != blueBitSet[5] || newBlueBitSet[7] !=
blueBitSet[7])
                    {
                        blue += 2;
                    }
                }
                else
                {
                    blue += 1;
                    std::bitset<8> newBlueBitSet(blue);
                    if(newBlueBitSet[5] != blueBitSet[5] || newBlueBitSet[7] !=
blueBitSet[7])
                    {
                        blue -= 2;
                    }
                }
            }
            else
            {
                if(blue < 255)
                {
                    blue += 1;
                    std::bitset<8> newBlueBitSet(blue);
                    if(newBlueBitSet[5] != blueBitSet[5] || newBlueBitSet[7] !=
blueBitSet[7])
                    {
                        blue -= 2;
                    }
                }
                else
                {
                    blue -= 1;
                    std::bitset<8> newBlueBitSet(blue);
                    if(newBlueBitSet[5] != blueBitSet[5] || newBlueBitSet[7] !=
blueBitSet[7])
                    {
                        blue += 2;
                    }
                }
            }
        }

        return false;
    }

    bool redBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(0).second],

_secretDataInBits[_currentSecretBitIndex++]);
```

```
    bitset.set(16, redBit);
    red = bitset.to_ulong() >> 16;
    if(_currentSecretBitIndex >= _secretDataInBitsSize)
    {
        return true;
    }

    bool greenBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(1).second],

_secretDataInBits[_currentSecretBitIndex++]);
    bitset.set(8, greenBit);
    green = bitset.to_ulong() >> 8;
    if(_currentSecretBitIndex >= _secretDataInBitsSize)
    {
        return true;
    }

    bool blueBit =
_logicFunctionPtr(bitset[_schema.oneBitCorrespondenceScheme.at(2).second],

_secretDataInBits[_currentSecretBitIndex++]);
    bitset.set(0, blueBit);
    blue = bitset.to_ulong();
    if(_currentSecretBitIndex >= _secretDataInBitsSize)
    {
        return true;
    }

    return false;
}

bool HuffmanEmbeder::canEmbedSecretData() const
{
    uint maxSecretImageBitsCount = _conteinerImage.width() *
_conteinerImage.height() * 3;

    return _secretDataInBitsSize < maxSecretImageBitsCount;
}

void HuffmanEmbeder::writeHolePixelCount(uint
startPositionAfterWrittenWidthInByte,
                                         uint holePixelCount)
{
    std::bitset<24> holeBitSet(holePixelCount);

    for(size_t i = startPositionAfterWrittenWidthInByte;
            i < startPositionAfterWrittenWidthInByte + _MaxHolePixelCount * 3; i
+= 3)
    {
        if((_rgbValuesOfConteiner[i] & 1) != (holeBitSet[i -
startPositionAfterWrittenWidthInByte]))
        {
            _rgbValuesOfConteiner[i] ^= 1;
        }
        if((_rgbValuesOfConteiner[i + 1] & 1) != (holeBitSet[i -
startPositionAfterWrittenWidthInByte + 1]))
        {
            _rgbValuesOfConteiner[i + 1] ^= 1;
        }
```

```
        if((_rgbValuesOfConteiner[i + 2] & 1) != (holeBitSet[i -
startPositionAfterWrittenWidthInByte + 2]))
        {
            _rgbValuesOfConteiner[i + 2] ^= 1;
        }
    }
}

uint8_t HuffmanEmbeder::getCapacityOfSize()
{
    uint coverAllPixelsCount = _conteinerImage.width() *
_conteinerImage.height();
    uint maxSecretImagePixelCount = coverAllPixelsCount / 8;

    uint8_t capacity = static_cast<uint8_t>(floor(log(maxSecretImagePixelCount)
/ log(2)) + 1);

    return capacity;
}
```

ДОДАТОК В. Програмна реалізація методу на основу псевдовипадкового вбудовування

```
#include "RandomWithKeyEmbeder.h"
#include "Logger.h"
#include "KeysHolder.h"

#include <QRandomGenerator>
#include <QLinkedList>
#include <algorithm>

RandomWithKeyEmbeder::RandomWithKeyEmbeder(const QString& conteinerPath,
                                           const QString& secretPath,
                                           const QString& conteinerWithDataPath,
                                           std::vector<uint8_t>* keyBytes):
    BaseCodeMethod(conteinerPath, secretPath, conteinerWithDataPath, keyBytes),
    _conteiner(nullptr),
    _secret(nullptr),
    _conteinerWithData(nullptr),
    _key(nullptr)
{
    parseKey();
}

