Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
« » червня 2021р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

дипломної роботи
бакалавра
(назва освітнього рівня)

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітня програма Кібербезпека
(назва освітньої програми)

на тему: «Сучасні методи захисту інформації з використанням аудіостеганографії»

Виконавець: студент IV курсу, групи КБ-41

_____ **Гедеон Максим Ігорович**
(підпис) (прізвище ім'я по-батькові)

	Прізвище, ініціали	**Підпис**
Керівник	Зюбіна Р. В.	

Нормоконтроль	Даков С. Ю.	

Київ 2021

Міністерство освіти і науки України

«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій

Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
завідувач кафедри кібербезпеки
та захисту інформації
_____ Н.В. Лукова-Чуйко
«10» жовтня 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студенту **КБ-41** (група) **Гедеона Максима Ігоровича** (прізвище ім'я по-батькові)

Тема дипломної роботи Сучасні методи захисту інформації з використанням аудіостеганографії

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема дипломної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №2 від 08.10.2020 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Стеганографічні методи захисту інформації

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Необхідно провести аналіз методів стеганографічного захисту інформації, вдосконалити алгоритм LSB, розробити програмне забезпечення, яке використовує вдосконалений алгоритм.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблено програмний продукт, який дозволяє

приховувати факт наявності конфіденційної інформації.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання:12 жовтня 2020 року

Завдання видав		Р. В. Зюбіна
	(підпис)	(ініціали, прізвище)
Завдання прийняла до виконання		М. І. Гедеон
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	25.01.2021 – 22.01.2020	*виконано*
2	Аналіз літератури	29.01.2020 – 11.02.2020	*виконано*
3	Обґрунтування вибору рішення	12.02.2020 – 15.02.2020	*виконано*
4	Аналіз актуальності стеганографічних методів захисту інформації	16.02.2020 – 04.03.2020	*виконано*
5	Аналіз ризиків та використання стеганографічних методів захисту інформації	05.03.2020 – 21.03.2020	*виконано*
6	Дослідження вразливостей та методів стеганографічного захисту інформації	22.03.2020 – 08.04.2020	*виконано*
7	Модифікація алгоритму і реалізація програмного продукту	09.04.2020 – 10.05.2020	*виконано*
8	Оформлення пояснювальної записки	11.05.2020 – 27.05.2020	*виконано*
9	Підготовка до захисту дипломної роботи	28.05.2020 – 08.06.2021	*виконано*

Завдання видав		Р. В. Зюбіна
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		М. І. Гедеон
	(підпис)	(ініціали, прізвище)

Термін подання дипломної роботи до ЕК 08 червня 2021 року

РЕФЕРАТ

Дипломна робота складається зі вступу, чотирьох розділів, загальних висновків, списку використаних джерел, додатків, має 68 сторінок основного тексту, 18 рис., 1 табл., 1 додаток, 75 джерел.

Актуальність. Комп'ютерна стеганографія активно розвивається, використовуються відомі та нові методи стеганографії, засновані на різних галузях науки. Стеганографічні системи виходять на новий етап розвитку.Сьогодні більшість з них враховують характеристики та властивості стеганографічних контейнерів, що зберігають дані при приховуванні інформації. Комп'ютерна стеганографія застосовується в багатьох сферах людської діяльності:

- таємно передавати конфіденційну інформацію у мультимедійних файлах;
- захист авторських прав на аудіо- та відеоматеріали в електронному вигляді;
- створення секретних файлів;
- досягнення мережевих систем моніторингу та управління;
- маскування програмного забезпечення.

При передачі конфіденційних повідомлень, вбудованих в аудіоконтейнери, для безпеки цієї передачі необхідно мінімізувати спотворення контейнера. Тому актуальним та важливим є розробка ефективних цифрових методів захисту інформації, включаючи комп'ютерну стеганографію та методи стеганалізу, розподілених у різних контейнерах.

Мета роботи: аналіз методів і алгоритмів цифрової стеганографії та розробка програмного продукту для практичного застосування стеганографії в аудіоконтейнерах

Об'єкт дослідження: процес захисту інформації, прихованої в аудіоконтейнер

Предмет дослідження: методи і алгоритми цифрової стеганографії

Практична цінність: розроблено програмний продукт, який дозволяє приховувати факт наявності конфіденційної інформації

стеганографія, захист інформації, найменш значущий біт, стеганоконтейнер.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві багато послуг надаються через комп’ютерні мережі та інформаційні технології. Інформація, що надається у цифровій формі, повинна бути надійно захищена від численних загроз: несанкціонованого доступу та використання, саботажу, підробки, витоку, порушення ліцензійних угод, відмови від авторського права тощо. Інформаційна безпека надзвичайно важлива як у комерційній, так і в державній сферах. Загрози національним інтересам та безпеці України в галузі інформатизвції включають: комп'ютерний тероризм та злочинність; державна таємна або конфіденційна інформація; маніпулювати обізнаністю громадськості, особливо шляхом поширення неправдивої інформації.

Тому розробка ефективних методів захисту цифрової інформації, особливо проблем комп’ютерної стеганографії та методів стегоаналізу, має велике значення для країни та суспільства. Найбільшим розвитком в Україні та світі є методологія для забезпечення конфіденційності та вірогідності інформації, такої як криптографія. Однак на основі стеганографії можна створити альтернативні засоби захисту, а в деяких додатках можна використовувати модулі криптографічної стеганографії. Крім того, є деякі важливі завдання інформаційної безпеки, які неможливо вирішити лише криптографією, особливо коли існування конфіденційної інформації потрібно приховувати. Метод стеганографії, по суті, забезпечує вищий рівень захисту, оскільки захищені дані та факт їх передачі залишаються поза увагою сторонніх осіб.

Завдяки глибокій інтеграції засобів шифрування в інформаційну систему, сучасні комп’ютерні технології обробки даних значно покращили рівень захисту інформації. Як відомо, на відміну від захисту паролем інформації, приховане програмне забезпечення в основному намагається приховати той факт, що існує конфіденційна інформація. Технологія стеганографії, яка приховує інформацію в цифровому потоці сигналів і реалізується на базі комп'ютерного обладнання та

програмного забезпечення, є предметом цифрової стеганографії. З популярністю персональних комп'ютерів та Інтернету, можливість таємно передавати інформацію привертає увагу багатьох людей, тому актуальність досліджень стенографічних способів зростає. Переважна більшість теоретичних та практичних досліджень у галузі стеганографії присвячена розробці нових та вдосконаленню чинних методів приховування даних.

Зростання сучасних комунікаційних можливостей вимагає розробки спеціальних методів безпечного зберігання та передачі інформації. Враховуючи збільшення обсягу даних, що передаються через локальні та глобальні мережі, безпека мережі стає все більш важливою. Щоб захистити інформацію від несанкціонованого доступу та використання, необхідно забезпечити конфіденційність та цілісність даних. Інформаційну безпеку може забезпечити криптографія, стеганографія або як криптографія, так і стеганографія. При використанні криптографії інформація буде модифікована та перетворена. Результатом перетворення є те, що вміст повідомлення приховано. Своєю чергою, стеганографія приховує факт передачі або зберігання інформації. Це досягається введенням захищеної інформації в різні мультимедійні об'єкти (контейнери), які не втрачають своїх атрибутів користувача. Щодо комп'ютерних технологій, існує окрема галузь стеганографії - комп'ютерна стеганографія. Тут як контейнери використовуються файли в різних форматах, мережеві пакети тощо. Наприклад, ви можете ввести інформацію у звуковий сигнал, а потім відтворити її майже повністю (з тією ж якістю), що і вхідний сигнал, не усвідомлюючи цього. З іншого боку, приховування даних можна використовувати в некомерційних сферах, щоб приховати інформацію, яку хтось хоче зберегти в таємниці. Тому необхідно розробити ефективний метод виявлення прихованих вкладень в мультимедійних об'єктах, що передаються в комп'ютерних мережах.Комп'ютерні технології надають новий поштовх для розвитку стеганографії, а комп'ютерна стеганографія забезпечує споживачам невидиму якість, вбудовування даних у контейнер - файли, що містять цифровий звук або зображення.

Стеганографія має багато застосувань на практиці, тому інтерес до цієї галузі все ще високий. Прикладами таких програм є: захист інформації від несанкціонованого доступу; протистояння системам моніторингу та управління мережевими ресурсами; від захисного програмного забезпечення незареєстрованих користувачів; певні види захисту авторських прав та права інтелектуальної власності. Однак сучасне покоління технології стеганографії аудіоконтейнерів потребує подальшого вдосконалення. Ці вдосконалення включають ефективні методи для поліпшення стеганографічної стабільності контейнера для криптоаналізу.

Актуальність теми полягає у можливості застосування кіберстеганографії для приховування власних даних у будь-якій сфері життя людини. Через активний розвиток мережі Інтернет і збільшення кількості інформації, яка щомиті передається через незахищені канали зв’язку є актуальною проблема приховування важливих даних від сторонніх очей зловмисників.

Основною метою роботи є аналіз теперішніх методів кіберстеганографії, дослідження способів покращення наявних методів та створення покращень для чинних методів у галузі кіберстеганографії звукових файлів.

Об’єктом дослідження виступають стенографічні методи та алгоритми оптимізації методів стеганографічного аналізу.

Предметом дослідження виступають способи приховування даних за допомогою кіберстеганографії у музичних файлах.

РОЗДІЛ 1
АНАЛІЗ АКТУАЛЬНОСТІ, РИЗИКІВ ТА ВИКОРИСТАННЯ СТЕГАНОГРАФІЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ В ІНФОРМАЦІЙНИХ СИСТЕМАХ ТА МЕРЕЖАХ

1.1. Постановка задачі

Сьогодні в Інтернеті ви можете знайти безліч безкоштовних програм для стеганографії. Алгоритми, на яких базуються такі програми, виконують втручання в конфіденційну інформацію в так званих контейнерах (зображення, аудіо, відео). Використання таких програм дозволяє третім особам передавати будь-яку конфіденційну інформацію по відкритих каналах зв'язку, не помічаючи цього, одночасно передаючи неконфіденційну, відкриту (видиму) інформацію. Ця невидимість передачі даних може бути використана для вчинення злочинних намірів. Стеганаліз дозволяє запобігти несанкціонованій передачі інформації за допомогою стеганографічних методів. Основне завдання стеганалізу - визначити той факт, що в контейнері є прихована інформація.

Зазвичай це складний процес, і необхідно приховати передачу даних різними методами в різних форматах контейнерів. Як відомо, надійність прихованого повідомлення в контейнері швидко зменшується із збільшенням розміру самого повідомлення. Це означає, що якщо зловмисник перехопить переповнений контейнер, він легко відокремить повідомлення від контейнера.

Тому проблема полягає в тому, щоб вставити якомога більше інформації в контейнер, щоб зловмисник не зміг відновити вихідне повідомлення, навіть якщо контейнер перехоплений.

В результаті постановка задачі виглядає так – модифікувати алгоритм LSB та реалізувати програмний продукт, який буде мати максимальну швидкодію та надійність відносно до найменш можливих розмірів результуючого виконуваного файлу.

1.2. Дослідження предмету стеганографії

Розуміння суті поняття стеганографія приходить через пізнання поняття криптографія. Криптографія - це мистецтво захисту інформації шляхом перетворення її в незрозумілий формат, який називається зашифрованим текстом. Вам потрібно ключ, щоб розшифрувати цей непрочитаний формат. Криптографія спостерігалася на багатьох етапах еволюції людини.

Згадки про криптографію датуються ще 1900 р. до н. е.. У давньоєгипетських довідниках використовувались нестандартні ієрогліфи. З 500 р. до 600 р. до н. е. Єврейські книжники використовували зворотний алфавіт із простими кодами. З 50 до н. е. до 60 р. до н. е. Юлій Цезар використовував прості заміни звичайного алфавіту в публічних комунікаціях. Криптографія пережила історію варіативних змін. Сьогодні криптографія вийшла на новий рівень - квантову криптографію. Квантова криптографія поєднує фізику та криптографію, щоб створити нову криптографічну систему, яку неможливо зламати без відправника та одержувача. Завдяки довгій історії криптографії стеганографія також отримала чималий розвиток і насичену історію зародження та еволюції.

Стеганографія походить від грецького stegano (приховування чи таємниця) та - graphy (письмо чи живопис). Стеганографію можна визначити як приховування інформації, вставляючи її в інші, здавалося б, звичайні повідомлення, графіку чи звуки. Найдавніша технологія стеганографії була розроблена в Стародавній Греції близько 440 р. до н. Грецькі правителі використовували ранні версії стеганографії, які включали вишкрібання голови раба, вирізання повідомлення на шкірі голови, очікування росту волосся, а потім надсилання повідомлення за допомогою цього раба. Одержувач матиме підпорядкований заголовок, щоб відкрити повідомлення. Одержувач відповість тією ж стеганографією. Тоді ж була використана ще одна рання стеганографія. До цього методу залучений Демерст, який написав секретного листа спартанцям, щоб запобігти несподіваному вторгненню Ксеркса. Інформація була вирізана на дереві, а потім покрита шаром свіжого воску.

З часом стеганографія продовжує розвиватися на новому рівні. Стеганографія також широко використовувалася під час війни. Під час американської революції британська та американська армії використовували різні види невидимого чорнила. Невидимі чорнило виготовляються з досить доступних інградієнтів: молоко, оцет, сік та сеча. Вам потрібно світло або тепло, щоб розшифрувати ці приховані повідомлення. Під час Другої світової війни німці ввели мікродобавки. Міні-вкладення - це звичайні документи, фотографії та плани, зменшені та додані до звичайних документів. Нульовий пароль також використовується для передачі секретних повідомлень. Нульовий пароль - це незашифроване повідомлення з реальним повідомленням, вбудованим у поточний текст. Приховані повідомлення важко виявити в звичайних повідомленнях.

Стеганографія - це мистецтво приховувати секретну інформацію в документах, щоб лише відправник та одержувач могли знати про її існування. Конфіденційна інформація шифрується, тим самим приховуючи існування повідомлення.

Порівняно з повністю вивченими криптографічними системами, поняття та оцінка безпеки стеганографічних систем є складнішими та багатогранними. Особливо через відсутність теоретичних та практичних досліджень безпеки стеганосистем та різноманітних завдань захисту інформаційної стеганографії.

Можна сказати, що приховування є одним із методів підтримки інформаційної безпеки. Це спосіб спілкування, який приховує той факт, що у вас є секретна інформація. Стеганографічні методи ефективно використовується для захисту інформації від сторонніх користувачів та маскування програмного забезпечення. На конференції «Інформаційне приховування» (технологія приховування інформації) 1996 р. Було обговорено всі основні поняття стеганографії та використано єдиний термін. Будь-яка інформація, яка приховує конфіденційні дані, називається контейнером. Будь-який файл або потік даних можна використовувати як контейнер. Контейнер, який не містить секретних повідомлень, називається порожнім контейнером, а контейнер, що містить секретні повідомлення -

заповненим контейнером або стеганоконтейнером. Канал передачі стеганографічного контейнера називається стеганоканалом або таємним каналом.

Секретний ключ, необхідний для «вставки» інформації в контейнер, називається просто ключем або стеганографічним ключем. Залежно від кількості рівнів захисту в стеганографічній системі можна використовувати одну або кілька ключів.

1. Визначення 1.1. Стеганосистема - це група (Σ= (X, M, K, Y, E, D), порожній контейнер X, повідомлення M, ключ K, заповнений контейнер Y та перетворення E і D (алгоритми вставки та вилучення), які їх з'єднують.

2. Визначення 1.2. Контейнер (перевізник) - це не конфіденційна інформація, яка буде приховувати конфіденційні дані (повідомлення). У комп'ютерній стеганографії контейнером може бути будь-який файл або потік даних. Цифрові контейнери, як правило, являють собою зображення, аудіо- чи відеосигнали.

3. Визначення 1.3. Повідомлення є конфіденційним і повинно бути прихованим.

4. Визначення 1.4. Стеганоключ (стеганографічний ключ) - це елемент стеганографічної системи, який параметризує алгоритми вставки та вилучення та відомий лише відправнику та одержувачу стеганографічного контейнера. Стегновий ключ може визначити область розрізу, основа розподілу частот, правило поділу контейнерів для сегментів, потужність розповсюдження, індекси коефіцієнтів кореляції, точки квантування, кодові книги, вектори розширення тощо.

5. Визначення 1.5. Порожній контейнер - це контейнер, який не містить повідомлень, прихованих стеганографією.

6. Визначення 1.6. Контейнери, що містять приховану інформацію, називаються наповненими або стеганографічними контейнерами, або стеганографічними картами. Завдання системи стеганографії - помістити вхідне повідомлення в контейнер, щоб жоден сторонній користувач не помітив нічого, крім його основного змісту. Основний вміст контейнера не впливає на відправника чи одержувача, вони зацікавлені лише в успішній передачі повідомлення (стеганографії), що міститься в ньому. Важливо пам’ятати, що той факт, що

контейнер надсилається автором до одержувача, не повинен здаватися дивним, і контейнер не повинен суттєво відхилятися від специфікації.

Загальна модель системи стеганографії наведена на рисунку 1.1.

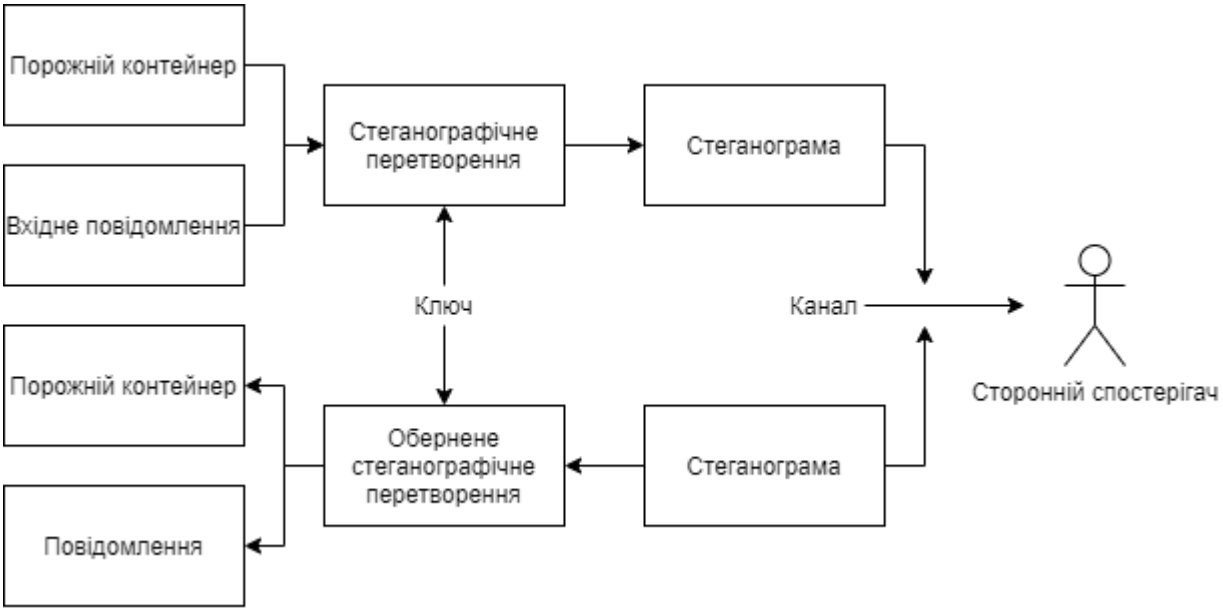

Рисунок 1.1 – Система стеганографії

Подібно до криптографії, якщо один і той же ключ використовується при вставці та видаленні повідомлень, система вважається симетричною, а якщо вони різні, вона називається асиметричною. Наука стеганографія поєднує досягнення теорії ймовірностей, статистики, криптографії й цифрової обробки сигналів та зображень та розпізнавання образів, теорії інформації, теорії дискретних ортогональних перетворень. Зверніть увагу, що існує метод, згідно з яким систему стеганографії можна розглядати як узагальнення шифрування в цілому. Стенографічні методи мають унікальні властивості, що робить їх незамінним методом розв'язання певних проблем захисту.

Комп’ютерна стеганографія розвивається в багатьох напрямках, що має багато спільних характеристик, а деякі відмінності в характеристиках зумовлені особливостями практичного застосування. Тому серед стеганографічних систем існують системи, які приховують передачу даних, цифрові водяні знаки, ідентифікаційні номери («відбитки пальців») та заголовки. Завданням будь-якої

стеганографічної системи є розміщення конкретних повідомлень у контейнері, щоб сторонні люди не могли помітити різницю між модифікованим контейнером та оригінальним контейнером. Як правило, система стеганографічна побудована для забезпечення заданого компромісу з її основними характеристиками (такими, як невидимість, стабільність, безпека, пропускна здатність, обчислювальна складність). Розглянемо характеристики кожного типу стеганографічної системи.