RandomWithKeyEmbeder::~RandomWithKeyEmbeder()
{
    delete _conteiner;
    delete _key;
    delete _conteinerWithData;
    delete _secret;
}

void RandomWithKeyEmbeder::initialize()
{
    _conteiner = new WavFileHolder(_conteinerPath);
    _secret = new FileHolder(_secretPath, true);
    _key = new FileHolder(_keyPath, true);

    if(QFile::exists(_conteinerWithDataPath))
    {
        QFile::remove(_conteinerWithDataPath);
    }

    if(_conteiner->getFile().copy(_conteinerWithDataPath))
    {
        _conteinerWithData = new WavFileHolder(_conteinerWithDataPath);
    }
    else
    {
        Logger::instance()->log("Can not copy file.");
    }
}

bool RandomWithKeyEmbeder::code()
{
    Logger::instance()->dlog(QString::number(_conteiner->allSamplesCount()) +
```

```
        "_secret.arraySize() =" + QString::number(_secret->byteArraySize()));

    QmlKeysHolder::instance().setSecretSize(_secret->byteArraySize());
    emit QmlKeysHolder::instance().secretSizeChanged();

    const quint32 ConteinerSize = _conteiner->allSamplesCount(); // samples
count in cover
    quint32 secretSize = _secret->byteArraySize() * 4; //размер учатсками по 2
бита
    if(secretSize >= ConteinerSize)
    {
        Logger::instance()->log("Cover is too small to embed choosed secret
data.");

        return false;
    }

    QTime time;
    time.start();

    _usedPosition.reserve(secretSize);
    for(quint32 i = 0; i < secretSize; ++i)
    {
        _usedPosition.emplace_back(i);
    }


    std::vector<quint32> usedPosition(secretSize);
    for(quint32 i = 0; i < secretSize; ++i)
    {
        usedPosition.emplace_back(i);
    }

    bool res = _usedPosition == usedPosition;

    std::shuffle(_usedPosition.begin(), _usedPosition.end(),
std::mt19937(_seed1));

    QRandomGenerator gen(_seed2 != 0 ? _seed2 : _seed1 );
    QRandomGenerator noiseGen(_seed1);

    quint32 j = 0; //counter

    for(quint32 i = 0; i < ConteinerSize; ++i)
    {
        if(gen.bounded(ConteinerSize) / (qreal)ConteinerSize <
((qreal)secretSize - j) /
                (ConteinerSize - i))
        {
            // secrets blocks embedded
            qint8 val = _conteinerWithData->getDataAt(i) & -4; //1111 1100
            quint8 secret2Bits = _secret->get2BitsByIndex(_usedPosition[j]);

            if((secret2Bits & 2) ^ (_key->readBit() << 1))
            {
                val |= 2;
            }

            if((secret2Bits & 1) ^ _key->readBit())
            {
                val |= 1;
```

```
            }

            _conteinerWithData->setDataAt(i, val);
            ++j;
        }
        else
        {
            //noise blocks embedded
            _conteinerWithData->setDataAt(i, _conteinerWithData->getDataAt(i) |
noiseGen.bounded(4));
        }
    }
    Logger::instance()->dlog("Code time with embedded noise without
parallelizing" + QString::number(
                                time.elapsed()));

    _conteinerWithData->writeFile();
    _conteinerWithData->closeFile();

    return true;
}

void RandomWithKeyEmbeder::parseKey()
{
    if(!_keyBytes || _keyBytes->empty())
    {
        Logger::instance()->log("RandomWithKeyEmbeder::parseKey: Invalid
_keyBytes.");

        return;
    }

    _seed1 = _keyBytes->at(3) << 24 ^ _keyBytes->at(2) << 16 ^ _keyBytes->at(
                1) << 8 ^ _keyBytes->at(0);

    _seed2 = _keyBytes->at(8) << 24 ^ _keyBytes->at(7) << 16 ^ _keyBytes->at(
                6) << 8 ^ _keyBytes->at(5);