Стеганографічна система передачі даних використовується для організації секретних комунікацій. Вони відрізняються від усіх інших контейнерів тим, що в цьому випадку оригінальний вміст контейнера не має значення для відправника або одержувача, який зацікавлений лише в успішній передачі розповсюдженого повідомлення. Однак важливо пам'ятати, що той факт, що контейнер відправляється від відправника до одержувача, не повинен здаватися дивним, і не повинно бути значних відхилень між контейнером та специфікацією. Основна мета цієї системи — приховати присутність сигналу, щоб вона не могла розрізнити порожні та повні контейнери, не знаючи ключа. Для такої системи зазвичай передбачається, що контейнер не буде спотворений під час передачі каналом зв'язку (Y '= Y), оскільки секретний зв'язок здійснюється через відкритий канал цифрової мережі, такої як Інтернет тощо, яка забезпечує передачу інформації без спотворень.

Як вже згадувалося раніше, комп’ютерна стеганографія застосовується у варіативних сферах людської діяльності: секретна передача конфіденційної інформації в мультимедійних файлах; захист авторських прав аудіо- та відеоматеріалів в електронній формі; створення секретних файлів; подолання систем моніторингу та управління мережевими ресурсами; маскування програмного забезпечення, військових інший тип інтелектуальної власності.

1.3. Аналіз актуальності використання стеганографічних методів захисту інформації

Щодо актуальності сучасної стеганографії, можна сказати, що комп’ютерна стеганографія розвивалася дуже швидко: вона базується на різних галузях науки,

використовує існуючі та розробляє нові методи стеганографії. Стеганографічні системи вступають у новий етап свого розвитку, сьогодні більшість із них приховують інформацію з урахуванням характеристик та властивостей стеганографічних контейнерів, що зберігають дані. Тому поки що для багатьох фахівців необхідно бути ознайомленим з базовими знаннями сучасної комп'ютерної стеганографії. Завданням сучасних фіхівців є не лише розробка, аналіз методів стеганографії, а й кваліфікований відбір сучасних методів для їх вдосконалення для використання у вирішенні конкретних проблем інформаційної безпеки.

Зупинимось на кожному завданні стеганографії. Перше завдання передбачає таємну передачу секретної інформації. Метод приховування, призначений для моніторингу та управління мережевими ресурсами, може бути використаний у тих областях, де використовуються надійні методи шифрування. Коли інформація проходить через сервери управління локальної та глобальної комп'ютерних мереж, метод стеганографії призначений протистояти спробам управління інформаційним простором. Іншим важливим завданням стеганографії є маскування програмного забезпечення. Якщо ви не хочете демонструвати програмне забезпечення, ви можете замаскувати його як стандартний програмний продукт загального призначення (наприклад, текстовий редактор) або сховати у мультимедійному файлі (наприклад, у саундтреку чи грі). Зверніть увагу, що інформація не передається від однієї особи іншій, а залишається на диску. У цьому випадку метод стеганографії дозволяє приховати не факт передачі, а факт існування інформації.

При обробці медичних зображень (рентген, результати ЕКГ тощо) необхідно пов'язати саме зображення із даними пацієнта (ім'я, дата, особистий лікар). Метод реалізації може бути використаний для характеристики інформації про пацієнта у графічному файлі, що не тільки усуває випадкову чи навмисну заміну та втрату медичної документації, але й дозволяє автоматично обробляти результати та зберігати результати на комп'ютері. Крім того, медична безпека також досліджує способи психологічного впливу на людину (наприклад, за допомогою 25-го кадру) та вживає заходів для запобігання заподії шкоди.

Ще однією сферою використання стеганографії є захист авторських прав для запобігання піратству. На комп'ютерній графіці є спеціальний знак, який досі невидимий очам користувача, але розпізнається за допомогою спеціального програмного забезпечення. Даний програмне забезпечення програмний продукт використовується в комп'ютерній версії журналів. Ця область стеганографії використовується не тільки для обробки зображень, а й для аудіо- та відеофайлів і призначена для захисту прав інтелектуальної власності. Захист інтелектуальної власності включає введення водяних знаків та відбитків пальців, які доводять, що використання авторських прав незаконно. Водяний знак - це ярлик з авторським правом, де вміст перериваєтся. Відбиток пальця - це мітка, вбудована в копію об'єкта, яка може бути іншою ідентифікацією клієнта (наприклад, прихованим реєстраційним номером). Це дозволяє власникам інтелектуальної власності виявляти клієнтів та виявляти тих, хто порушує ліцензійні угоди.

Існує різниця між класичним формулюванням проблеми стеганографії (яка називається «проблемою ув'язненого») та проблемою захисту авторських прав. У першому питанні успішна атака на систему полягає у виявленні прихованих фактів. У другому питанні - навпаки, кожен може знати про існування вбудованого тегу. Отже, успішна атака на систему полягає не в виявленні тегу, а у його видаленні або модифікації. Метод крадіжки цієї прихованої системи можна запобігти введенням цифрових підписів на основі об'єктів або міток часу. Збільшення місткості бази даних, що містить відео- та аудіопродукцію, вимагає ефективної системи пошуку та ідентифікації вмісту файлу. З цією метою до мультимедійних даних вводиться інформація про автора, виконавця, зміст, дату твору тощо.

Іншим напрямком стеганографії є можливість так званого «розумного заперечення» на основі стеганографічної файлової системи. Секретна файлова система - це механізм захисту, який може забезпечити високий ступінь захисту для запобігання примусовому розголошенню інформації. Цей механізм дозволяє обґрунтовано відхилити існування одного файлу. Навіть якщо зловмисник має повний доступ до системних ресурсів. Методи стеганографії використовуються для автоматичного управління рекламою, що надсилається по радіо - це автоматизовані

системи, призначені для перевірки того, що рекламне повідомлення відтворюється відповідно до контракту.

1.4. Аналіз ризиків використання стеганорграфічних методів захисту інформації

Великі компанії дедалі більше усвідомлюють ризики, пов'язані з приховуванням. Використовуючи брандмауери, системи виявлення вторгнень та інші відповідні запобіжні заходи, які не можуть виявити приховані повідомлення у файлах, хакери можуть виконувати такі завдання, як створення інженерних креслень для проникнення в комп'ютерні системи компанії, наприклад зберігання їх у власних аудіофайлах компанії та вебсайтах. Коли уряди усього світу підозрюють, що такі злочинці, як контрабанда наркотиків, відмивання грошей або терористи, використовують зашифровані повідомлення, вони намагаються отримати ключі шифрування для зчитування повідомлень. Як результат, тих користувачів, хто не використовує методи зв’язку зі зловмисними намірами, можна легко перехопити та прочитати.

Пошук в Інтернеті відкриє тисячі посилань на сторінки зі стеганографією. Посилання містять посилання на програмне забезпечення, яке можна завантажити безплатно, та математичні формули, що описують, як працює ця технологія. Злочинці різного типу намагаються приховувати дані щодня і мають можливість сховати їх в Інтернеті. Коли приховування секретної інформації використовується у зловмисних цілях, ця комунікаційна технологія загрожує безпеці глобальної інформаційної інфраструктури.

Стеганографія представляє загрозу національній безпеці. Хоча стеганографія здається хорошим способом безпечного обміну конфіденційною інформацією, нею також можна зловживати. Існує припущення, що терористи використовують ці методи для спілкування з, здавалося б, невинними вебсайтами через Інтернет. Теорія полягає в тому, що терористичні організації підозрюються у приховуванні карт і фотографій об’єктів тероризму та розміщенні описів терористичної діяльності на

спортивних чатах, порнографічних ресурсах та інших вебсайтах. Насправді незабаром після 11 вересня 2001 року, відповідні органи безпеки просканували понад 2 мільйони зображень з eBay, щоб знайти приховану інформацію, що міститься в ній. Кажуть, нічого не знайшли, але уряд, безсумнівно, бачив ризик.

Загроза, яку представляє стеганографія, цілком реальна. Використання стаганографії неможливо легко виявити або перехопити, оскільки інформація, що транслюється через Інтернет є дуже об'ємною. Приховані повідомлення можна зберігати без нагляду на вебсайті та переглядати з усього світу. Хоча в цей час основною загрозою є забезпечення національної безпеки, ця технологія, безсумнівно, використовується для інших неетичних цілей. Тому, хоча стеганографія ще не є глобальною, IT-аудитори боряться з нею. Це ситуація, яка вимагає розгляду та розуміння можливих майбутніх подій та наслідків.

Не всі методи приховування є поганими. Водяні знаки можна вставити для ідентифікації осіб, що ускладнює підробку тими людьми, хто намагається зробити копії. Іншим позитивним фактором є те, що абсолютно легальна або конфіденційна інформація може передаватися більш надійно. Крім того, компанія використовує технологію, щоб зробити щоденні операції більш безпечними. Наприклад, Digimarc, провідний постачальник безпечних медіарішень, забезпечує безпечні рішення для розпізнавання водяних знаків урядам у всьому світі.

1.5. Дослідження видів стеганографічного захисту інформації

За типом спеціальних атрибутів формату виділяють такі методи стеганографії:

1. На основі формату файлу комп'ютера, зарезервованого для розширення. Поле розширення має кілька форматів мультимедіа і має на меті вдосконалити, оновити формат та зробити нову версію сумісною зі старою версією. Зазвичай ці поля заповнюються нульовою інформацією, і програма не враховує ці поля, тому їх можна використовувати для передачі іншої інформації. Недоліками цих методів є низький ступінь конфіденційності та передача невеликої кількості прихованої інформації.

2. Текстові файли на основі спеціальних форматів, добре відомих давно, використовувались задовго до появи комп'ютерних технологій.

3. На основі приховування дискети в невикористаних місцях. Назва цієї групи само собою зрозуміле, і вона має ті ж переваги та недоліки, що і метод, заснований на використанні зарезервованого розширеного формату.

4. На основі функції імітації - ця стеганографія заснована на формуванні тексту і є узагальненням акровіршу. Для даного прихованого повідомлення буде сформовано змістовний текст, що містить приховане повідомлення. Текст є граматично та граматично правильним і статистично еквівалентний тексту на подібні теми. Такий текст може не викликати сумнівів для системи моніторингу мережі, але він все одно може змусити людей швидко визначити, що зміст тексту не має значення.

5. На основі використання кодів виправлення помилок; дані приховані в додатковій інформації коду захисту від шуму для виправлення випадкових помилок та забезпечення точності передачі цифрової інформації. Якщо інформація прихована, а код одержувача видалений, спостерігач навіть не дізнається, що повідомлення надіслано.

6. На основі видалення заголовка файлу. У цьому методі приховане повідомлення зашифровується, а ідентифікатор заголовка видаляється, залишаючи лише на перший погляд, випадкові зашифровані дані, які можуть бути спотвореною інформацією.

Це лише деякі методи, що ілюструють евристику в стеганографії. Недоліками відомого способу, заснованого на використанні спеціальних властивостей формату файлу, є:

1. Низький рівень конфіденційності (конфіденційність базується на незнанні ворогом прихованого алгоритму);

2. Передайте невелику кількість прихованої інформації. До переваг можна віднести простоту впровадження. Слід зазначити, що вже Опублікував багато програм для реалізації певних алгоритмів.

1.6. Стислий огляд методів стеганографії

Розглянемо кілька методів. Одним з найбільш часто використовуваних методів є алгоритм LSB (Найменший значущий біт), який замінює найменш значущий біт у декількох байтах медіафайлу, щоб приховати послідовність байтів, що містить приховані дані. Це зазвичай ефективно, коли заміна нижчих бітів не спричиняє значної втрати якості.

Задля представлення принципів роботи даного методу можна розглянути приклад за допомогою 24-бітового регістрового зображення RGB. Точка зображення в цьому форматі кодується 3 байтами, і кожен байт відповідає за інтенсивність одного з трьох компонентів зображення.

Коли кольори трансформуються з синього, червоного й зеленого каналів, пікселі отримують бажаний відтінок. Задля чіткішого розуміння способу у який працює даний методу LSB, детальніше розглядається кожен із трьох байтів, представлених у бітовій формі. Молодші числа мають менший вплив на кінцевий образ, ніж старі. З цього можна зробити висновок, що заміна одного або двох нижчих, найменш значущих бітів на будь-який інший біт трохи спотворить відтінок пікселя, але глядач взагалі не помітить цієї зміни. Допустимо, що потрібно сховати 6 цифр у даній точці зображення: 100101. Для цього ми ділимо їх на три пари й замінюємо двома нижчими бітами в кожному каналі.

В результаті ми отримаємо новий відтінок, дуже схожий на оригінальний. Навіть на малих зображеннях такі зміни важко ідентифікувати. Як показано на практиці, людське око не може сприймати заміну двох нижніх бітів. При необхідності можна взяти три цифри, що мало впливає на зображення. Тепер можна розрахувати корисний об'єм цього контейнера для приховування даних за допомогою RGB.

Зайнявши два з восьми бітів на кожному каналі, ми зможемо приховати три байти корисної інформації для кожних чотирьох пікселів зображення, що еквівалентно 25% зображення. Отже, за допомогою файлу зображення розміром 200

Кб ми можемо сховати в ньому будь-які дані розміром до 50 КБ, щоб ці зміни були невидимі неозброєним оком.

Усі контейнери слід розділити на дві категорії: «чисті» та «шумні». У «чистому» контейнері відстежується зв'язок між молодшими бітами, яке потрібно змінити, та іншими бітами елемента, а також видно залежності між наймолодшими бітами. Реалізація повідомлень у «чистому» контейнері порушить теперішні залежності, що дуже легко для спостерігачів. Якщо в контейнері «шумно», визначити вкладення стає складніше. Тому рекомендується використовувати файл, не створений на цьому комп'ютері, як файл-контейнер методу LSB. Метод LSB має як переваги, так і недоліки. До переваг належать:

1. Розмір файлу контейнера залишається незмінним;
2. Неможливо виявити зміну при зміні одного положення;
3. Можливість зміни пропускної здатності шляхом зміни кількості бітів, які потрібно замінити.

Недоліком цього методу є те, що він нестійкий до всіх типів атак. Тобто алгоритм можна використовувати лише тоді, коли в каналі даних немає «шуму». Розглянемо інший метод, парне кодування, який є одним із найнадійніших методів аудіостеганографії. Замість того, щоб розділити сигнал на окремі вибірки, цей метод розбиває сигнал на окремі частини й вбудовує кожен біт секретної інформації. Якщо у вибраній області парні біти не кодуються секретними бітами, то своєю чергою процес змінює найменш значущий біт одного зі зразків області.

Є можливість скористатись методом фазового кодування. Головний принцип роботи даного методу міститься в зміні фази кінцевого звукового сегмента еталонною фазою, яка є секретною інформацією. Налаштуйте інші сегменти фази, щоб підтримувати певну фазу між сегментами. Що стосується відношення сигналу до шуму, фазове кодування є одним з найбільш ефективних методів кодування. Коли різниця фаз між кожною частотною складовою різко змінюється, шум стає очевидним. Однак, якщо фаза сильно не змінюється, людське вухо не може розпізнати будь-яких змін.

Резюмуючи сказане, можна припустити, що даний метод заснований на змінах, які вносяться в аудіофайл, не будуть чутними для людського вуха. Фазове кодування містить етапи:

1. Розподіл звукового сигналу на менші сегменти так, щоб їх загальна довжина дорівнювала довжині повідомлення;
2. Створення матриці фаз за допомогою дискретного перетворення Фур'є;
3. Обчислювання різниці фаз між сусідніми сегментами;
4. Оскільки фазовий зсув між двома сусідніми сегментами можна легко виявити, різниця фаз повинна зберігатися в стеганографічному сигналі. Тому секретне повідомлення вбудовується лише на стадії першого абзацу;
5. Використовування нової фази першого сегмента для створення нової матриці фаз та відмінностей між ними;
6. Відновлення звукового сигналу, застосувавши зворотне дискретне перетворення Фур'є, використовуючи нову матрицю та вихідну матрицю значень, а потім з'єднати звукові сегменти.

Одержувач повинен знати довжину сегмента, щоб отримати секретне повідомлення з аудіофайлу. Тоді приймач може використовувати дискретне перетворення Фур’є для отримання секретної інформації.

При приховуванні звуку метод розширеного спектра намагається передавати секретну інформацію про частотний спектр звукового сигналу. Цей метод чимось схожий на вищезгаданий метод LSB, який випадковим чином розподіляє біти повідомлень у цілому аудіофайлі. Однак, на відміну від методу LSB, метод розширеного спектра використовує код, який не має нічого спільного з фактичним сигналом для розподілу секретної інформації по частотному спектру аудіофайлу. В результаті пропускна здатність, зайнята кінцевим сигналом, перевищує розмір, необхідний для передачі.

У порівнянні з LSB, фазовим та парним методами кодування, метод розширеного спектра може зробити значний внесок у продуктивність із помірною швидкістю передачі даних та вищою стабільністю. Однак метод розширеного спектра має очевидний недолік - він генерує шум в аудіофайлі.

Метод ехо вбудовує секретну інформацію в аудіофайл, вводячи тим самим ехо в дискретний сигнал. У порівнянні з іншими методами, основними перевагами цього методу є висока швидкість передачі даних та вища стабільність. Якщо з вихідного сигналу можна виділити лише одне відлуння, тоді можна закодувати лише один біт інформації. Отже, перед процесом кодування вихідний сигнал ділиться на блоки. Після кодування ці блоки об'єднуються, утворюючи кінцевий вихідний сигнал.

Висновки до розділу 1

Підсумовуючи, можна зробити висновок, що основним недоліком використання методу ехо-сигналу, методу розширеного спектра та парності є те, що вони генерують шум в аудіофайлах, що цілком очевидно для людського вуха, і надійність цих методів сумнівна. Щодо фазового кодування, головним недоліком цього методу є низька швидкість передачі даних, оскільки секретне повідомлення кодується лише в першому сегменті сигналу. Тому цей метод застосовується лише при передачі невеликої кількості даних.

Серед вищезазначених стеганографічних методів найменш значущий біт або метод LSB є найпростішим методом вбудовування інформації. Метод LSB дозволяє кодувати велику кількість даних в аудіофайли, забезпечуючи вищий рівень безпеки, ніж інші методи. Це ефективний метод приховування інформації про класифікацію від зловмисників. Він може гарантувати, що розмір файлу є постійним навіть після кодування, і застосовується для будь-якого типу формату аудіофайлів.

Конкретна реалізація будь-якого з цих способів реалізації тісно пов'язана з фізичними властивостями сигнального повідомлення та контейнера сигналу. У більшості випадків аудіосигнали та зображення вибираються як контейнери. Для того, щоб якісно відтворювати звукові сигнали та зображення, сучасні технології використовують цифровий запис таких сигналів. Завдяки своїй аналоговій природі звукові сигнали та зображення містять надлишкову інформацію, яку можна легко замінити конкретними повідомленнями. Бітовий склад стеганографічного об’єкту відрізняється від бітового складу контейнера, який не повинен сприйматися

людиною і має значний вплив на роботу телекомунікаційної системи. Виняток становлять багато методів реалізації цифрових водяних знаків, коли водяний знак діє як приховане повідомлення і може бути видимим або накладати певні обмеження на редагування контейнерів анонімного перегляду.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ВЛАСТИВОСТЕЙ ТА МЕТОДІВ СТЕГАНОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

2.1. Характеристики стеганографічних методів захисту інформації

Завдання будь-якої системи, яка реалізована за допомогою стеганографії - вставити повідомлення в контейнер, щоб жоден сторонній спостерігач не помітив різниці між оригінальним контейнером і модифікованим контейнером. Як правило, система побудована для забезпечення компромісу з її основними характеристиками, включаючи нематеріальність, стабільність, безпеку, пропускну здатність створеної стеганографії та обчислювальну складність реалізації.

Включення секретного повідомлення повинні зберігати сприйняту якість оригінального контейнера. Для аудіосигналів повідомлення повинно бути нечутним, для зображень - невидимим. Нематеріальність повідомлення може бути досягнута шляхом мінімальних модифікацій перетворення стеганоперетворенні контейнера, наприклад, на рівні квантування під час оцифрування. Крім того, врахування особливостей слухової та зорової системи людини допомагає досягти нематеріальності. Тому людське вухо працює в режимі аналізатора частоти, який інтегрований у критичний діапазон слуху. Він здатний сприймати коливання від 20 до 20 000 Гц і найбільш чутливий до звукових компонентів на частотах від 500 до 6000 Гц. Наступні функції слухової системи людини можуть бути використані для розробки аудіо стереометодів:

1. Модифікація компонентів звукового сигналу нижче абсолютного звукового порогу, які непомітні для людини;

2. Поріг чутності деяких звукових компонентів зміниться за наявності інших звукових компонентів: коли звук гучніший (маскується ним), слабка, але чутна звукова вібрація стає непомітною;

3. Коли люди сприймають звукові сигнали, крім частоти маскування, існує також маскування за часом, яке поділяється на маскування початкове та маскування кінцеве.