    //skip secretSize reading

    std::string str(_keyBytes->begin() + 15, _keyBytes->end());
    _keyPath = QString::fromStdString(str);
}
```

ДОДАТОК Г. Користувацький інтерфейс

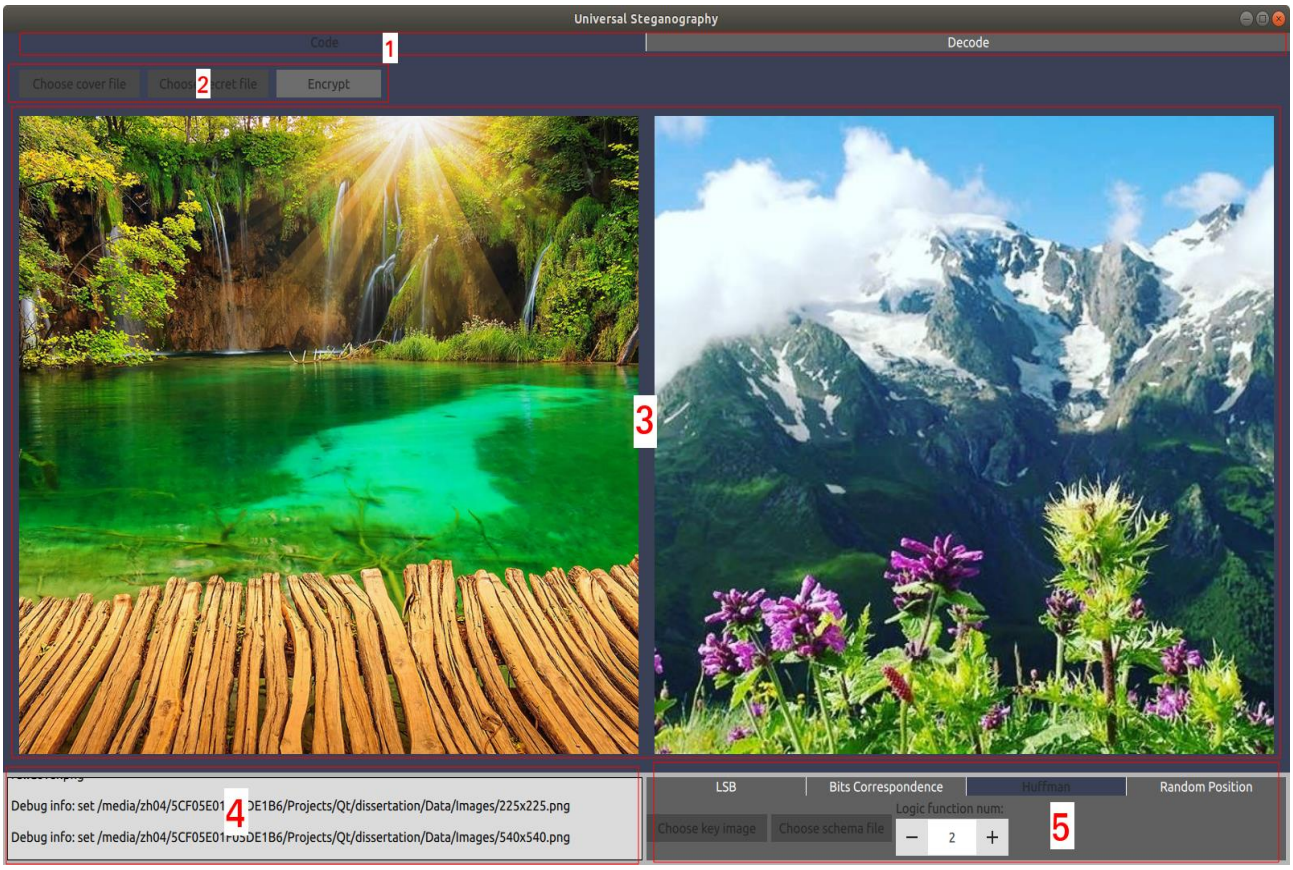

Рис. Г.1 Графічний інтерфейс користувача

ДОДАТОК Ґ. Ілюстративний матеріал

Рис. Ґ.1 Контейнер cover.png з роздільністю 1024x1024

Рис. Ґ.2 Конфіденційні графічні дані secret.png з роздільністю 225x225

Рис. Ґ.3 Порожній та заповнений контейнери

Рис. Ґ.4 Зображення-ключ

Рис. Ґ.5 Зображення-контейнер

Рис. Ґ.6 Конфіденційні графічні дані

Рис. Ґ.7 Конфіденційне зображення (зліва) та декодоване зображення (справа)

Рис. Ґ.8 Конфіденційне зображення

Рис. Ґ.9 Зображення-контейнер

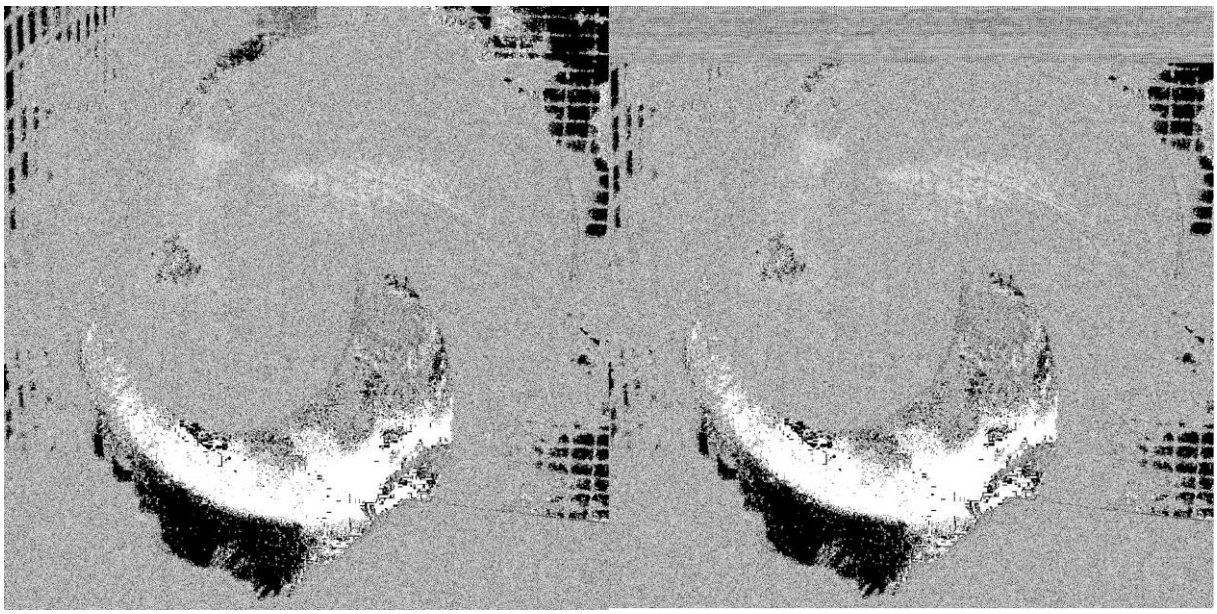

Рис. Ґ.10 Молодший біт компоненти R порожнього (зліва) та заповненого (справа) контейнерів

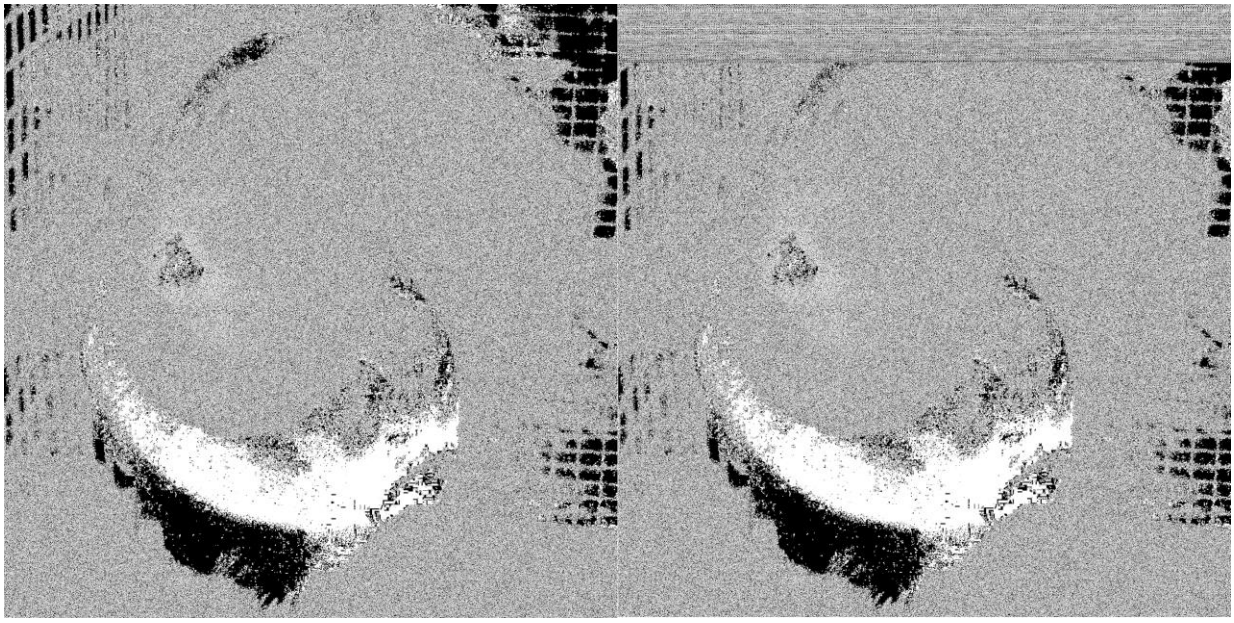

Рис. Ґ.11 Молодший біт компоненти G порожнього (зліва) та заповненого (справа) контейнерів

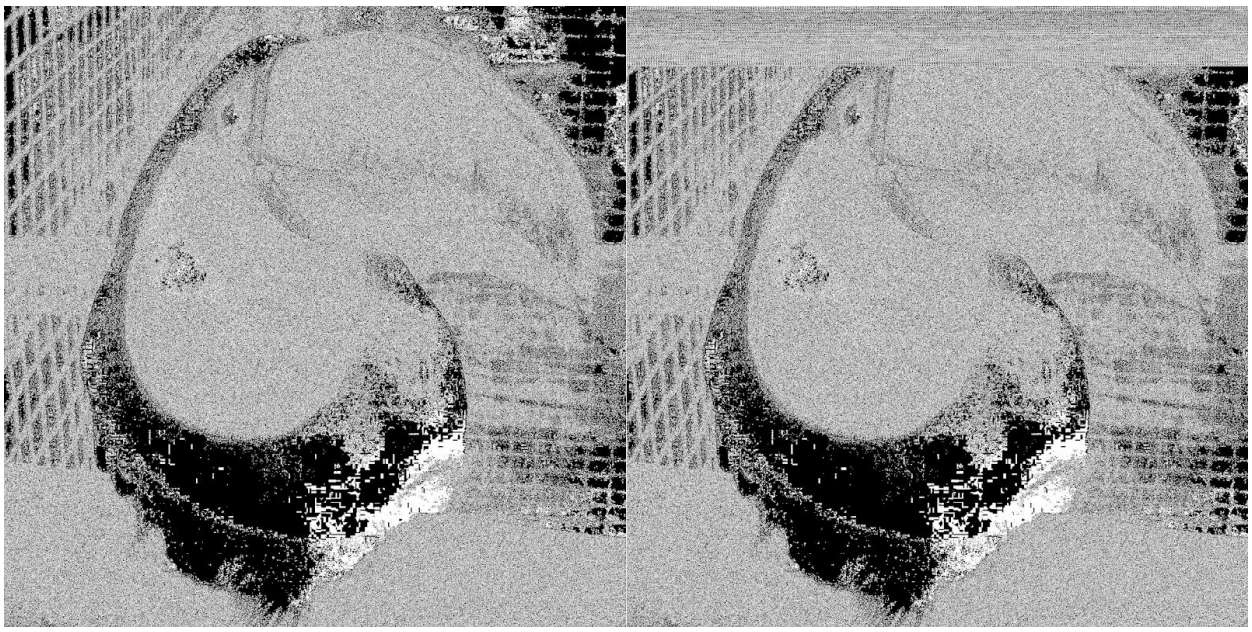

Рис. Ґ.12 Молодший біт компоненти B порожнього (зліва) та заповненого (справа) контейнерів