4. На практиці нечутливими числовими показниками зазвичай є відношення сигналу до шуму SNR, стандартна похибка MSE, максимальна різниця MD тощо.

Для створення надійної стеганогарфічної системи необхідно забезпечити її стабільність. Суть поняття стабільності залежить від типу нападу, який характерний для певної стеганографічної системи. Тому для потенційних систем передачі даних пасивні атаки є найбільш характерними, тому в цьому випадку під стійкими переважно розуміють системи, які можуть ефективно протидіяти їм.

Для інших типів стеганографічних систем стійкість зазвичай оцінюється за кількістю помилок, які допускає законний користувач, коли законний користувач отримує повідомлення з контейнера після деформації контейнера внаслідок ненавмисної або активної атаки. Наприклад, дослідження стійкості до процесів друку та сканування повинні проводитись із станографічними контейнерами для захисту інформації на папері.

Необхідний рівень стабільності визначається використанням системи. Тому, коли йдеться про стійкість до активних атак, існують системи ЦВЗ зі стабільними, крихкими та напівкрихкими водяними знаками. Розглянемо стабільність пасивних та активних моделей більш докладно.

Коли й тільки коли несанкціоновані користувачі не можуть розрізнити порожні контейнери та заповнені контейнери, особливо за допомогою візуального та статистичного аналізу, система вважається здатною протистояти пасивним атакам.

Найпоширеніші програмні продукти, які використовують комп’ютерну стеганографію для передачі прихованої інформації, мають різні модифікації найменш значущого бітового методу, і їх суть полягає в тому, щоб замінити нижній біт контейнера бітом, який приховує повідомлення. Користувач вибирає довільний контейнер, його розмір дозволяє розмістити в ньому повідомлення і в результаті

отримується прихований контейнер, який візуально не відрізняється від порожнього контейнера.

Існують кореляційні зв’язки між низькими бітами сусідніх елементів контейнера та між низькими бітами елементів контейнера та іншими бітами, які можуть бути знищені підчас модифікації підчас сеганографічного процесу. У цьому випадку достатньо простого візульаного аналізу бітових зрізів для виявлення стеганографічного контейнераа. Як правило, через похибки квантування та інші фактори під час зацифрування цифрових контейнерів, вони стають біль стійкими, ніж контейнери, які з самого початку були цифровими. Однак, вставляючи повідомлення в НЗБ за допомогою шумного контейнера, воно повинно розподілятися по нижньому рівню всього контйнера, тому що, інакше різницю між незміненою та зміненою розсіяною частиною можна виявити за допомогою візуальних атак на відповідні фрагменти бітів.

Місткість визначається як максимальна кількість даних повідомлення, яка може бути розподілена до одного елемента контейнера за умови, що він відповідає вимогам нематеріальності та стабільності.

Сьогодні існують різні, а іноді діаметрально протилежні методи визначення обсягу прихованої інформації. Ці відмінності спричинені такими факторами, як інформаційна безпека, типи порушників, їх можливості, типи контейнерів та повідомлень. Зокрема, як теоретично досяжна межа вона не залежить від характеристик фактичного застосування, а використовуються оцінки пропускної здатності, отримані в теорії та інформаційній моделі стеганографічної системи.

Місткість визначає обсяг потенційної інформації, яку можна приховати тим чи іншим методом стеганографії. А обсяг, який фактично використовується в процесі стеганграфії конкретного контейнера (тобто додавання додаткової інформації), ми будемо називати повнотою контейнера. Очевидно, що в тій чи іншій стеганографічній системі наповнення будь-якого контейнера не може перевищувати потужність створеної нею стеганографічної системи. Заповнення зручно вимірювати як відсоток місткості. Отже, ступінь наповнення порожнього контейнера становить 0%, а максимальна кількість заповнення - 100%.

2.2. Галузі застосування стеганографічних методів захисту інформації

Цифрова стеганографія, яка нещодавно ідентифікувала себе як наука включає такі напрямки:

1. Вбудовування інформації для негласної передачі;
2. Вбудовування цифрових водяних знаків (ЦВДЗ);
3. Вбудовування ідентифікаційного номера (відбиток пальця);
4. Вбудовування підзаголовку.

ЦВДЗ може бути в основному використаний для запобігання копіюванню та несанкціонованому використанню. Завдяки бурхливому розвитку мультимедійних технологій, захист авторських прав та прав інтелектуальної власності в цифровій формі набув гострого значення. Наприклад, фотографії, аудіо- та відеозаписи тощо. Переваги представлення та передачі повідомлень у цифровому вигляді можуть бути затьмарені легкістю викрадення чи модифікації. Тому розробляються різні засоби захисту інформації, організації та технічні засоби. Одним з найбільш ефективних технічних засобів захисту мультимедійної інформації є вбудова в об'єкт захисту невидимих міток- ЦВДЗ. Розвитком цієї галузі керують одні з провідних компаній. Оскільки метод ЦВДЗ почав розвиватися лише нещодавно, існує багато незрозумілих проблем, які потребують вирішення.

Назва цього методу походить від відомого способу захисту цінних паперів (включаючи гроші) від підробки. Термін «цифровий водяний знак» вперше був використаний у 1993 році. На відміну від звичайних водяних знаків, ЦВДЗ не тільки видно, але, як правило, невидно. Спеціальний декодер аналізує невидимий ЦВДЗ для визначення його правильності. ЦВДЗ може містити деякий реальний код, інформацію про власника або будь-яку контрольну інформацію. За допомогою ЦВДЗ найбільш придатними об'єктами захисту є нерухомі зображення, аудіо- та відеодані.

Технологія вбудовування ідентифікаційного номера виробника містить в собі чимало спільного з ЦВДЗ. Різниця полягає в тому, що в першому випадку кожна захищена копія має свій унікальний вбудований номер (звідси та назва - «відбиток

пальця» в прямому сенсі). Цей ідентифікаційний номер дозволяє виробникам відстежувати майбутнє своїх нащадків: чи брав участь хтось із покупців у незаконному копіюванні. Якщо так, «відбиток пальця» швидко вкаже на винуватця.

Вбудовані заголовки використовуються для підписання медичних зображень тощо. Метою є збереження варіативно відображеної інформації в цілому. Це може бути єдиним застосуванням стеганографії, де немає очевидного потенційного зловмисника.

Цифрова стеганографія - молода наука, тому її термінологія ще не до кінця сформована. Основна концепція стеганографії була узгоджена на першій міжнародній конференції з приховування даних. Однак навіть поняття "Стеганографія" має різні тлумачення. Так, деякі дослідники розуміють під стеганографією лише неявну передачу інформації. Інші називають стеганографію такими додатками, як метеорний радіозв'язок, псевдовипадкова настройка радіочастот та широкосмуговий радіозв'язок. Неформальне визначення цифрової стеганографії може бути таким: "Наука про невидиме та надійне приховування певних послідовностей бітів в інших послідовностях бітів аналогової природи". До цього визначення належать усі чотири області приховування даних. Тому радіозастосунків немає. Крім того, це визначення містить дві основні вимоги до стеганографічного перетворення: невидимість і надійність або стійкість до різних деформацій. Посилання на аналоговий характер цифрових даних підкреслює той факт, що вбудовування інформації здійснюється в зацифрований безперервний сигнал. Тому цифрова стеганографія не може розв'язувати проблему вбудовування даних у заголовки пакетів даних IP та файли різних форматів у текстові повідомлення.

Незалежно від напрямку стеганографії, їх вимоги в основному однакові, як показано нижче. Найбільша різниця між встановленням завдання прихованої передачі даних та встановленням вбудованого ЦВДЗ полягає в тому, що в першому випадку зловмисник повинен виявити приховане повідомлення, тоді як у другому випадку всі знають про його існування. Щобільше, зловмисник може законно

володіти обладнанням для виявлення ЦВДЗ (наприклад, як частина програвача DVD).

Слово «невидимий або непомітний» у нашому визначенні цифрової стеганографії означає примусити людину бути включеною до системи передачі даних стеганографії. Людина тут може розглядатися як додатковий приймач даних, що висуває дуже важко формалізувати вимоги системи передачі.

На рисунку 2.1 представлена класифікація систем цифрової стеганографії. Стенографічна система утворює стеганографічний канал, а заповнений контейнер передається через стеганографічний канал. На думку Г. Сіммонса, стеганографія, як наука, займається вирішенням так званої «проблеми в'язня».

Суть «проблеми в'язня» полягає в наступному: всупереч тому, що прохід між ними контролює охоронець В, двоє в'язнів А та Б хочуть потайки обмінюватися інформацією. Щоб зробити секретні повідомлення можливими, припустимо, що і А, і Б знають деякі секретні ключі. Дії В можуть не лише намагатися виявити приховані канали зв'язку, але також можуть порушити передані повідомлення, змінити їх та створити нові помилкові повідомлення. Виходячи з цього, ми можемо виділити три типи правопорушників, які повинні чинити опір стеганографічній системі: пасивні правопорушники, активні правопорушники та кримінальні злочинці. Зверніть увагу, що пасивні зловмисники можуть перебувати лише у стеганографічній системі, яка приховує передачу даних. Активні та кримінальні правопорушники є типовими представниками системи ЦВДЗ.

Для того, щоб зробити систему надійною, при конструюванні потрібно дотримуватися багатьох вимог.

Безпека системи повинна повністю визначатися конфіденційністю ключа. Це означає, що зловмисник може повністю зрозуміти всі алгоритми системи стеганографії та статистичні характеристики набору повідомлень і контейнерів, і це не дасть йому жодної додаткової інформації про наявність або відсутність повідомлень у контейнері.

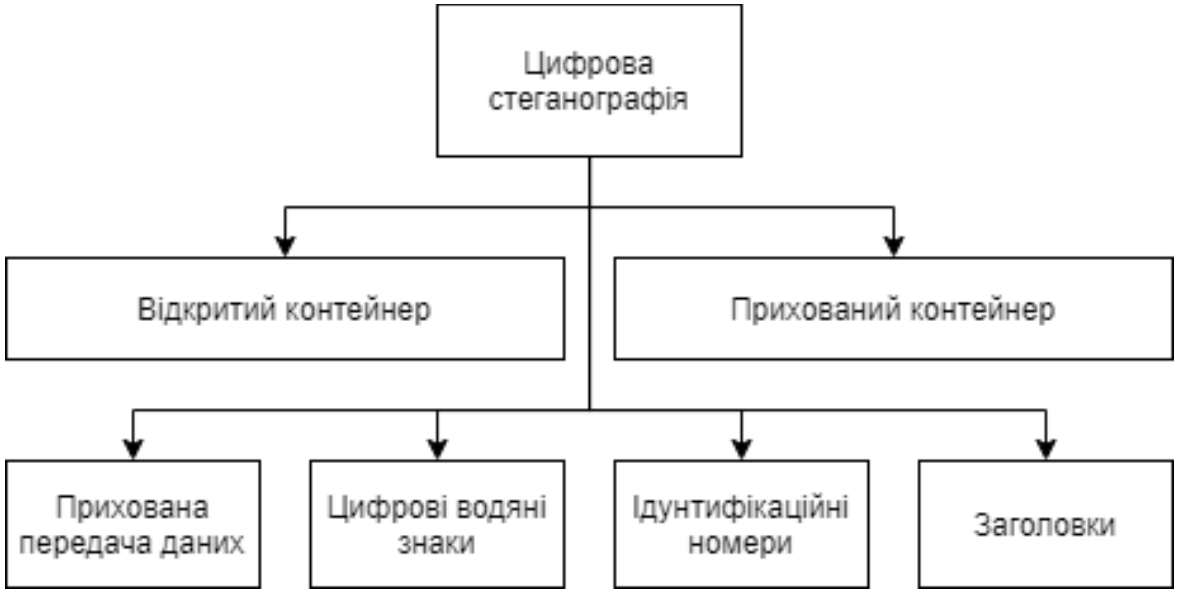

Рисунок 2.1 – Класифікація систем цифрової стеганографії

Знання зловмисником існування повідомлень у контейнері не повинно допомогти йому виявити повідомлення в інших контейнерах.

Наповнений контейнер не повинен візуально відрізнятися від пустого контейнера. Щоб задовольнити цю вимогу, начебто прихована інформація повинна бути вбудована у візуально незначний набір сигналів.

Отже, якщо зображення ще більше стиснути, прихована інформація може бути знищена. Виходячи зі сказаного, біти повинні бути вбудовані у візуально значущий набір, а відносна невидимість може бути досягнута за допомогою спеціальних методів, таких як модуляція з розширеним спектром.

Ймовірність того, що система ЦВДЗ помилково виявляє приховані повідомлення в сигналах, які її не містять, повинна бути дуже низькою. У деяких додатках таке виявлення може призвести до серйозних наслідків. Наприклад, неправильне виявлення ЦВДЗ на DVD-диску може призвести до того, що програвач припинить відтворення.

Необхідно забезпечити необхідну пропускну здатність (ця вимога в основному пов'язана зі стеганографічною системою для потенційної передачі інформації). У третій частині ми ввели поняття прихованої пропускної здатності та розглянули шляхи її реалізації.

Система повинна мати прийнятну обчислювальну складність. Одночасно система ЦВДЗ, тобто складний стеганографічний кодер і простий стеганографічний декодер, може бути асиметричною у складності реалізації. Поставимо такі вимоги до ЦВДЗ:

1. Законні користувачі повинні легко отримати ЦВДЗ;

2. ЦВДЗ має бути стабільним або нестійким проти навмисних та випадкових впливів.

3. Якщо ЦВДЗ використовується для підтвердження правдивості, то неприпустима заміна контейнера повинна призвести до знищення ЦВДЗ (крихкого ЦВДЗ).

4. Якщо ЦВДЗ містить ідентифікаційний код, логотип компанії тощо, він повинен зберігатися при максимальній деформації контейнера, що, звичайно, не призведе до значної деформації вихідного сигналу. Наприклад, ви можете редагувати кольорову гаму або яскравість зображення, ви можете посилити низькі частоти в аудіозаписі тощо.

5. Крім того, ЦВДЗ має бути здійсненним з точки зору афінного перетворення зображення (тобто, обертання зображення, масштабування).

Необхідно розрізняти стабільність ЦВДЗ і здатність декодера його правильно виявити. Наприклад, коли ви обертаєте зображення, ЦВДЗ не складатиметься, і декодер не зможе його вибрати. У деяких додатках ЦВДЗ повинен бути стабільним для одних перетворень, а для інших - нестабільним. Наприклад, можна дозволити копіювання зображень (копіювальних апаратів, сканерів), але заборонити будь-які зміни до них.

Має бути можливість додати додатковий ЦВДЗ до знака. Наприклад, ярлик DVD-диску дозволяє одноразово копіювати. Після виготовлення такої копії необхідно додати ярлик, що забороняє подальше копіювання. Звичайно, можна видалити перший ЦВДЗ і замінити його другим ЦВДЗ, але це суперечить припущенню, що ЦВДЗ неможливо видалити. Найкращий вихід додати ще одне ЦВДЗ, і перше не буде розглядатися надалі. Однак, якщо не вживаються спеціальні

заходи, наявність кількох ЦВДЗ у повідомленні може допомогти зловмисникові здійснити напад.

На цей час технологія ЦВДЗ знаходиться на початковій стадії розробки. Практика показує, що для того, щоб новий метод криптографії широко застосовувався в суспільстві, знадобиться 10 - 20 років. Подібну ситуацію можна спостерігати також за допомогою стеганографії. Однією з проблем ЦВДЗ є різні вимоги до них, залежно від програми. Розглянемо основний принцип використання ЦВДЗ більш детально.

Спочатку розглянемо питання піратства або необмеженого несанкціонованого копіювання. Наприклад, А продає свої мультимедійні повідомлення П. Хоча інформацію можна зашифрувати під час передачі, ніщо не заважає П скопіювати її після розшифровки. Тому в цьому випадку необхідний додатковий рівень захисту від копіювання, який не може бути забезпечений традиційними методами. ЦВДЗ може бути введений, щоб дозволити читання та заборонити копіювання інформації.

Важливим питанням є визначення дійсності отриманої інформації, а саме підтвердження особи. Засоби цифрового підпису зазвичай використовуються для перевірки даних. Однак ці інструменти не підходять для забезпечення автентифікації мультимедійної інформації. Річ у тім, що повідомлення з електронними цифровими підписами повинні зберігатися і передаватися «поступово» з абсолютною точністю. Мультимедійна інформація може дещо спотворюватися під час зберігання (через стиснення) та під час передачі (ефект поодиноких або пакетних помилок у каналі зв'язку). Однак його якість все ще прийнятна для користувачів, але цифрові підписи не працюватимуть. Одержувач не зможе розрізнити реальні, хоча і дещо спотворені, та неправдиві повідомлення. Крім того, мультимедійні дані можна перетворювати з одного формату в інший. В той самий час традиційні методи захисту цілісності неможливі.

Можна сказати, що ЦВДЗ може захищати вміст аудіо- та відеоповідомлень, а не цифрові подання у вигляді бітових послідовностей. Крім того, важливим недоліком цифрового підпису є те, що його легко видалити з перевіреного повідомлення, а потім додати до нього новий підпис. Видалення підпису змусить

зловмисника відмовитись від авторства або ввести автора повідомлення в оману законним одержувачем. Конструкція системи ЦВДЗ повинна виключати можливість таких порушень. Це видно на рисунку 2.2: застосування ЦВДЗ не обмежується програмами захисту інформації.

Рисунок 2.2 – Сфери застосування стеганографічних додатків

Основні методи використання технології ЦВДЗ можна розділити на чотири групи: захист від копіювання (використання), приховані анотації документів, сертифікація правдивості інформації та прихована комунікація. Популяризація мультимедійних технологій призвела до багатьох досліджень, пов’язаних із розробкою стандартних алгоритмів ЦВДЗ, що використовуються в MP3, MPEG-4, JPEG2000 та DVD для запобігання копіюванню.

2.3. Стеганографічні середовища

Революція комп’ютерних технологій та Інтернету надала стеганографії особливого значення. Носії інформації можна розділити на різні типи даних, такі як текст, диск, аудіо, зображення, звук, мережевий трафік або інші цифрові дані.

Для того, щоб приховати інформацію в тексті (лінгвістична стеганографія), використовується звичайна письмова мова або формат подання тексту. Найважче приховати об’єкт електронної версії тексту, оскільки його друкована версія може бути зображенням в електронному вигляді, який обробляється відповідними методами. У порівнянні з зображеннями чи аудіофайлами ця складність зумовлена головним чином відносною відсутністю надлишкового тексту. Хоча це може зробити модифікацію зображення невидимою для людського ока або людської слухової системи зміни звуку аудіофайлу, будь-які зайві літери, зайві символи або додаткові знаки пунктуації можуть бути виявлені випадковим читачем.

Існує три основні методи приховування даних у тексті, найпоширеніші:

1. Метод довільного інтервалу;
2. Синтаксичний метод
3. Семантичний метод.

У більшості випадків стеганографічний метод графічних зображень із контейнерними ролями застосовується з таких причин:

1. Поширювати цифрові фотографії та відеозаписи, які потребують запобігання незаконному копіюванню та розповсюдженню;

2. Порівняно велика кількість графіки та зображень забезпечує широкий простір (великий розмір) для приховування даних;

3. Розмір контейнера відомий заздалегідь, що дозволяє вибрати найкращий контейнер;

4. Людське око відносно слабке до незначних змін у цифровій графіці;

5. Нещодавно добре розроблені методи цифрової обробки зображень.

Метод стеганографії рідко використовується для відеоданих, оскільки файл складається з рухомих зображень (кадрів) та звукових доріжок. Також варто зазначити, що звукові доріжки та кадри все ще не часто використовуються як контейнери.

В цей час існує три основних методи приховування інформації у відеоданих:

1. Метод вбудовування на рівні коефіцієнта - біти прихованого повідомлення вбудовуються в коефіцієнт. З огляду на використання алгоритмів стиснення,

основною проблемою є накопичення зсувів та помилок. Для зменшення змін використовуються додаткові спеціальні сигнали. Через обмеження швидкості передачі даних під час вбудовування змінюється лише 10-12% коефіцієнт. При використанні цього методу прихована інформація зберігається під час фільтрації, шуму (додаткового шуму) та вибірки.

2. Метод вбудовування інформації на рівні бітової площини має характеристики високої пропускної здатності та простоти обчислення. Але є головний недолік: вбудовану таким чином інформацію можна легко видалити. Якщо послідовність бітів повторити, якість відео погіршиться, а прихована інформація буде знищена.

3. Метод вбудовування інформації через різницю енергій між коефіцієнтами заснований на диференціальному вбудовуванні енергії. Цей метод може бути використаний для багатьох алгоритмів стиснення. Вбудовувати інформацію, видаляючи деякі коефіцієнти.

В основному, приховування у відео використовуватиме метод приховування у аудіо та у зображенях, оскільки це вже відеозображення зображень та звуків. Відео складається з рухомих зображень. Насправді це перевага, оскільки користувач навіть не помітить будь-яких незначних спотворень через безперервний обсяг даних.