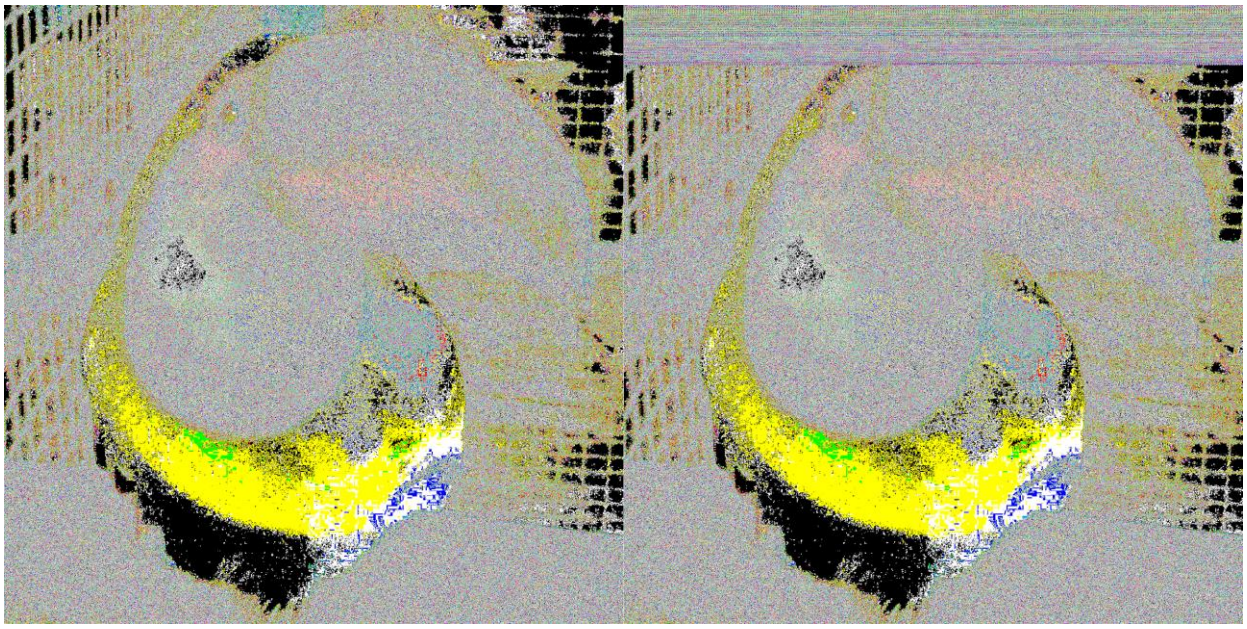

Рис. Ґ.13 Молодший біт компонент R, G та B порожнього (зліва) та заповненого (справа) контейнерів

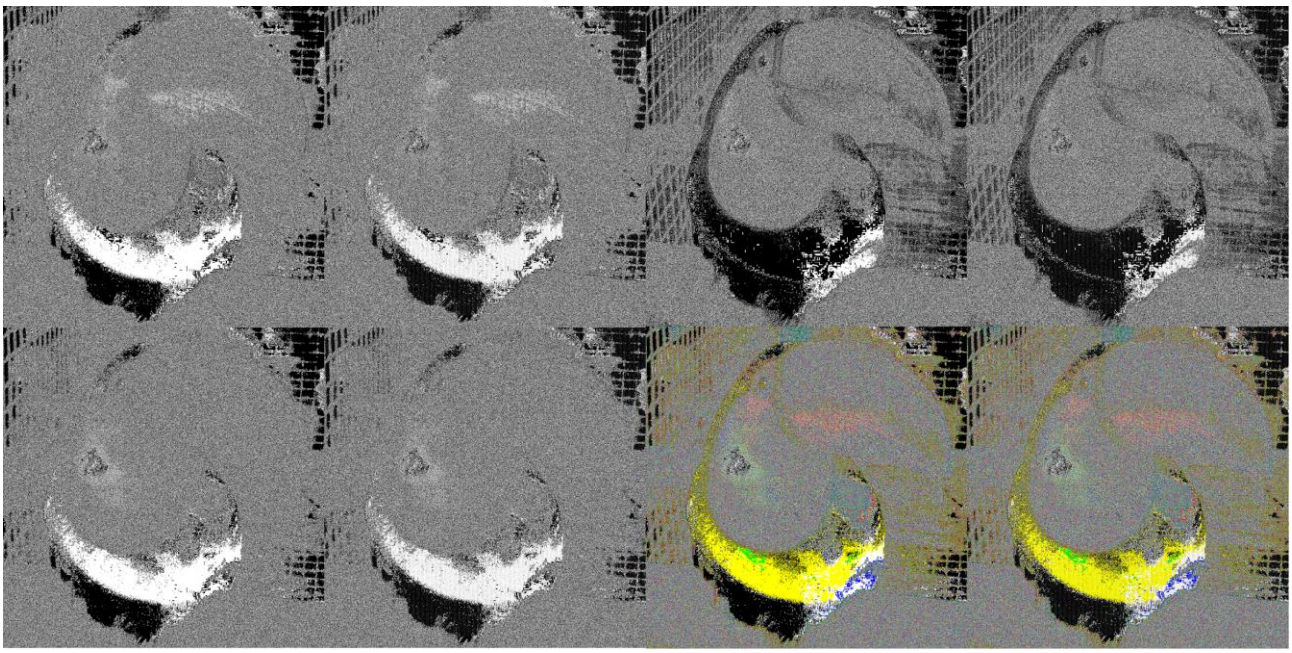

Рис. Ґ.14 Результат візуальної атаки LSB на порожній (зліва) та заповнений (справа) контейнер методом на основі дерева Хаффмана