Спеціально розроблений метод стеганографії приховування інформації у звуковому середовищі. Його характеристика полягає в тому, що слуховий апарат людини працює в надширокому динамічному діапазоні та має досить невеликий діапазон відмінностей. Виходячи з цього, можна зробити висновок, що в аудіофайлах є багато місця для прихованих даних. Крім того, слуховий апарат не може розрізнити абсолютні фази, лише відносні фази. Крім того, існують деякі спотворення, спричинені зовнішнім середовищем, які можна використовувати для приховування даних.

Через великий частотний діапазон особливо важко приховати дані в аудіофайлах. Аудіосигнали також чутливі до випадкових шумів. Якщо шум у звуковому файлі становить від одного до одного мільйона, шум можна виявити.

Приховуючи звуки, користувачі повинні скористатися слабкими сторонами людських слухових апаратів, але вони також повинні звернути увагу на їх високу чутливість.

2.4. Методи стеганографічного захисту інформації

2.4.1. Найменш значущий біт

Дуже популярним методом є алгоритм LSB (Найменший значущий біт), який замінює найменш значущий біт у деяких байтах файлу, щоб приховати послідовність байтів, що містить приховані дані. Зазвичай це ефективна методика у випадках, коли заміна LSB не спричиняє значного погіршення стану файлу.

Під час обчислень найменш значущий біт (LSB) - це позиція біта в двійковому цільовому числі, яка дає особливе значення, тобто вона визначає, парне чи непарне число. LSB іноді вважають найбільш правильним бітом, завдяки умові в позначенні положення писати менш важливі числа праворуч. Це схоже на найменш важливе десяткове ціле число, число в крайньому (правому) положенні.

Двійкове подання десяткового числа 149 освітлення LSB. MSB у 8-бітному двійковому значенні представляє значення зі 128 знаками після коми. LSB вказує, що значення дорівнює 1.

Наприклад, щоб приховати букву «a» (код ASCII 97, 01100001) у восьми байтах обкладинки, ви можете встановити LSB кожного байту таким чином: 1001001001010011100110111101001010001010000000100111001000101011

Програма, яка декодує обкладинку, зчитує вісім найменш значущих бітів цих байтів для відтворення прихованого байта -0110001-літери «a». Як бачите, за допомогою цього методу ви можете приховати один байт на кожні вісім байт обкладинки. Зверніть увагу, що ймовірність того, що біт, який потрібно замінити, така ж, як і замінений біт, становить 50%, тобто частота, коли біт не змінюється, що допомагає мінімізувати погіршення якості.

Цей метод є найпопулярнішим методом дослідження при приховуванні цифрової аудіоінформації (та інших типів носіїв інформації). У цій техніці LSB послідовність кожного зразка оцифрованого аудіофайлу замінюється двійковим еквівалентом секретного повідомлення. Це найпростіший спосіб вставити інформацію в цифровий аудіофайл. Це дозволяє приховати великі обсяги даних в аудіофайлах або забезпечити більш високу швидкість вкладення, не впливаючи на якість звукових файлів.

Еквівалентна частота дискретизації коливається від 8 кбіт за секунду до 44,1 кбіт за секунду. У кодуванні LSB ідеальна швидкість передачі даних становить 1 кбіт на секунду на частоті 1 кГц. Однак у деяких варіантах кодування LSB два найменш значущих біта замінюються двома бітами повідомлення. Це збільшує обсяг даних, які можна закодувати, але також збільшує кількість шуму, який з'являється в аудіофайлі. Тому перед тим, як прийняти рішення про використання операції LSB, слід врахувати вміст сигналу.

Наприклад, аудіофайли, записані в шумній станції метро, можуть маскувати низькошвидкісний шум кодування. З іншого боку, буде звучати той самий голос у звуковому файлі, що містить соло фортепіано. Щоб отримати секретне повідомлення з кодованого LSB аудіофайлу, одержувач повинен отримати доступ до зразкової індексної послідовності, яка використовується в процесі вбудовування. Як правило, довжина секретного повідомлення, яке кодується, менше, ніж зразок аудіофайлу. Потім вам потрібно вирішити, як вибрати 7 зразків підмножин, що містять секретне повідомлення, і повідомити про це рішення одержувача. Простий прийом полягає у виконанні кодування LSB від початку аудіофайлу до повного вбудування повідомлення; решта зразків залишаються незмінними. Однак це створює проблеми із безпекою, оскільки перша частина аудіофайлу матиме різні статистичні властивості, ніж друга частина немодифікованого аудіофайлу. Одним із рішень цієї проблеми є розміщення секретного повідомлення із випадковими бітами так, щоб довжина повідомлення дорівнювала загальній кількості вибірок. Однак процес введення зараз закінчується зміною зразків набагато більшою, ніж необхідна

секретна передача. Це збільшує можливість викриття майбутніми зловмисниками секретних повідомлень.

Більш складним методом є використання генератора псевдовипадкових чисел для розподілу повідомлень у аудіофайлі у випадковому порядку. Популярним методом є використання методу випадкових інтервалів, при якому секретний ключ, що належить відправнику, використовується як насіння в генераторі псевдовипадкових чисел для створення випадкової послідовності індексів вибірки. Одержувач може також отримати доступ до ключа та знань генератора псевдовипадкових чисел, щоб можна було відновити випадкову послідовність вибірок.

Однак потрібно встановити перевірку, щоб запобігти дворазовому генеруванню псевдовипадкових чисел. Якщо це трапиться, конфлікт відбудеться, коли змінений режим у деяких повідомленнях буде знову змінений. Конфліктні проблеми можна вирішити, щоб відстежувати всі використані зразки. Інший метод полягає у використанні захищеної хеш-функції для обчислення підмножини вибірки за допомогою псевдовипадкового розташування всього набору. Цей метод гарантує, що один і той же індекс не буде генеруватися більше одного разу.

2.4.2 Паритетне кодування

Кодування за парністю - один із надійних методів стеганографії голосу. Цей метод не розбиває сигнал на окремі вибірки та вставляє кожен біт конфіденційного повідомлення в біт парності. Якщо біт парності не відповідає секретному біту, що кодується, процес інвертує LSB одного з вибірок у регіоні. Таким чином, відправник має більше можливостей для кодування секретних бітів.

Процес декодування розпочинається, коли декодер отримує секретне повідомлення шляхом обчислення та розподілу регіональної паритетності, яка використовується в процесі кодування. Подібним чином, відправник і одержувач можуть використовувати спільний ключ як насіння в генераторі псевдовипадкових чисел для створення того самого набору регіональних вибірок. Є 2 основні недоліки

використання таких методів, як кодування LSB. Вухо є чутливим, навіть якщо метод кодування парності досить близький, щоб не чути вхідний шум, він може виявити найменший шум, що надходить в аудіофайл.

Обидва ці методи мають ще один недолік, оскільки вони ненадійні. Якщо аудіофайл, вбудований у секретне повідомлення з кодуванням LSB або парним кодуванням модифікувати, вбудована інформація буде втрачена. Використання методів резервного копіювання при кодуванні секретних повідомлень може трохи збільшити потужність.

2.4.3 Фазове кодування

Спосіб фазового кодування працює, замінюючи фазу вихідного аудіо сегмента еталонною фазою, що представляє інформацію про класифікацію. Відрегулюйте решту фаз сегментів, щоб зберегти відносну фазу між сегментами. Що стосується відношення сигналу до шуму, фазове кодування є одним з найбільш ефективних методів кодування. Коли фазова залежність між кожною частотною складовою різко зміниться, буде очевидна фазова дисперсія. Однак, поки фазова модифікація досить мала, може бути досягнуто беззвучне кодування. Цей метод заснований на тому, що фазова складова звуку - це не те, що сприймає людське вухо.

Фазове кодування враховує недоліки методу шумопоглинання аудіостеганографії. Фазове кодування залежить від того, що фазова складова звуку менш чутлива до людських вух, ніж шум. Ця технологія не вносить збурень, але кодує біти повідомлення як фазовий зсув фазового спектра цифрового сигналу, тим самим досягаючи безшумного кодування відношення сигналу до шуму. Або ми можемо сказати, що фазове кодування залежить від заміни вибраних фазових компонентів прихованими даними. Зверніть увагу, що у всіх методах приховування фазове кодування запобігає спотворенню сигналу. Фазове кодування використовує незалежну багатосмугову фазову модуляцію для вставки даних у фазову складову. У цьому методі тонка фазова модифікація досягається за допомогою керованої фазової

зміни звукового хоста. Оригінальний звуковий сигнал розділений на менші сегменти, довжина яких дорівнює розміру закодованого повідомлення.

Кодування фаз описується в наступному порядку:

1. Розділити оригінальний звуковий сигнал на менші сегменти, щоб довжина була такою ж, як розмір закодованого повідомлення.

2. Створити фазову матрицю, застосовуючи дискретне перетворення Фур'є (DFT).

3. Вирахувати різницю фаз між сусідніми відрізками лінії.

4. Фазовий зсув між сусідніми сегментами легко виявити. Це означає, що ми можемо змінити абсолютну фазу відрізка лінії, але відносна різниця фаз між сусідніми відрізками лінії повинна бути збережена.

5. Секретна інформація лише вставляється у фазовий вектор першого сегмента сигналу наступним чином: якщо біт дорівнює одиниці, то фаза буде дорівнювати двом, а якщо нулю , то муніс двом.

6. Використати нову фазу першого сегмента, щоб створити нову матрицю фаз та початкову різницю фаз.

7. Реконструювати звуковий сигнал, застосувавши зворотне дискретне перетворення Фур'є, використовуючи нову фазу матриці та матрицю вихідного значення, а потім об'єднайте різні частини звуку.

Одержувач повинен знати довжину сегмента, щоб витягти конфіденційну інформацію з аудіофайлу. Потім приймач може використовувати DFT для отримання сцени та вилучення секрету.

Одним недоліком, пов'язаним з фазовим кодуванням, є низька швидкість передачі даних, оскільки секретне повідомлення кодується лише в першому сегменті сигналу. Це можна вирішити, збільшивши довжину сегмента сигналу. Однак це суттєвіше змінить фазову залежність між кожною частотною складовою сегмента, полегшуючи виявлення коду. Тому, коли ви хочете приховати лише невелику кількість даних (наприклад, водяний знак), використовуйте метод фазового кодування.

2.4.4 Розповсюдження спектру

В аудіостеганографії основним методом поширення спектра є спроба розповсюдження класифікаційної інформації за спектром звукових сигналів. Це схоже на систему, реалізовану за допомогою LSB, яка випадковим чином розподіляє біти повідомлень по всьому аудіофайлу. Однак воно відрізняється від кодування

LSB - це метод поширення спектру, який використовує коди, незалежні від фактичного сигналу, для розподілу секретної інформації по спектру аудіофайлів. Як результат, смуга пропускання, зайнята кінцевим сигналом, перевищує смугу дійсно необхідну для передачі.

Порівняно з кодуванням LSB та фазовим кодуванням, метод поширення спектра може покращити продуктивність у деяких областях, оскільки забезпечує помірну швидкість передачі даних та високу надійність щодо методу видалення. Однак метод поширення спектра має головний недолік, він вносить шум в аудіофайл.

2.4.5 Приховування відлуння

Технологія приховування відлуння використовує секретну інформацію в аудіофайлах для введення відлуння в дискретні сигнали. У порівнянні з іншими методами, приховування відлуння має переваги у забезпеченні високої швидкості передачі даних та вищої надійності.

Якщо у вихідному сигналі приймається лише одне відлуння, можна закодувати лише один біт інформації про класифікацію. Тому перед процесом кодування вихідний сигнал ділиться на блоки. Після завершення процесу кодування ці блоки об'єднуються для створення кінцевого сигналу.

Для успішного приховування даних три параметри відлуння повинні бути різними: амплітуда, швидкість ослаблення та зміщення (час затримки) вихідного сигналу. Всі три параметри встановлені нижче порогу людського слуху, тому ехо неможливо легко виявити. Крім того, зміщення змінено, щоб представляти двійкове

повідомлення, що використовується для кодування. Одне значення зміщення є двійковим значенням, а друге значення зміщення - двійковим нулем.

Якщо вихідний сигнал виробляється лише ехом, код може містити лише один біт інформації. Тому перед початком процесу кодування початковий сигнал ділиться на кілька блоків. Після завершення процесу кодування ці блоки об'єднуються для створення кінцевого сигналу. Тепер ми використаємо повідомлення "HEY", щоб заповнити просту форму процесу приховування відлуння.

Для простоти, хоча зразки випадкових чисел повинні зберігатися між кожною парою блоків за звичайних обставин, щоб зменшити можливість виявлення, сигнал буде повністю розділений на кілька блоків

Спочатку сигнал ділиться на кілька блоків, а потім кожному блоку призначається один або нуль відповідно до секретного повідомлення. У цьому випадку повідомлення є двійковим еквівалентом "HEY". За допомогою цієї реалізації процесу приховування відлуння зазвичай можна отримати сигнал із досить очевидним набором відлуння, що збільшує ризик виявлення.

Друга реалізація процесу прихованого відлуння вирішує цю проблему. Спочатку відлуння генерується з усього вихідного сигналу з використанням двосмугового значення зміщення нуля. Потім значення зміщення подвійної частоти використовується для формування другого ехо-сигналу з усього вихідного сигналу.

Отже, «одне» ехо містить лише одне, а «нульове» ехо містить лише нуль. Для того, щоб поєднати два відлуння для отримання остаточного коду, використовуються два сигнали змішувача.

Значення сигналу змішувача складають 1 і 0 відповідно, залежно від бітів, які потрібно кодувати в блоці. Ехо-сигнал «один» множиться на сигнал «один» змішувача, а «нульовий»" ехо-сигнал відтворюється на «нульовий» сигнал змішувача. Потім два результати додаються для отримання кінцевого сигналу.

Остаточний сигнал не такий чіткий, як сигнал, отриманий при першій реалізації приховування відлуння. Це пояснюється тим, що два ефекти змішувача доповнюють один одного, і ці переходи використовуються в кожному сигналі. Ці дві характеристики сигналу змішувача забезпечують більш плавний перехід між ехо.

Для того, щоб отримати секретне повідомлення із сигналу, приймач повинен мати можливість розділити сигнал на ту саму послідовність блоків, яка використовується в процесі кодування.

2.4.6 Порівняння методів

Узагальнимо недоліки старіших методів та те, чим вони відрізняються від сучасних методів. Основні недоліки використання чинних методів (наприклад, приховування відлуння, розповсюдження спектра та кодування перевірки на парність) дуже чутливі до шуму, і вони зазвичай можуть виявляти навіть найменші шуми, що вносяться в аудіофайл, а інша проблема - надійність.

Через те, що секретне повідомлення кодується лише в першому сегменті сигналу, фазове кодування має головний недолік низької швидкості передачі даних. Тому використовуйте цей метод лише тоді, коли вам потрібно передати невелику кількість даних.

Серед різноманітних методів приховування інформації, запропонованих для вбудовування інформації в аудіофайли, найменш значущий біт (LSB) - це найпростіший метод вбудови інформації про класифікацію в цифрові аудіофайли, замінюючи найменш значущу бітову інформацію аудіофайлів на двійкову. Тому метод LSB дозволяє кодувати велику кількість інформації про класифікацію в аудіофайлах.

Процес використання LSB для приховування конфіденційної інформації:

- Сховати аудіофайли в потоці бітів.
- Конвертувати кожен символ секретного повідомлення в бітовий потік.
- Замінити звуковий біт LSB знаковим бітом LSB у секретному повідомленні.

Висновки до розділу 2

Цей розділ розкриває теоретичні аспекти наявних алгоритмів для розв’язання задач аудіостеганографії. Розглянуто та проаналізовано порівняння переваг та недоліків описаних методів, специфіку реалізації та способи використання.

Метод LSB показує кращий та надійніший рівень безпеки та є ефективним методом приховування конфіденційної інформації від хакерів та надсилання її до місця призначення безпечним та нерозкритим способом.

Запропонована система також гарантує, що розмір файлу не зміниться навіть після кодування, а також застосовується до будь-якого типу аудіофайлу. У порівнянні з іншими алгоритмами, це також дозволяє приховати багато інформації про класифікацію у файлі контейнера. Оскільки він перевершує інші алгоритми, його було обрано для модифікації, про що буде сказано в наступному розділі.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Математична постановка задачі

Звичайний процес стеганографічного перетворення описується так:

$$E: C \times M \rightarrow S, \tag{3.1}$$

$$D: S \rightarrow M, \tag{3.2}$$

$$S = \{(c_1, m_1), (c_2, m_2), \dots, (c_q, m_q)\} = \{s_1, s_2, \dots, s_q\}; \tag{3.3}$$

S (3.3) – множина стеганограм(заповнених контейнерів). Залежність (3.1) описує процес приховування інформації, залежність (3.2) – процес витягнення інформації, що прихована.

Отже, загалом стеганосистема - це сукупність контейнерів (оригінал та результат), повідомлень та перетворень, що їх пов’язують.

Є два способи вибрати контейнер:

1. Довільний
2. вибір найбільш підходящого контейнера в конкретній ситуації, який найменше змінюється під час процесу перетворення

3.2 Вибір набору даних

Як контейнери для стеганографічних перетворень, мною були вибрані аудіо файли типу WAV без стиснення. Враховуючи обмеження людського слуху, стиснення даних не вплине на сприйняття якості звуку. А враховуючи те, що обрані мною контейнери, формуються без стиснення – кінцевий користувач не почує

різницю навіть при переповненому контейнері. Також безумовною перевагою такого вибору є те, що нестиснені аудіо файли мають значно більший розмір вихідного файлу, що дозволяє стеганографічно приховувати в контейнері більшу кількість інформації, тобто можна приховувати файли більшого розміру, ніж це б могло бути можливим з контейнером, який є аудіо файлом без стиснення.

Розмір таких контейнерів, що є аудіо файлами WAV без стиснення, прямо пропорційно залежить від глибини звучання, часу звучання та кількості каналів на аудіо доріжці.

Як можливі файли для використання в якості прихованої інформації, не використовуються файли якогось конкретного типу, чи розширення. Це значно звужує можливості використання такого програмного продукту та цілковито знецінює його потенціал, як продукту, що дозволяє приховувати конфіденційну інформацію в WAV контейнерах та передавати їх через відкриті канали зв»язку. Адже інформація – це не тільки файл якогось конкретного типу, чи розширення, а це(маючи на увазі інформацію в комп»ютерних системах) – певний набір байт, який структурований і логічно навантаженений. Тому, в реалізації даного програмного продукту, реалізована можливість використовувати, в якості конфіденційних даних, будь-які типи файли та файли, що є бінарними.

3.3 WAV – файли

Файли WAV є прикладом формату RIFF, створеного IBM та Microsoft. Формат RIFF є "обгорткою" для кодування аудіо різних форматів.

Хоча файли WAV можуть містити стиснене аудіо, найпоширенішим форматом аудіо WAV є нестиснуте звучання з лінійною імпульсно-кодовою модуляцією (LPCM). LPCM також є стандартним форматом кодування звуку для аудіо компакт-дисків. Він використовується для зберігання двоканального звуку LPCM. Частота дискретизації становить 44100 Гц, а кожен зразок - 16 біт. Оскільки LPCM не стискає та не зберігає всі зразки звукової доріжки, професійні користувачі або експерти з аудіо можуть використовувати формат WAV із звуком LPCM, щоб

отримати найкращу якість звуку. Файли WAV також можна порівняно легко редагувати та обробляти за допомогою програмного забезпечення.

Таблиця 3.1

Структура формату файлу WAV

Позиція (HEX)	Позиція	Розмір	Назва	Пояснення
• RIFF Заголовок				
0000	0	4 байти	ChunkID	Містить літери "RIFF" в ASCII кодуванні
0004	4	4 байти	ChunkSize	36 + SubChunk2Size Це розмір всього файлу в байтах, не враховуючи 8 байтів перших двох полів: ChunkID і ChunkSize.
0008	8	4 байти	Format	Містить літери "WAVE" " в ASCII кодуванні
• “fmt “ підсекція				
000C	12	4 байти	Subchunk1ID	Містить літери "fmt " в ASCII кодуванні
0010	16	4 байти	Subchunk1Size	Приймає значення 16 для формату кодування PCM.
0014	20	2 байти	AudioFormat	PCM = 1 (Лінійне квантування)
0016	22	2 байти	NumChannels	Кількість звукових каналів. Моно = 1, Стерео = 2, і т.д.
0018	24	4 байти	SampleRate	Частота дискретизації у Герцах.

Продовження таблиці 3.1

001C	28	4 байти	ByteRate	SampleRate * NumChannels * BitsPerSample/8
0020	32	2 байти	BlockAlign	NumChannels * BitsPerSample/8 Кількість байт, яка міститься в одному семплі враховуючи кількість каналів.
0022	34	2 байти	BitsPerSample	Кількість біт в одному семплі. Так звана "глибина" чи точність звучання.
• LIST підсекція				
		4 байти	Subchunk2ID	Містить літери "LIST" в ASCII кодуванні.
		4 байти	Subchunk2Size	Розмір додаткових даних в байтах, які містяться в наступному полі.
		X	LIST	Додаткові дані
• Data підсекція (якщо в заголовку відсутній LIST – тоді Subchunk2)				
0024	36	4 байти	Subchunk2ID / Subchunk3ID	Містить літери "data" в ASCII кодуванні.
0028	40	4 байти	Subchunk2Size / Subchunk3Size	Розмір даних звукозапису в байтах, які містяться в наступному полі.
002C	44	X	Data	Фактичні дані звукозапису.

3.4 Цілісність файлу після атаки

Значення хеш-функції. Це типовий метод перевірки цілісності даних, який широко застосовується в різних протоколах та додатках. Це відіграє важливу роль у сучасній криптографії. Основна ідея про хеш-функцію полягає в тому, що хеш діє як компактний делегат, який називається відбитком пальця або цифровим відбитком вхідного об'єкта. Ці функції в поєднанні з моделлю цифрового підпису широко використовуються для перевірки цілісності даних.

Однією з основних категорій хеш-функцій є коди автентифікації повідомлень. Це дозволяє використовувати симетричний метод для ідентифікації повідомлень. Цей прийом використовує два основних вхідних параметра - повідомлення та ключ. Основна мета - спростити поєднання з іншими механізмами цілісності даних для різних додатків.

Контрольна сума. Контрольна сума є одним з основних методів перевірки вертикально. Її значення залежить від порівняння між введеним об'єктом та значенням, отриманим після кодування. Цей метод в основному використовується спільно з обчисленням хеш-функцій. Метод порівняння значень контрольної суми двох об'єктів допомагає виявити зміни в цілісності. Однак, оскільки вхідні та вихідні контрольні суми не збігаються, він не може відновити дані. Збережена контрольна сума може бути пошкоджена або змінена. Іншою причиною проблеми відновлення значення контрольної суми є те, що коли дані неможливо відновити для отримання значення контрольної суми, вони обчислюються за допомогою односторонньої хеш-функції.

3.5. Опис Методу

В даному методі приховування даних за допомогою стеганографічних перетворень в аудіофайлах-контейнерах типу WAV – було розроблено модифікацію алгоритму LSB – Last Significant Bit. За основу був взятий підвид LSB алгоритму зі зміною тільки одного найменш значущого біта в аудіоконтейнері.

Такий непростий вибір був зроблений задля того, щоб забезпечити більшу надійність при приховуванні файлу. Хоча такий метод веде до прямопропорційного зменшення можливого розміру файлу, що буде прихований в аудіоконтейнері, але дає значно менше спотворень і шуму у вихідному заповненому контейнері.

Більшість алгоритмів стеганографічних перетворень, на основі методу Last Significant Bit, мають можливість працювати тільки з текстовими потоками даних. Тобто це або можливість ввести обмежену кількість тексту з клавіатури, а потім, при витягуванні , цей текст тільки буде виведений на екран комп»ютера, або можливість зчитати тільки файли з розширенням .txt і в такі ж файли витягувати приховані повідомлення.

Я вважаю такі обмеження значно суттєвими для ефективного користування програмним продуктом, який реалізує приховування інформації зі допомогою стеганографічних методів. Тому в процесі підготовки контейнера для приховування в нього даних – спочатку зчитуємо заголовок самого контейнеру, а після цього формуємо додатковий заголовок. Структура заголовку така:

Таблиця 3.2

Стуркура додаткового заголовку

fileSize	nameSize	name
4 bytes	**4** bytes	**nameSize** bytes

Спочатку зберігаємо розмір файлу, який потрібно приховати в контейнері – fileSize. Ці дані займають 4 байти. Далі зберігаємо розмір повної назви файлу, що потрібно приховати в контейнері. Ці дані будуть потрібні при витягування файлу з контейнеру і займають 4 байти. Далі зберігаємо повну назву файлу, що приховується, включаючи його розширення. Ці даті також будуть потрібні для коректного витягнення файлу з контейнера і займають nameSize байт. Загалом такий заголовок займає 4 + 4 + nameSize байт.

Такий заголовок не записується зразу після основного заголовку WAV файлу, бо тоді б пішло пошкодження даних аудіофайлу, який використовується в якості

контейнера. Цей заголовок буде доданий на початок основного байтового потоку файлу, який потрібно приховати в контейнері, і буде прихований в ньому разом з основний файлом.

Після формування додаткового заголовку йде обчислення можливості рівномірного приховування файлу в контейнері. При умові , що така можливість є за даних умов – йде обчислення рівномірного розподілення інформації по контейнеру. Ця можливість прямопропорційно залежить від співвідношення кількості байт, які можливо приховати в контейнері до розміру файлу, що потрібно приховати + розмір додаткового заголовку.

$$N = \frac{Subchunk2Size - extHeaderSize * BitsPerSample}{BitsPerSample} \quad (3.4)$$

$$M = \frac{N}{HidingFileSize + extHeaderSize} \quad (3.5)$$

Де:

1. N – максимально можлива кількість байт, що можна приховати в даному контейнері;
2. M – коефіцієнт для рівномірного роподілення файлу, що приховується, по контейнеру;
3. Sunchunk2Size – розмір даних аудіофайлу (в байтах), не враховуючи розмір заголовку, в яких зберігається аудіодоріжка;
4. extHeaderSize – розмір додаткового заголовку (в байтах), який формується в процесі підготовки контейнера до стеганографічних перетворень;
5. BitsPerSample – глибина звучання аудіопотоку (кількість біт на один семпл);
6. HidingFileSize – розмір (в байтах) файлу, який приховується в контейнері.

Можливість розподілити приховування даних рівномірно по всій можливій довжині контенера не дозволяє скупчувати приховані дані в одній області контейнера. Ця можливість є за умови, що M > 0.

Після закінчення всіх підготовчих процесів та розрахунків – йде процес приховування файлу в стеганоконтейнері. Для забезпечення більшої надійності прихованого файлу і контейнері – ми використовуємо криптографічне шифрування даних перед тим, як приховати їх в контейнері.

В якості криптографічного шифрування я використовую потокове шифрування на основі чотирьохбайтового регістру зсуву з лінійним зворотним зв”язком. Безумовними перевагами вибору такого методу є:

1. висока швидкодія криптографічного алгоритму, що дозволяє потоково шифрувати великі об»єми пам»яті, не дуже навантажуючи робочий комп»ютер, і не сповільнювати роботу кінцевого програмного продукту;
2. застосовуються лише найпростіші операції додавання та множення, які реалізовані апаратно;
3. хороші криптографічні властивості.

Висновки до розділу 3

У рамках цього розділу було сформульовано змістовну постановку задачі, яка має бути реалізована в ході роботи над виконанням реалізації програмного продукту. Також була опрацьована математична постановка задачі.

Було наведено змістовне обгрунтування вибору алгоритму LSB(Last Significant Bit), який був взятий за основу для роботи над реалізацією програмного продукту.

Оглянуто існуючу модифікацію алгоритму LBS, яка була створена та реалізована в рамках роботи над розробкою програмного продукту. Також було обгрунтовано всі реалізовані модифікації та наведено їх простоту та ефективність в роботі.

Етапи дослідження описуються поетапно та подається набір вхідних даних, які були вибрані для дослідження та реалізації роботи програмного продукту з ними. Було наведено обгрунтування вибору таки всіхдих даних.

У наступному розділі наводяться та описуються експериментальні результати та демонстрація розроблених модифікацій.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ТА ОПИС ПРОГРАМНОГО ПРОДУКТУ

4.1 Засоби для розробки

В ході реалізації програмного продукту мною були використані такі засоби розробки програмно-технічних рішень для операційної системи Microsoft Windows версії 10:

1. Microsoft Visual Studio Community 2019
2. C++ 2019 Redistributable Pack
3. Microsoft Visual Studio Code
4. C++ Programming language
5. Git Version Control

Для роботи над реалізацією програмного продукту була вибрана мова програмування високого рівня C++. Мова C++ підтримує декілька парадигм програмування:

1. Об’єктно-орієнтована
2. Узагальнена
3. Процедурна

Незважаючи на те, що мову програмування C++ найчастіше використовують для написання програмних продуктів згідно парадигми ООП(об’єктно-орієнтоване програмування) – в моїй реалізації коду до програмного продукту використовується змішаний підхід між парадигмами ООП та процедурною.

Використання мови програмування C++ зумовлене тим, що на відміну від таких мов програмування, як C# , чи Java – зумовлена такими чинниками:

1. мова програмування C++ є значно швидшою в роботі кінцевого програмного продукту
2. мова програмування C++ має прямий доступ до пам’яті

3. мова програмування C++ займає значно менше оперативної пам’яті під час виконання кінцевого програмного продукту

4. виконавчий файл, скомпільований та зібраний з коду, який написаний мовою програмування C++ займає значно менше пам’яті на жорсткому диску комп’ютера

Синтаксис мови програмування C++ дуже схожий з такими мовами програмування, як C# та Java. Це зумовлено тим, що ці дві мови програмування – ідейні спадкоємці мови C++. Java Virtual Machine взагалі написана повністю мовою програмування C++. Тому зрозуміти код програмного продукту буде нескладно для тих, хто працював хоч з однією з перелічених мов програмування.

Так як вибраною операційною системою для реалізації програмного продукту є Microsoft Windows 10, а програмую я під систему GNU/Linux – то це створило деякі труднощі. Так як для операційної системи Microsoft Windows 10 немає таких програмних засобів, як система збірки GNU Make та компілятор GCC(GNU Compiler Collection) – було вирішено використати інтегроване середовище програмування від компанії Microsoft.

Microsoft Visual Studio 2019 - інтегроване середовище програмування, яке включає такі програмні засоби, як компілятор, інструмент відлагодження програми, систему автоматичної збірки та інші. Саме це середовище програмування було мною вибране тому, що в ній легко створити новий проект і настроїти його. Інтегроване середовище програмування автоматично настроює всі залежності, систему збірки та компілятор. Основний недолік Microsoft Visual Studio 2019 є те, що це середовище програмування займає буде багато місця жорсткому диску, займає значну частину оперативної пам’яті при роботі та досить таки відчутно навантажує систему при роботі.

Також великим плюсом інтегрованого середовища програмування Microsoft Visual Studio 2019 є можливість компілювати та компонувати програмний продукт в Debug та Release версії і настроєні системні препроцесори, наприклад, _Debug.

Це допомагає, наприклад, розділити деякі дані, які потрібні при розробці, але вони не повинні бути включені в фінальну збірку програмного продукту. На

прикладі просто логера (Рисунок 4.1), який дозволяє уніфікувати логи програми під час її роботи, можемо побачити, що в Debug версії виконуваного файлу, і тексті лог-повідомлень є додаткова інформація (Рисунок 4.2). Це інформація про те, в якому файлі знаходилась функція, в якій було створене повідомлення та номер рядка коду в цьому файлі, а в Release версії ця інформація відсутня (Рисунок 4.3).

logger.hpp > ...

```
1   #pragma once
2
3   #include <iostream>
4   #include <string>
5
6   #define LOG_INF  "LOG_INFO"
7   #define LOG_WARN "LOG_WARNING"
8   #define LOG_ERR  "LOG_ERROR"
9
10  #define __FILENAME__ (strrchr(__FILE__, '\\') ? strrchr(__FILE__, '\\') + 1 : __FILE__)
11
12  #ifdef _DEBUG
13      #define LOG(LOG_TYPE, MESSAGE)                                  \
14          do                                                          \
15          {                                                           \
16              std::cout <<                                            \
17                  LOG_TYPE << ": [" << __FILENAME__ << " : " <<       \
18                  __FUNCTION__ << "() : " << __LINE__ << "] : " << \
19                  MESSAGE <<                                          \
20              std::endl;                                              \
21          } while(0);                                                 \
22
23  #else
24      #define LOG(LOG_TYPE, MESSAGE)                                  \
25          do                                                          \
26          {                                                           \
27              std::cout <<                                            \
28                  LOG_TYPE << ": " << MESSAGE <<                      \
29              std::endl;                                              \
30          } while(0);                                                 \
31
32  #endif
33
```

Рисунок 4.1 – Приклад використання Release/Debug в інтегрованому середовищі програмування Microsoft Visual Studio 2019

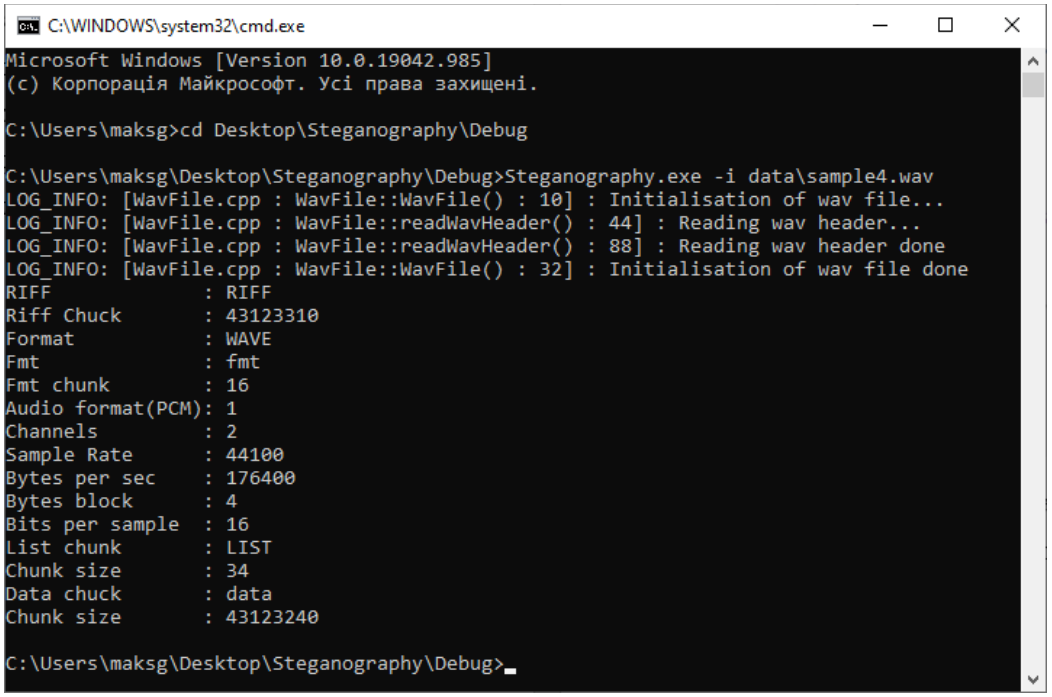

Рисунок 4.2 – Приклад повідомлень в Debug версії програмного продукту

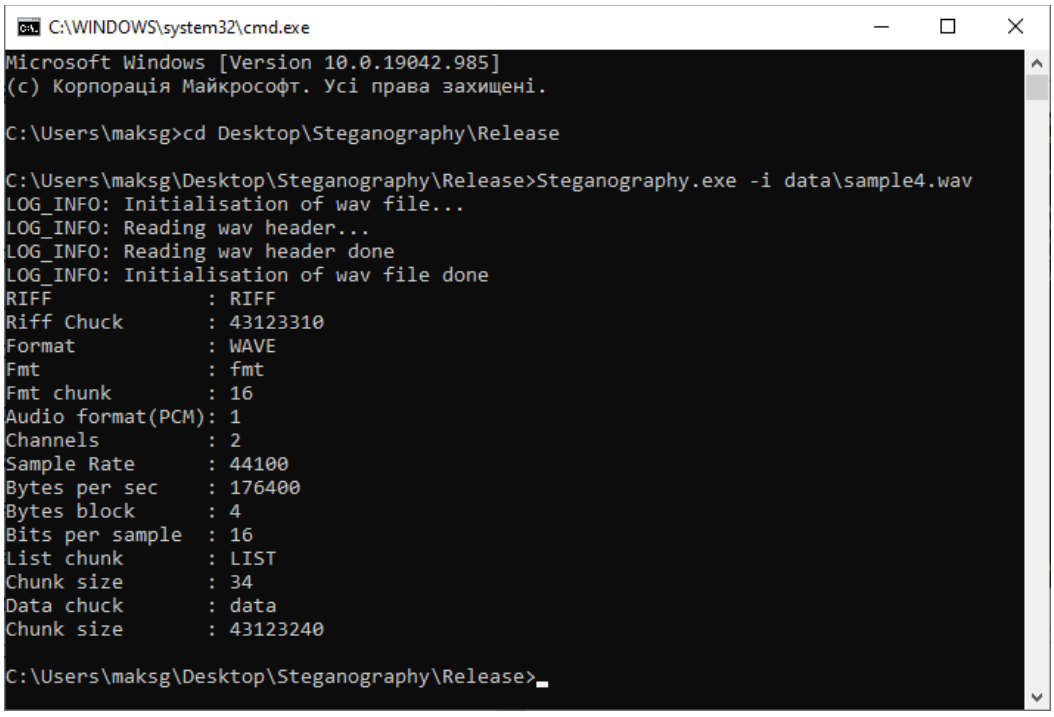

Рисунок 4.3 – Приклад повідомлень в Release версії програмного продукту

Незважаючи на всі переваги інтегрованого середовища програмування Microsoft Visual Studio 2019 – було вирішено використовувати його для компіляції та компонування програмного продукту.

Текстовим редактором для роботи з кодом програмного продукту був вибраний редактор Microsoft Visual Code. Цей текстовий редактор розповсюджується безкоштовно, на відміну від того ж інтегрованого середовища програмування.

Також в процесі роботи над програмним продуктом була використана система контролю версій Git. Система контролю версій є невід'ємною частиною процесу розробки будь-якого програмного забезпечення. Готові до використання виконувані файли можна знайти за цим посиланням: https://github.com/mhedeon/Steganography/tree/release_binaries.

4.2 Модель роботи програмного продукту

Далі наведено модель роботи програмного продукту.

На рисунку 4.4 Зображено модель роботи програмного продукту при приховуванні секретного файлу в пустому контейнері.

На рисунку 4.5 Зображено модель роботи програмного продукту при витягуванні секретного файлу із стеганограми.

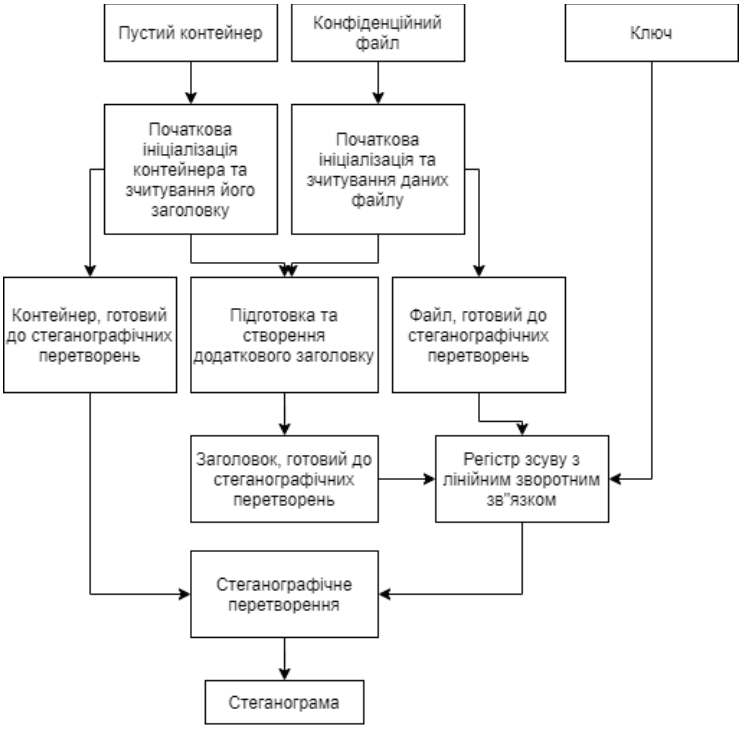

Рисунок 4.4 – Модель роботи програмного продукту при приховуванні секретного файлу в пустому контейнері

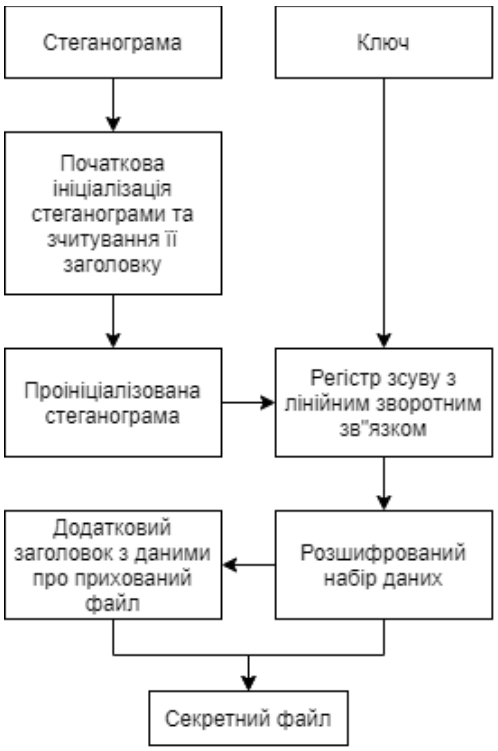

Рисунок 4.5 – Модель роботи програмного продукту при витягуванні секретного файлу із стеганограми

4.3 Демонстрація програмного продукту

В процесі виконання даної роботи було розроблено програмний продукт для стеганографічних перетворень. Так як однією з вимог до даного програмного продукту була мінімальна вага виконуваного файлу – результатом розробки програмного продукту є консольний застосунок без користувацького інтерфейсу. В результаті розмір готового програмного продукту складає всього лише 25 кілобайт.

Такий вибір несе в собі зразу декілька переваг:

1. мінімальний розмір виконуваного файлу
2. можливість запуску через скрипти, чи інші програми
3. можливість використовувати як окремий модуль
4. можливість підключення персоніфікованої графічної оболонки над виконуваним файлом

Отже, на рисунку 4.6 бачимо, що програма завершила свою роботу з виводом в командний рядок інформацією про те, як правильно запускати даний застосунок. Це сталось тому, що ми неправильно ввели команду запуску.

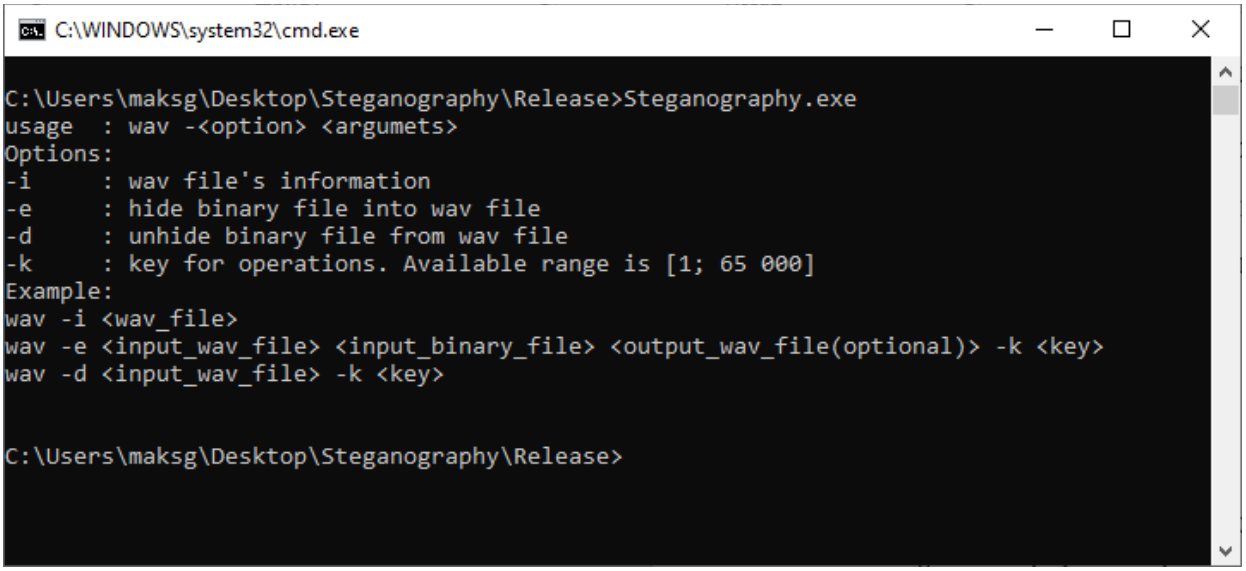

Рисунок 4.6 – Запуск програмного продукту з неправильними параметрами запуску

Підготуємо деякі файли для демонстрації коректної роботи програмного продукту. Ці файли збережемо в папці «data», в тому ж каталозі, де знаходиться виконуваний файл програмного продукту.

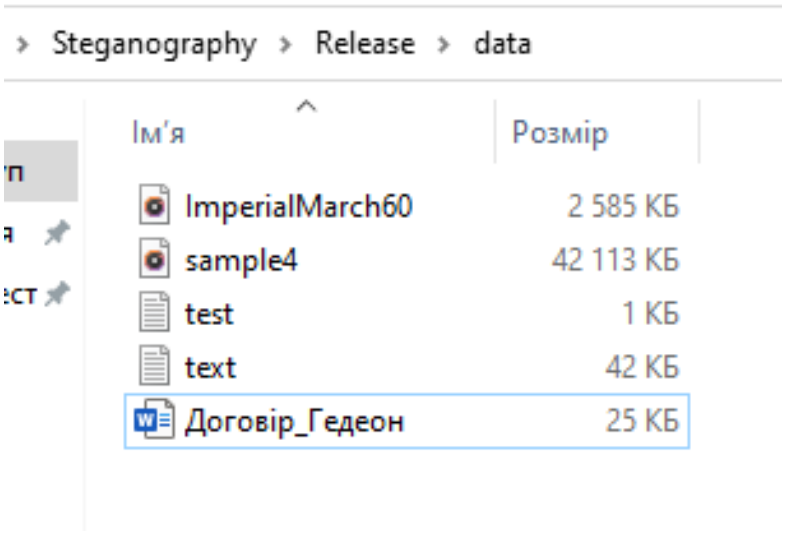

Рисунок 4.7 – Файли для демонстрації коректної роботи програмного продукту

Файл “test.txt” містить в собі деякий набір символів (Рисунок 4.8). Файл “text.txt” містить в собі текст мого звіту з науково-дослідної практики (Рисунок 4.9).

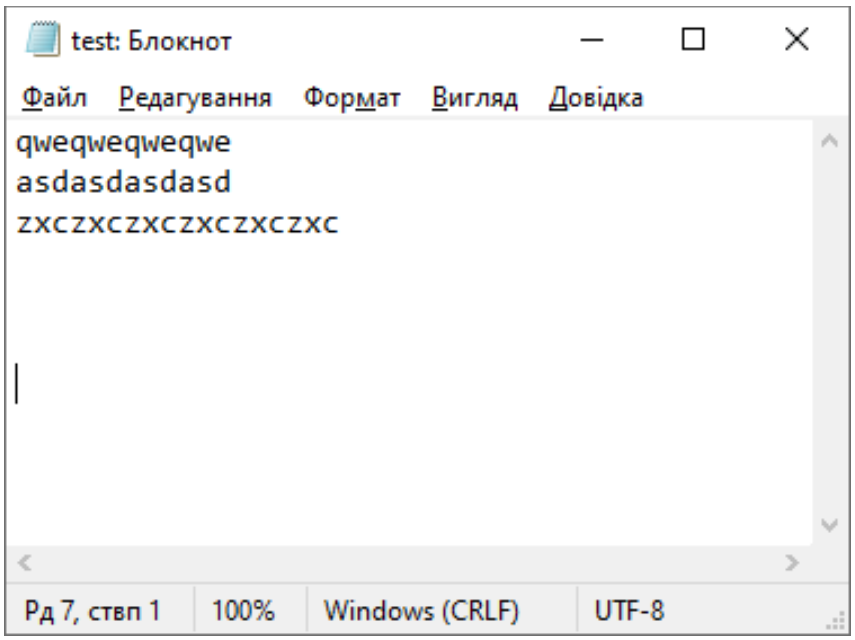

Рисунок 4.8 – Вміст файлу “test.txt”

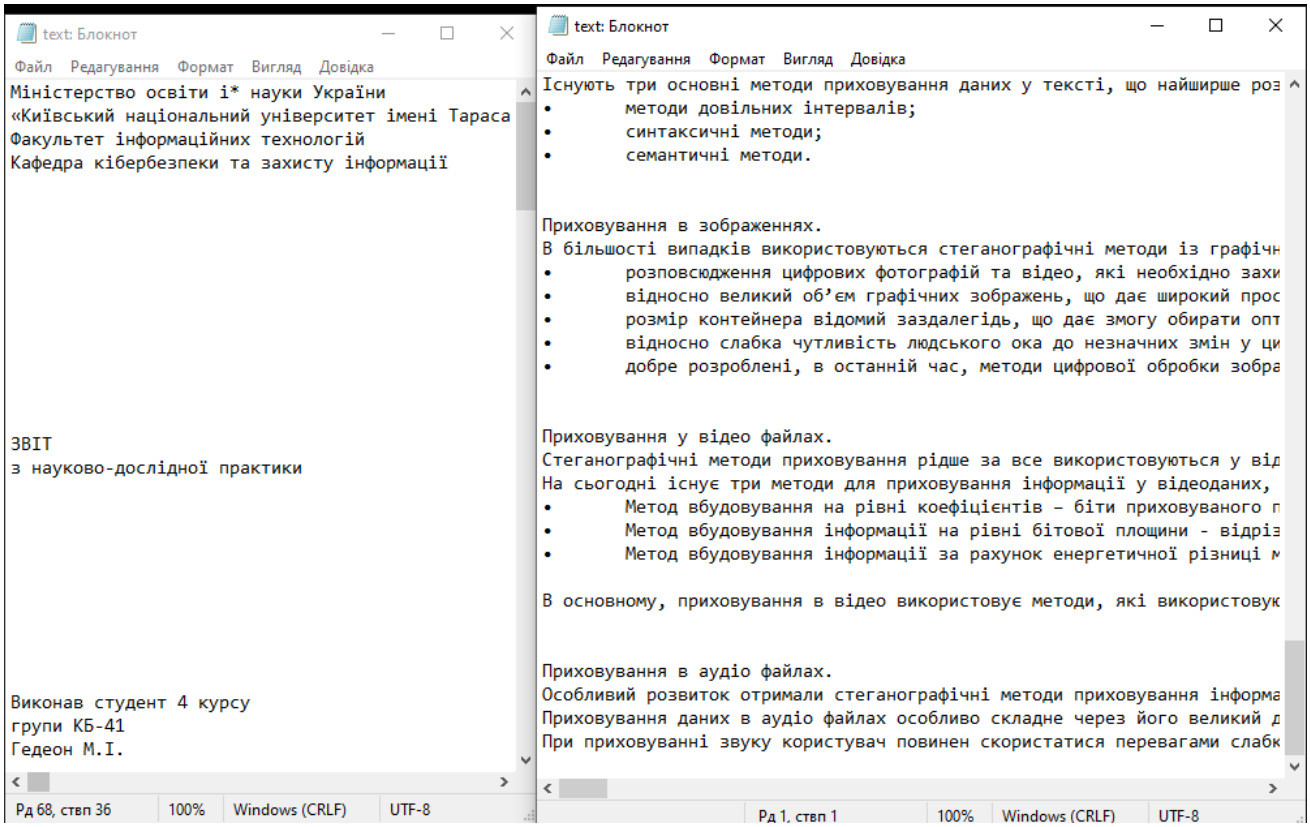

Рисунок 4.9 – Вміст файлу “text.txt”

Отже, тепер все готово для запуску програмного продукту. Спробуємо спочатку дізнатись інформацію з заголовків двох аудіофайлів “sample4.wav” і “ImperialMarch60.wav” (Рисунку 4.10).

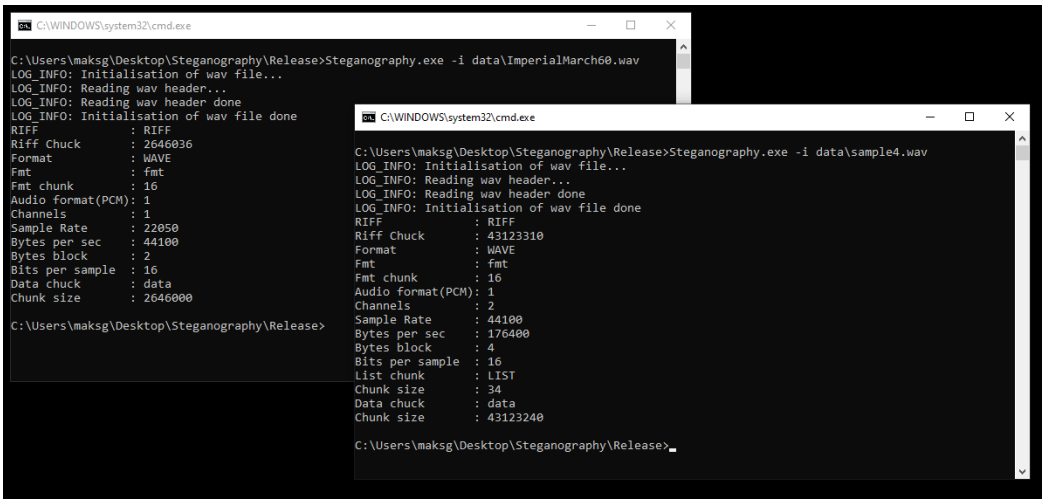

Рисунок 4.10 – Інформація з заголовків файлів

Приховаємо файл в контейнері з ключем 123321 (Рисунок 4.11).

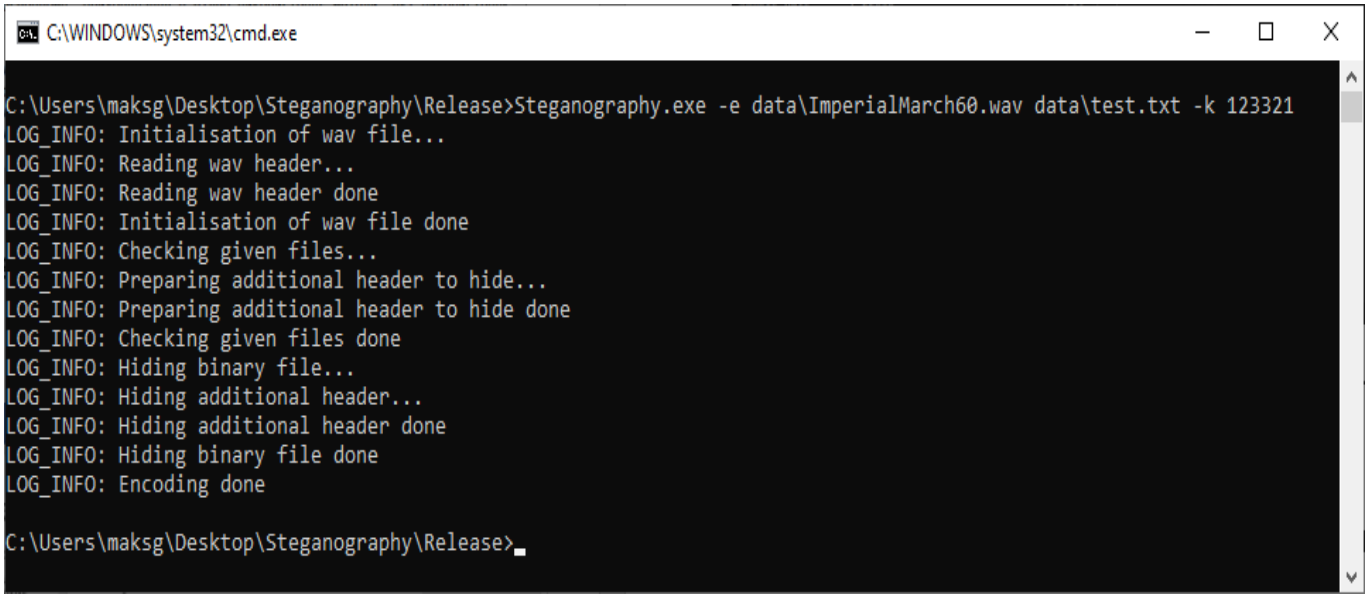

Рисунок 4.11 – Приховування секретного файлу в контейнері

Бачимо результат виконання стеганографічних перетворень – створено файл “output_file.wav” (Рисунок 4.12).

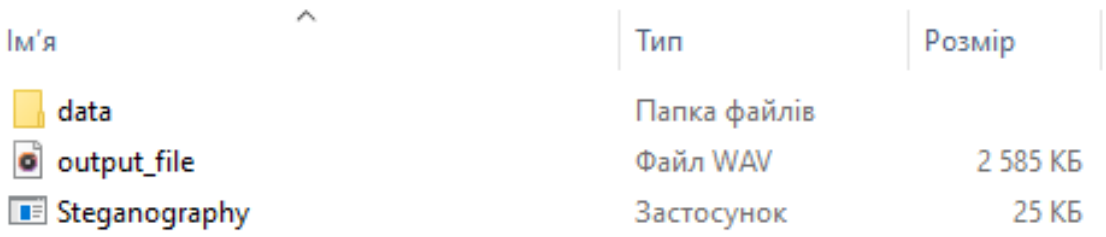

Рисунок 4.12 – Результат виконання приховування файлу

Тепер витягнемо прихований файл та перевіримо коректність даних в ньому після приховування та витягання з контейнеру (Рисунок 4.13).

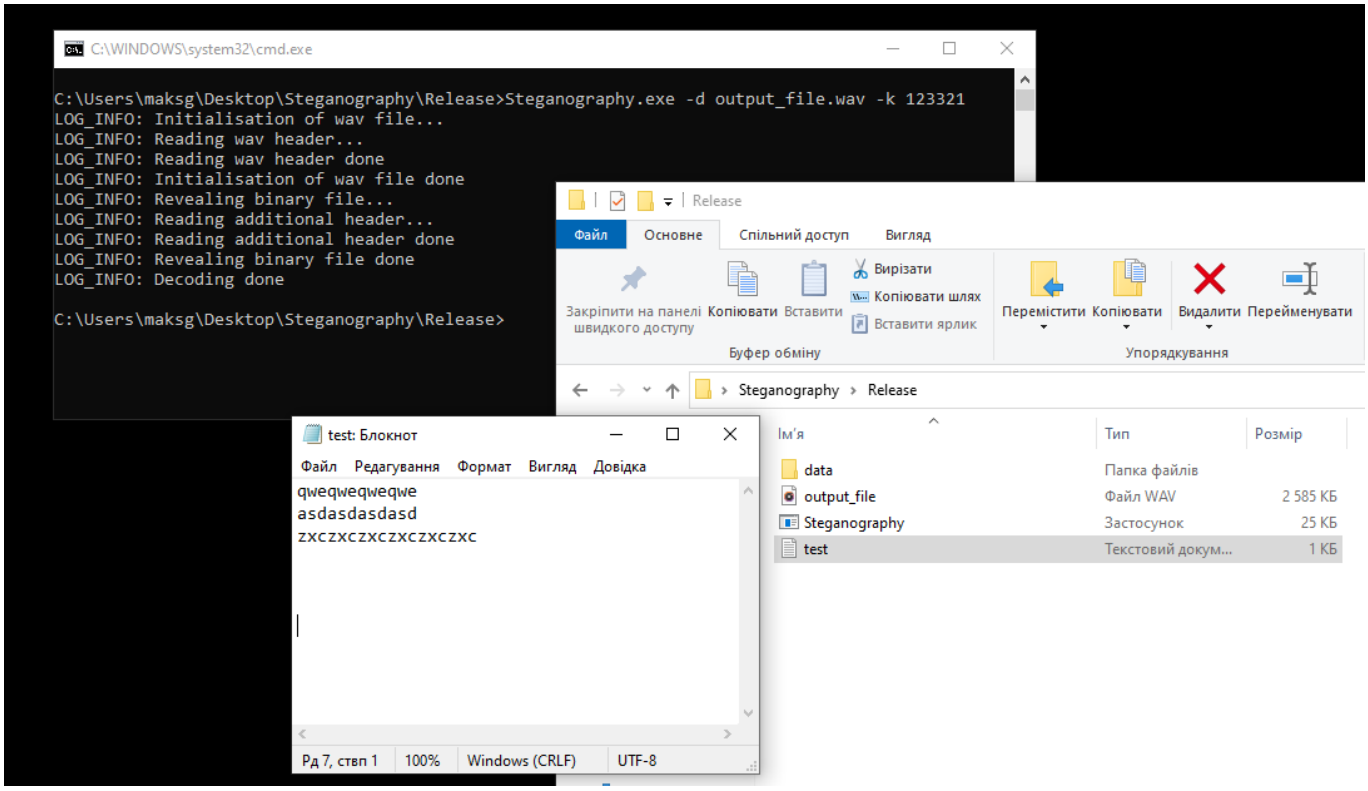

Рисунок 4.13 – Результат витягання прихованого файлу з контейнера

Тепер перевіримо наскільки спотворений вихідний контейнер відносно пустого контейнера та чи чутно різницю між ними. Для цього використаємо програму Audacity, яка є у вільному доступі. Спочатку прослухаємо пустий стеганоконтейнер та заповнений, щоб переконатись, що на слух різниці немає.

Далі порівнюємо амплітудні коливання обох контейнерів (Рисунок 4.14) і помітної різниці також не видно.

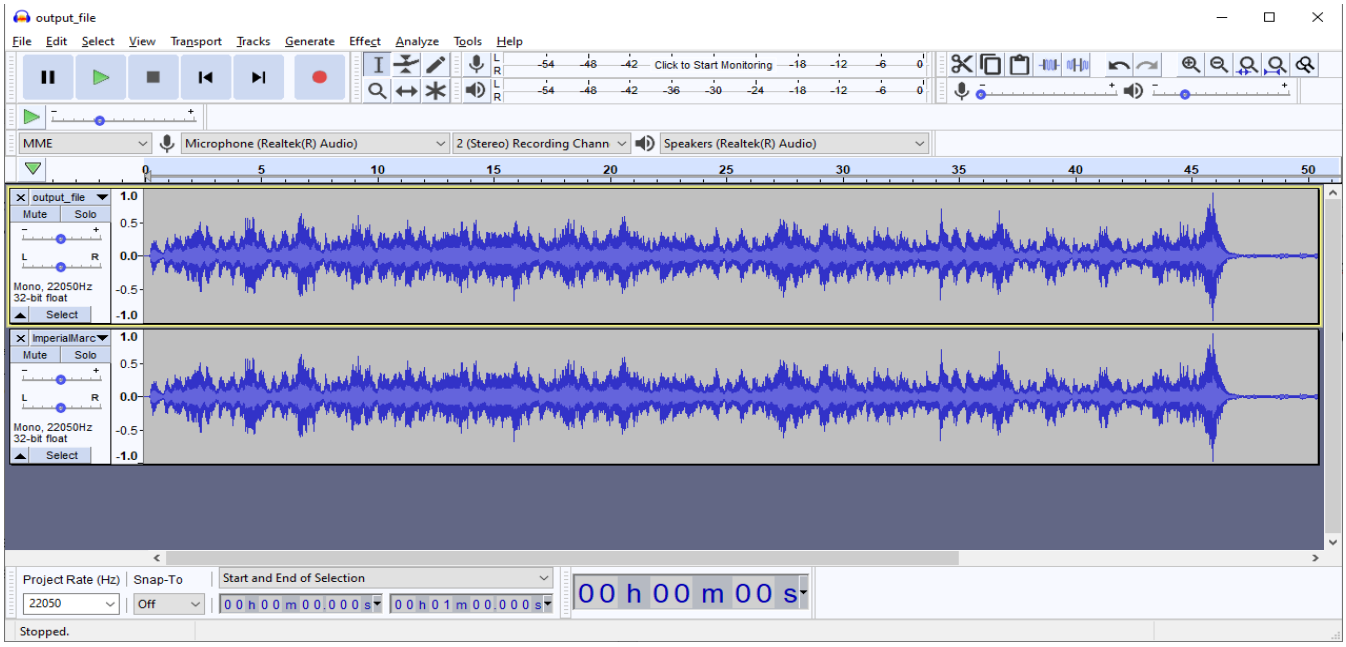

Рисунок 4.14 – Візуальне порівняння амплітудних коливань

Для подальшого тесту потрібно зробити декілька маніпуляцій з обома контейнерами в програмі Audacity. Спочатку інвертуємо один з контейнерів. Потім робимо з них мікс та рендеримо. Результат видно на рисунку 4.15. Такий метод використовується для порівнянь амплітудних коливань аудіофайлів. Якщо програмний продукт відпрацював корректно – при запуску такого результуючого аудіофайлу – користувач почує тільки тишу і ніякого фонового шуму.

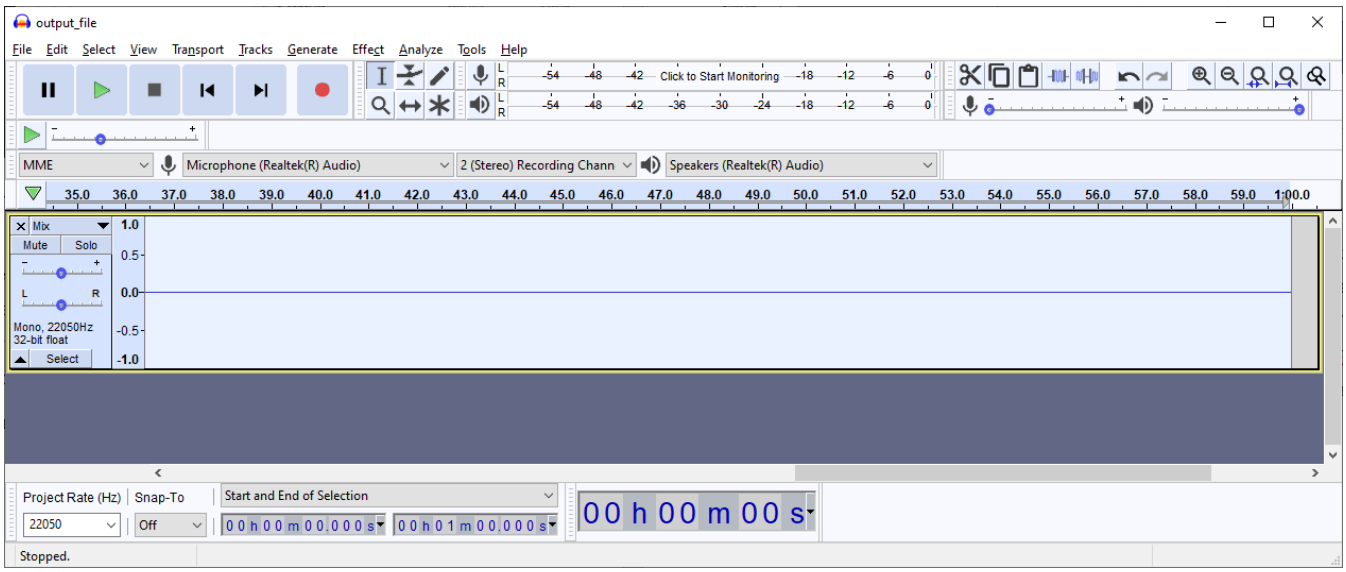

Рисунок 4.15 – Порівняння амплітудних коливань

Висновоки до розділу 4

У цій частині було розроблено програмний продукт для вирішення поставленої задачі. В ході виконання роботи було проаналізовано інструменти розробника, які є доступними на операційній системі Windows 10.

По завершенню аналізу доступних інструментів для розробки програмного продукту, було вирішено використовувати інтегроване середовище програмування Microsoft Visual Studio 2019 для створення проекту програмного продукту, компіляції коду та компонування.

Було здійснено та описано порівняльний аналіз трьох мов програмування високого рівня, а саме: C#, Java, C++. На основі порівнянь було визначено, що для даної постановки задачі найоптимальнішою мовою програмування для реалізації програмного продукту є – мова програмування C++.

Через особисті вподобання було вирішено не користуватись текстовим редактором, який є наявний в інтегрованому середовищі програмування Microsoft Visual Studio 2019, а використовувати інший продукт: Microsoft Visual Code.

Описано модель роботи програмного продукту при приховуванні секретного файлу в пустому контейнері та при витягуванні серкетного файлу з контейнера.

Було наведено переваги даного програмного продукту над іншими, проведено демонстрацію коректної роботи програмного продукту та порівняльний аналіз пустого та заповненого контейнерів.

ВИСНОВКИ

В результаті написання дипломної роботи та виконання дипломного завдання розроблено вдосконалений алгоритм найменш значущого біта (LSB). Мовою програмування для реалізації програмної частини була вибрана мова програмування високого рівня C++.

В поснювальній записці до дипломної роботи проведено аналіз предметної області, розглянуто доступні методи стеганографії та аудіостеганографії, проаналізовано переваги та недоліки обраного підходу та аналогів. Проаналізовано основні зміни для зменшення файлу контейнера та підвищення стійкості контейнера до атак. На основі цього аналізу було вирішено розробити метод модифікації найменш значущого бітового методу, оскільки цей метод забезпечує більш високий рівень безпеки та є ефективним методом, який може приховати секретну інформацію перед хакерами та ефективно захистити її підчас передавання секретної інформації у місце призначення непоміченим способом.

Більше того, цей метод гарантує, що розмір файлу не зміниться навіть після кодування. Порівняно з іншими алгоритмами, це також дозволяє приховувати більш конфіденційну інформацію у файлі контейнера.

У записці детально роз’яснено алгоритм за допомогою якого працює розроблений програмний продукт, а також містяться вказівки щодо коректного використання продукту користувачем даної системи.

З огляду на вищесказане, можна зробити висновок, що завдяки цій роботі мета була досягнута.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мельник С.В. Світові тенденції розвитку цифрової стеганографії в контексті завдань за-безпечення інформаційної безпеки держави / С.В.Мельник, С.В.Кондакова // Актуальні проблеми управління інформаційною безпекою держави : зб. матер. наук.-практ. конф. – К. : Наук.-вид. відділ НА СБ України, 2010.

2. Конахович Г.Ф. Компьютерная стегано-графия. Теория и практика / Г.Ф.Конахович, А.Ю.Пузыренко. – К. : МК-Пресс, 2006.

3. Грибунин В.Г. Цифровая стеганография / В.Г.Грибунин, И.Н.Оков, И.В.Туринцев. – М. : СОЛОН-Пресс, 2002.

4. Быков С.Ф., Мотуз О.В. Основы стегоанализа.// Защита информации. Конфидент. – СПб.: 2000, № 3.

5. Елтышева Е.Ю., Фионов А.Н. Построение стегосистемы на базе растровых изображений с учетом статистики младших бит // Вестник СибГУТИ. – 2009. № 1.

6. Жилкин М. Ю. Стегоанализ графических данных на основе методов сжатия // Вестник СибГУТИ. – 2008. № 2.

7. Кувшинов С.С. Методы и алгоритмы сокрытия больших объемов данных на основе стеганографии / Диссертация на соискание ученой степени кандидата технических наук. – Санкт-Петербург. 2010.

8. Бернет С., Пейн С.: Криптография. Официальное руководство RSA Security – М. «Бином», 2012.

9. Венбо Мао Современная криптография: теория и практика = Modern Cryptography: Theory and Practice. – М.: «Вильямс», 2005.

10. Воробьев В.И., Грибунин В.Г. Теория и практика вейвлет-преобразования. – СПб: ВУС, 2009.

11. Зима В.: Безопасность глобальных сетевых технологий – «БХВ-Петербург», 2011.

12. Нильс Фергюсон, Брюс Шнайер Практическая криптография : Practical Cryptography: Designing and Implementing Secure Cryptographic Systems. – М.: «Диалектика», 2012.

13. Павлов К.А Компьютерная безопасность. Криптографические методы защиты. ДМК Москва, 2010.

14. Ростовцев А.Г. , Михайлова Н.В. Методи криптоаналізу класичних шифрів. – К.: «Наука», 2012.

15. Саломан А. Криптографія з відкритим ключем. – К.: «Наука», 2013.

16. Серов Р.Е., Гончаров В.В., Основы современной криптографии – Москва, Горячая линия – Телеком, 2011.

17. Столлингс В. Криптография и защита сетей: теория и практика. М: Вильямс. 2001.

18. Чмора А.Л. Сучасна прикладна криптографія. 2-е вид., Стер. – М.: Геліос АРВ, 2012.

19. Шнайер, Брюс. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си – М.: Издательство ТРИУМФ, 2002.

20. Mallat S. A Theory For Multiresolution Signal Decomposition: The Wavelet Representation / / IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989. – Vol. 11.

21. Shapiro J. Embedded Image Coding Using Zerotrees Of Wavelet Coefficients / / IEEE Transactions on Signal Processing, 1993.

22. Said A., Pearlman W. A New Fast And Efficient Image Codec Based On Set Partitioning in Hierarchical Trees / / IEEE Transactions on Circuits and Systems for Video Technology, 1996.

23. Antonini M., Barlaud M., Mathieu P., Daubechies I. Image Coding Using Wavelet transform / / IEEE Transactions On Image Processing, 1992.

24. Аграновский А.В. Основы компьютерной стеганографии / А.В. Аграновский, П.Н. Девянин, Р.А. Хади, А.В. Черемушкин. – М: Радио и связь, 2003.

25. Кошкина Н.В. Стеганоаналіз цифрових зображень із застосуванням контрольного вкраплення // Матеріали з Міжнар. наук.-техн. конф. «Захист інформації і безпека інформаційних систем», 5–6 черв. 2014.

26. Стеганоанализ изображений в формате jpeg на базе атаки контрольным внедрением / Н.В. Кошкина // Управляющие системы и машины. — 2014.

27. Кошкина Н.В. Стеганоанализ бесключевых стеганосистем на основе атаки контрольным внедрением / Н.В. Кошкина // Междунар. научно-техн. журнал «Проблемы управления и информатики». – 2014.

28. Ахмад Х.М. Введение в цифровую обработку речевых сигналов / Х.М. Ахмад, В.Ф. Жирков. – Владимир: Издво Владим. гос. ун-та, 2007.

29. Кошкина Н.В. Обзор и классификация методов стеганоанализа / Н.В. Кошкина // УСИМ. – 2015.

30. Кошкіна Н.В. Методи стеганоаналізу з навчанням та класифікацією за характеристичними векторами / Н.В. Кошкіна // Праці міжнар. конф. “Питання оптимізації обчислень-XL”. – Київ: Ін-т кібернетики ім. В.М. Глушкова НАН України. – 2015.

31. Капуста А.М. Методы статистической классификации в задаче обнаружения встраивания информации / А.М. Капуста // Сб. работ 68-й науч. конф. студентов и аспирантов Белорусского гос. ун-та 16-19 мая 2011 г.: в 3-х ч.: ч. 1. – Минск, 2011.

32. Защелкин К.В. Решение проблемы классификации блоков контейнера при jpeg-атаке на стеганографический метод Бенгама-Мемона-Эо-Юнг / К.В. 91 Защелкин, А.А. Ищенко, Е.Н. Иванова // Радіоелектронні і комп’ютерні системи. – 2014.

33. Zadiraka V. Spectral methods of computer steganography problem decision / V. Zadiraka, N. Koshkina // Methods of effective protection of information flows /ed. by V. Zadiraka, Y. Nykolaichuk. – Ternopil: Ternograf, 2014.

34. Конахович Г.Ф. Компьютерная стеганография. Теория и практика / Г.Ф. Конахович, А.Ю. Пузыренко. – К.: МК-Пресс, 2006.

35. Кошкіна Н.В. Інформаційно-теоретична модель безпеки стеганографічних систем / Н.В. Кошкіна // Поступ в науку. – 2011.

36. Хорошко В.А. Введение в компьютерную стеганографию / В.А. Хорошко, М.Е. Шелест. – Киев: Національний Авіаційний Університет, 2002.

37. Задирака В.К. К вопросу стойкости стеганосистемы при пассивных атаках / В.К. Задирака, Л.Л. Никитенко // Междунар. научно-техн. журнал «Проблемы управления и информатики». – 2009.

38. Кошкіна Н.В. Ефективні спектральні алгоритми для вирішення задач цифрової стеганографії: дис. … канд. фіз.-мат. наук: 01.05.01 / Н.В. Кошкіна. – Київ, 2005.

39. Клопов В.А. Основы компьютерной стеганографии / В.А. Клопов, О.В. Мотуз // Конфидент. – 1997.

40. Кустов В.Н. Методы встраивания скрытых сообщений / В.Н. Кустов, А.А. Федчук // Конфидент. – 2000.

41. Задірака В.К. Спектральні алгоритми комп'ютерної стеганографії / В.К. Задірака, С.С. Мельнікова, Н.В. Бородавка //Искусственный интеллект. – 2002.

42. Швидченко И.В. Методы стеганоанализа для графических файлов / И.В. Швидченко // Искусственный интеллект. – 2010.

43. Швідченко І.В. Аналіз програмного забезпечення зі стеганоаналізу / І.В. Швідченко //Искусственный интеллект. – 2012.

44. Li F. JPEG steganalysis with high-dimensional features and bayesian ensemble classifier / F. Li, X. Zhang, B. Chen, G. Feng //IEEE signal processing letters. – 2013.

45. Яне Б. Цифровая обработка изображений / Б. Яне. – М.: Техносфера, 2007.

46. Вовк О.О. Сравнительный анализ устойчивости к атакам стеганографических методов скрытия информации / О.О. Вовк, А.А. Астраханцев // Мат. 9-й Межд. мол. научно-техн. конф. «Современные проблемы радиотехники и телекоммуникаций РТ-2013». – 2013.

47. Кошкина Н.В. О методе защиты интеллектуальной собственности на основе выделения точечных особенностей изображения / Н.В. Кошкина //Захист інформації. – 2007.

48. Avcibas I. Image steganalysis with binary similarity measures / I. Avcibas¸ M. Kharrazi, N.D. Memon, B. Sankur //EURASIP Journal on Applied Signal Processing. – 2005.

49. Кошкіна Н.В. Стійкі до активних атак методи комп'ютерної стеганографії / Н.В. Кошкіна // Вісн. НАН України. – 2013.

50. Задирака В.К. Статистический анализ систем с цифровыми водяными знаками / В.К. Задирака, Н.В. Кошкина, Л.Л. Никитенко // Искусственный интеллект.– 2008.

51. Мелешко Е.В. Метод встраивания двухуровневых цифровых водяных знаков в медиафайлы для защиты авторских прав / Е.В. Мелешко // Збірник наукових праць Харківського університету Повітряних Сил. – 2013.

52. Suresh A. Image Texture Classification using Gray Level Co-Occurrence Matrix Based Statistical Features / A. Suresh, K.L. Shunmuganathan // European Journal of Scientific Research. – 2012.

53. Voloshynovskiy S.V. Visual communications with side information via distributed printing channels: extended multimedia and security perspectives / S.V. Voloshynovskiy, O. Koval, F. Deguillaume, T. Pun // Proc. of SPIE: Security, Steganography, and Watermarking of Multimedia Contents VI, San Jose, USA, January 2004.

54. Lin C.-Y. Distortion Modeling and Invariant Extraction for Digital Image PrintandScan Process [Електронний ресурс] / C.-Y. Lin, S.-F. Chang // Intl. Symp. on Multimedia Information Processing, Taipei, December 1999.

55. Терещенко А.Н. Реализация операции умножения с использованием преобразования Уолша / А.Н. Терещенко, С.С. Мельникова, Л.А. Гнатив, В.К. Задирака, Н.В. Кошкина // Междунар. научно-техн. журнал «Проблемы управления и информатики». – 2010.

56. Задирака В.К. К вопросу стойкости стеганосистемы при пассивных атаках / В.К. Задирака, Л.Л. Никитенко // Междунар. научно-техн. журнал «Проблемы управления и информатики». – 2009.

57. Гнатив Л.А. Методы синтеза эффективных ортогональных преобразований высокой и низкой корреляции и их быстрых алгоритмов для кодирования и сжатия цифровых изображений / Л.А. Гнатив, Е.С. Шевчук //Кибернетика и системный анализ. – 2002.

58. Yang S. Quantization-Based Digital Audio Watermar-king in Discrete Fourier Transform Domain / S. Yang, W. Tan, Y. Chen, W. Ma // Journal of Multimedia. – 2010.

59. Поліновський В.В. Інформаційна технологія для досліджень методів стеганографії і стеганоаналізу / В.В. Поліновський, В.Ю. Корольов, В.А. Герасименко, М.Л. Гоpинштейн // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. – 2011.

60. Manjula Devi T.H. Detecting original image using histogram, DFT and SVM / T.H. Manjula Devi, H.S. Manjunatha Reddy, K.B. Raja, K.R. Venugopal, L.M. Patnaik //Intern. journal of recent trends in engineering. – 2009.

61. Sheikhan M. Blind image steganalysis via joint co-occurrence matrix and statistical moments of contourlet transform / M. Sheikhan, M.S. Moin, M. Pezhmanpour //10th Int. Conf. on Intelligent Systems Design and Applications. – 2010.

62. Yang X. Universal image steganalysis based on wavelet packet decomposition and empirical transition matrix in wavelet domain / X. Yang, Y. Lei, X. Pan, J. Liu // International forum on computer science-technology and applications. – 2009.

63. Кошкина Н.В. Защита космических и астрономических изображений методами компьютерной стеганографии / Н.В. Кошкина, О.Ю. Никитина // Праці IV міжнар. наук.-техн. конф. "Гіротехнології, навігація, керування рухом і конструювання авіаційно-космічної техніки", Ч. 2. – Київ: НТУУ «КПІ». – 2007.

64. Кошкіна Н.В. До питання часо-частотного аналізу сигналів в задачах комп'ютерної стеганографії / Н.В. Кошкіна // Праці міжнар. конф. "Питання оптимізації обчислень-XXXVI. Київ: Ін-т кібернетики ім. В.М. Глушкова НАН України. – 2011.

65. Barni M. A DWT-based technique for spatio-frequency masking of digital signatures / M. Barni, F. Bartolini, V. Cappellini, A. Lippi, A. Piva // Proc. of the 11th SPIE Annual Symposium, Electronic Imaging, Security and Watermarking of Multimedia Contents. – 1999.

66. Защелкин К.В. Решение проблемы классификации блоков контейнера при jpegатаке на стеганографический метод Бенгама-Мемона-Эо-Юнг / К.В. Защелкин, А.А. Ищенко, Е.Н. Иванова // Радіоелектронні і комп'ютерні системи. – 2014.

67. Вовк О.О. Сравнительный анализ устойчивости к атакам стеганографических методов скрытия информации / О.О. Вовк, А.А. Астраханцев // Мат. 9-й Межд. мол. научно-техн. конф. «Современные проблемы радиотехники и телекоммуникаций РТ-2013». – 2013.

68. Simitopoulos D. Robust Image Watermarking Based on Generalized Radon Transformations / D. Simitopoulos, D.E. Koutsonanos, M.G. Strintzis // Circuits and Systems for Video Technology. – 2003.

69. Chiu Y.-C. Copyright Protection against Print-and-Scan Operations by Watermarking for Color Images Using Coding and Synchronization of Peak Locations in Frequency Domain / Y.-C. Chiu, W.-H. Tsai // Journal of Information Science and Engineering. – 2006.

ДОДАТОК А

Лістинг програми

main.cpp

```cpp
#include "header.hpp"
#include "WavFile.hpp"
#include "myFile.hpp"
#include "logger.hpp"

void usage(void);
int wavInfoHandler(char *);
int wavEncoderHandler(char *, char *, uint16_t);
int wavEncoderHandler(char *, char *, char *, uint16_t);
int wavDecoderHandler(char *, uint16_t);

int main(int argc, char* argv[]) {

    switch (argc)
    {
    case 1:
    case 2:
        usage();
        break;

    case 3:
        if (strcmp(argv[1], "-i") == 0)
        {
            wavInfoHandler(argv[2]);
        }
        else
        {
            usage();
            break;
        }
        break;

    case 4:
        usage();
        break;
    case 5:
        if (strcmp(argv[1], "-d") == 0 && strcmp(argv[3], "-k") == 0)
        {
            uint16_t key = (uint16_t)atoi(argv[4]);
            if (key < 1 || key > 65000)
            {
                LOG(LOG_ERR, "Key must be [1; 65 000]");
                usage();
                break;
            }
            wavDecoderHandler(argv[2], key);
        }
        else
        {
```

main.cpp

main.cpp > ...

```
46              wavDecoderHandler(argv[2], key);
47          }
48          else
49          {
50              usage();
51              break;
52          }
53          break;
54      case 6:
55          if (strcmp(argv[1], "-e") == 0 && strcmp(argv[4], "-k") == 0)
56          {
57              uint16_t key = (uint16_t)atoi(argv[5]);
58              if (key < 1 || key > 65000)
59              {
60                  LOG(LOG_ERR, "Key must be [1; 65 000]");
61                  usage();
62                  break;
63              }
64              wavEncoderHandler(argv[2], argv[3], key);
65          }
66          else
67          {
68              usage();
69              break;
70          }
71          break;
72      case 7:
73          if (strcmp(argv[1], "-e") == 0 && strcmp(argv[5], "-k") == 0)
74          {
75              uint16_t key = (uint16_t)atoi(argv[6]);
76              if (key < 1 || key > 65000)
77              {
78                  LOG(LOG_ERR, "Key must be [1; 65 000]");
79                  usage();
80                  break;
81              }
82              wavEncoderHandler(argv[2], argv[3], argv[4], key);
83          }
84          else
85          {
86              usage();
87              break;
88          }
89          break;
90      default:
91          usage();
92      }
93
94      return 0;
```

main.cpp - Steganography - Visual Studio Code

main.cpp

main.cpp > ...

```
93
94      return 0;
95  }
96
97  void usage(void)
98  {
99      std::cout << "usage  : wav -<option> <argumets>\n" <<
100             "Options:\n" <<
101             "-i     : wav file's information\n" <<
102             "-e     : hide binary file into wav file\n" <<
103             "-d     : unhide binary file from wav file\n" <<
104             "-k     : key for operations. Available range is [1; 65 000]\n" <<
105             "Example: \n" <<
106             "wav -i <wav_file>\n" <<
107             "wav -e <input_wav_file> <input_binary_file> <output_wav_file(optional)> -k <key>\n" <<
108             "wav -d <input_wav_file> -k <key>\n" <<
109     std::endl;
110 }
111
112 //function definition
113 int wavInfoHandler(char* wavFilePath)
114 {
115     WavFile wavFile(wavFilePath, 1);
116
117     wavFile.printFileInfo();
118
119     return 0;
120 };
121
122 int wavEncoderHandler(char *wavFilePath, char *wavBinaryPath, uint16_t key)
123 {
124     return wavEncoderHandler(wavFilePath, wavBinaryPath, (char *)"output_file.wav", key);
125 }
126
127 int wavEncoderHandler(char* wavFilePath, char* wavBinaryPath, char* outFilePath, uint16_t key)
128 {
129     WavFile wavFile(wavFilePath, key);
130     if (wavFile.encryptFile(wavFilePath, wavBinaryPath, outFilePath) != WAV_SUCCESS)
131     {
132         LOG(LOG_WARN, "Encoding failed");
133         return -1;
134     }
135
136     LOG(LOG_INF, "Encoding done");
137     return 0;
138 };
139
140 int wavDecoderHandler(char* wavFilePath, uint16_t key)
```

```
138 };
139
140 int wavDecoderHandler(char* wavFilePath, uint16_t key)
141 {
142     WavFile wavFile(wavFilePath, key);
143     if (wavFile.decryptFile(wavFilePath) != WAV_SUCCESS)
144     {
145         LOG(LOG_WARN, "Decoding failed");
146         return -1;
147     }
148
149     LOG(LOG_INF, "Decoding done");
150     return 0;
151 };
152
```

myFile.cpp

```cpp
#include "myFile.hpp"
#include "logger.hpp"

myFile::myFile(const char *filePath, Mode mode): internalError(false), name(0), path(0), size(0)
{
    if (!myFile::ifFileExist(filePath))
    {
        char buff[100] = { 0 };

        snprintf(buff, sizeof(buff), "Cannot create myFile obj correctly. File \"%s\" doesn't exist", filePath);
        LOG(LOG_ERR, buff);

        internalError = true;
    }
    else
    {
        FILE *file;

        path = (char *)malloc(sizeof(char) * (strlen(filePath) + 1));
        strcpy(path, filePath);

        // +1 position to avoid '\' at the start of fileName
        name = strrchr(path, '\\') + 1;

        // +1 position to avoid '.' at the start of fileName
        ext = strrchr(path, '.') + 1;

        if (mode == MODE_READ)
            file = fopen(path, "rb");
        else
            file = fopen(path, "wb");

        fseek(file, 0L, SEEK_END);
        size = ftell(file);
        fseek(file, 0L, SEEK_SET);

        fclose(file);
    }
}

myFile::~myFile()
{
    free(path);
}

char * myFile::getExt() const
{
    if (internalError)
    {
        LOG(LOG_ERR, "InternalError");
    }

    return ext;
}

char * myFile::getName() const
{
    if (internalError)
    {
        LOG(LOG_ERR, "InternalError");
    }

    return name;
}
```

myFile.cpp

myFile.cpp > ...

```
62
63      return name;
64  }
65
66  char * myFile::getPath() const
67  {
68      if (internalError)
69      {
70          LOG(LOG_ERR, "InternalError");
71      }
72
73      return path;
74  }
75
76  uint32_t myFile::getSize() const
77  {
78      if (internalError)
79      {
80          LOG(LOG_ERR, "InternalError");
81      }
82
83      return size;
84  }
85
86
87  bool myFile::ifFileExist(const char *filePath)
88  {
89      FILE* fileToCheck = NULL;
90
91      if ((fileToCheck = fopen(filePath, "rb")) == NULL) {
92          return false;
93      }
94      fclose(fileToCheck);
95
96      return true;
97  }
98
99  uint32_t myFile::getFileSize(const char *filePath)
100 {
101     FILE* fileToCheck = NULL;
102     uint32_t fileToCheckSize = 0;
103
104     if (myFile::ifFileExist(filePath))
105     {
106         fileToCheck = fopen(filePath, "rb");
107
108         fseek(fileToCheck, 0L, SEEK_END);
109         fileToCheckSize = ftell(fileToCheck);
110         fseek(fileToCheck, 0L, SEEK_SET);
```

myFile.hpp

myFile.hpp > myFile > getFileSize(const char *)

```cpp
#pragma once

#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <stdbool.h>
#include <string>

enum Mode
{
    MODE_READ = 0,
    MODE_WRITE
};

class myFile
{
private:
    bool internalError;
    char *ext;
    char *name;
    char *path;
    uint32_t size;
public:
    myFile(const char *filePath, Mode mode);
    ~myFile();

    char * getExt() const;
    char * getName() const;
    char * getPath() const;
    uint32_t getSize() const;

    static bool ifFileExist(const char *filePath);
    static uint32_t getFileSize(const char *filePath);
};
```

logger.hpp

```hpp
#pragma once

#include <iostream>
#include <string>

#define LOG_INF  "LOG_INFO"
#define LOG_WARN "LOG_WARNING"
#define LOG_ERR  "LOG_ERROR"

#define __FILENAME__ (strrchr(__FILE__, '\\') ? strrchr(__FILE__, '\\') + 1 : __FILE__)

#ifdef _DEBUG
    #define LOG(LOG_TYPE, MESSAGE)                                  \
        do                                                          \
        {                                                           \
            std::cout <<                                            \
                LOG_TYPE << ": [" << __FILENAME__ << " : " <<       \
                __FUNCTION__ << "() : " << __LINE__ << "] : " << \
                MESSAGE <<                                          \
            std::endl;                                              \
        } while(0);                                                 \

#else
    #define LOG(LOG_TYPE, MESSAGE)                                  \
        do                                                          \
        {                                                           \
            std::cout <<                                            \
                LOG_TYPE << ": " << MESSAGE <<                      \
            std::endl;                                              \
        } while(0);                                                 \

#endif
```

lfsr.hpp

```hpp
#pragma once

#include <cstdint>

class LFSR
{
public:
    LFSR(uint16_t);
    ~LFSR();

    char getByte(char);

private:
    uint16_t lfsr;

};
```

lfsr.cpp

```cpp
#include "lfsr.hpp"

LFSR::LFSR(uint16_t pass): lfsr(0)
{
    lfsr = pass;
}

LFSR::~LFSR()
{
}
#include <iostream>
char LFSR::getByte(char byte)
{
    lfsr = ((( (lfsr>>0)^(lfsr>>2)^(lfsr>>11) ) & 1 ) << 11 ) | (lfsr>>1);
    byte ^= (char)(lfsr & 1);

    return byte;
}

```

WavFile.hpp - Steganography - Visual Studio Code

WavFile.hpp

```cpp
#pragma once

#include <cstdint>
#include <stdbool.h>
#include "lfsr.hpp"

#define WAV_SUCCESS 0
#define WAV_ERROR 1
#define WAV_HEADER_SIZE 44

struct s_wavHeader {
    char ChunkID[4];
    uint32_t ChunkSize;
    char Format[4];
    char Subchunk1ID[4];
    uint32_t Subchunk1Size;
    uint16_t AudioFormat;
    uint16_t NumChannels;
    uint32_t SampleRate;
    uint32_t ByteRate;
    uint16_t BlockAlign;
    uint16_t BitsPerSample;
    uint32_t listSize; //if it present in header - add +8 bytes to skip whole LIST subchunk
    char Subchunk2ID[4];
    uint32_t Subchunk2Size;
};

```

WavFile.hpp

```
    uint16_t BitsPerSample;
    uint32_t listSize; //if it present in header - add +8 bytes to skip whole LIST subchunk
    char Subchunk2ID[4];
    uint32_t Subchunk2Size;
};

struct s_myHeader {
    uint32_t fileSize; // in bytes
    uint32_t nameSize; // in bytes
    char *name;

    // Doesn't write into file.
    // Need just for calculates while hiding.
    // (sizeof(fileSize) + sizeof(nameSize) + nameSize)
    uint32_t headerSize;
};

class WavFile {
private:
    struct s_wavHeader wavHeader = { 0 };
    struct s_myHeader myHeader = { 0 };

    bool internalError;

    uint32_t possibleBytesToHideCount;
    uint32_t step;
    LFSR *lfsr; // linear feedback shift register

    char readHiddenByte(FILE **file);
    void hideByte(FILE **fileContainer, FILE **fileResult, char byte);
    void prepareHeader(char* childfile);
    void writeHeader(FILE **parFile, FILE **outFile);
    void readHeader(FILE **parFile);
    int readWavHeader(const char* filename);                                    // read WAVE Header
    int checkFilesForHiding(char* parentfile, char* childfile);     // check files
public:
    WavFile(const char* filename, uint16_t key);
    ~WavFile();

    void printFileInfo();                                           // print header variables
    int encryptFile(char* parentfile, char* childfile, char* outputfile);
    int decryptFile(char* encFilePath);
};

```

WavFile.cpp

WavFile.cpp > WavFile::WavFile(const char *, uint16_t)

```cpp
#include <iostream>
#include <cstdlib>
#include <cstring>
#include "WavFile.hpp"
#include "myFile.hpp"
#include "logger.hpp"

WavFile::WavFile(const char* filePath, uint16_t key): internalError(false), possibleBytesToHideCount(0), step(0), lfsr(NULL)
{
    LOG(LOG_INF, "Initialisation of wav file...");
    memset(&wavHeader, 0, sizeof(s_wavHeader));
    memset(&myHeader, 0, sizeof(s_myHeader));

    if (!myFile::ifFileExist(filePath))
    {
        char buff[100] = { 0 };

        snprintf(buff, sizeof(buff), "Cannot create WavFile obj correctly. File \"%s\" doesn't exist", filePath);
        LOG(LOG_ERR, buff);

        internalError = true;
    }
    else if (readWavHeader(filePath) != WAV_SUCCESS)
    {
        LOG(LOG_ERR, "Cannot create WavFile obj correctly. Internal ERROR in readWavHeader()");

        internalError = true;
    }
    else
    {
        lfsr = new LFSR(key);
        LOG(LOG_INF, "Initialisation of wav file done");
    }
}

WavFile::~WavFile()
{
    free(myHeader.name);
    delete lfsr;
}

int WavFile::readWavHeader(const char* wavFileName)
{
    LOG(LOG_INF, "Reading wav header...");
    int res = WAV_ERROR;

    FILE* wavFile = fopen(wavFileName, "rb");
```

WavFile.cpp - Steganography - Visual Studio Code

WavFile.cpp X

WavFile.cpp > WavFile::readWavHeader(const char *)

```
}

int WavFile::readWavHeader(const char* wavFileName)
{
    LOG(LOG_INF, "Reading wav header...");
    int res = WAV_ERROR;

    FILE* wavFile = fopen(wavFileName, "rb");

    if (fread((char*)&(wavHeader.ChunkID), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint32_t*)&(wavHeader.ChunkSize), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((char*)&(wavHeader.Format), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((char*)&(wavHeader.Subchunk1ID), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint32_t*)&(wavHeader.Subchunk1Size), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint16_t*)&(wavHeader.AudioFormat), 2, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint16_t*)&(wavHeader.NumChannels), 2, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint32_t*)&(wavHeader.SampleRate), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint32_t*)&(wavHeader.ByteRate), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint16_t*)&(wavHeader.BlockAlign), 2, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint16_t*)&(wavHeader.BitsPerSample), 2, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((char*)&(wavHeader.Subchunk2ID), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
    if (fread((uint32_t*)&(wavHeader.Subchunk2Size), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};

    if (!strncmp(wavHeader.Subchunk2ID, "LIST", 4))
    {
        wavHeader.listSize = wavHeader.Subchunk2Size;
        fseek(wavFile, wavHeader.listSize, SEEK_CUR);
        wavHeader.listSize += 8; //adding lost SubChunkID and SubChunkSize size

        if (fread((char*)&(wavHeader.Subchunk2ID), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};
        if (fread((uint32_t*)&(wavHeader.Subchunk2Size), 4, 1, wavFile) != 1 || ferror(wavFile)) {LOG(LOG_WARN, "qwe"); goto error;};

        if (strncmp(wavHeader.Subchunk2ID, "data", 4))
        {
            wavHeader.Subchunk2Size = 0;
            char buff[100] = { 0 };

            snprintf(buff, sizeof(buff), "Unknown DATA subchunk(%.4s)", wavHeader.Subchunk2ID);
            LOG(LOG_ERR, buff);

            internalError = true;

            res = WAV_ERROR;
        }
    }

    res = WAV_SUCCESS;
    fclose(wavFile);
    LOG(LOG_INF, "Reading wav header done");

```

WavFile.cpp X

WavFile.cpp > WavFile::readWavHeader(const char *)

```
87      fclose(wavFile);
88      LOG(LOG_INF, "Reading wav header done");
89
90      return res;
91
92  error:
93      LOG(LOG_ERR, "InternalError while reading .wav header");
94      perror("WavFile::readWavHeader() ERROR");
95      internalError = true;
96      fclose(wavFile);
97
98      return WAV_ERROR;
99  }
100
101 void WavFile::printFileInfo()
102 {
103     if (internalError)
104     {
105         LOG(LOG_ERR, "InternalError");
106         return;
107     }
108
109     std::cout << "RIFF            : " << wavHeader.ChunkID[0] << wavHeader.ChunkID[1] << wavHeader.ChunkID[2] << wavHeader.ChunkID[3] << std::endl;
110     std::cout << "Riff Chuck      : " << wavHeader.ChunkSize << std::endl;
111     std::cout << "Format          : " << wavHeader.Format[0] << wavHeader.Format[1] << wavHeader.Format[2] << wavHeader.Format[3] << std::endl;
112     std::cout << "Fmt             : " << wavHeader.Subchunk1ID[0] << wavHeader.Subchunk1ID[1] << wavHeader.Subchunk1ID[2] << wavHeader.Subchunk1ID[3] <<
        std::endl;
113     std::cout << "Fmt chunk       : " << wavHeader.Subchunk1Size << std::endl;
114     std::cout << "Audio format(PCM): " << wavHeader.AudioFormat << std::endl;
115     std::cout << "Channels        : " << wavHeader.NumChannels << std::endl;
116     std::cout << "Sample Rate     : " << wavHeader.SampleRate << std::endl;
117     std::cout << "Bytes per sec   : " << wavHeader.ByteRate << std::endl;
118     std::cout << "Bytes block     : " << wavHeader.BlockAlign << std::endl;
119     std::cout << "Bits per sample  : " << wavHeader.BitsPerSample << std::endl;
120     if (!!wavHeader.listSize)
121     {
122         std::cout << "List chunk      : LIST" << std::endl;
123         std::cout << "Chunk size      : " << wavHeader.listSize << std::endl;
124     }
125     std::cout << "Data chuck      : " << wavHeader.Subchunk2ID[0] << wavHeader.Subchunk2ID[1] << wavHeader.Subchunk2ID[2] << wavHeader.Subchunk2ID[3] <<
        std::endl;
126     std::cout << "Chunk size      : " << wavHeader.Subchunk2Size << std::endl;
127 }
128
129 int WavFile::checkFilesForHiding(char* wavFilePath, char* binFilePath)
130 {
131     LOG(LOG_INF, "Checking given files...");
132     if (internalError)
133     {
```

WavFile.cpp

WavFile.cpp > WavFile::readWavHeader(const char *)

```
130  {
131      LOG(LOG_INF, "Checking given files...");
132      if (internalError)
133      {
134          LOG(LOG_ERR, "InternalError");
135          return WAV_ERROR;
136      }
137
138      FILE* wavFile = NULL, * binFile = NULL;
139      uint32_t wavFileSize = 0, binFileSize = 0;
140
141      if (!myFile::ifFileExist(wavFilePath))
142      {
143          char buff[100] = { 0 };
144
145          snprintf(buff, sizeof(buff), "File check failed. File \"%s\" doesn't exist", wavFilePath);
146          LOG(LOG_ERR, buff);
147
148          internalError = true;
149
150          return WAV_ERROR;
151      }
152
153      if (!myFile::ifFileExist(binFilePath))
154      {
155          char buff[100] = { 0 };
156
157          snprintf(buff, sizeof(buff), "File check failed. File \"%s\" doesn't exist", binFilePath);
158          LOG(LOG_ERR, buff);
159
160          internalError = true;
161
162          return WAV_ERROR;
163      }
164
165      if (strncmp(wavHeader.ChunkID, "RIFF", 4))
166      {
167          LOG(LOG_ERR, "Unknown ChunkID");
168
169          internalError = true;
170
171          return WAV_ERROR;
172      }
173
174      if (strncmp(wavHeader.Format, "WAVE", 4))
175      {
176          LOG(LOG_ERR, "Unknown Format");
177
178          internalError = true;
```

WavFile.cpp

WavFile.cpp > WavFile::readWavHeader(const char *)

```
177
178         internalError = true;
179
180         return WAV_ERROR;
181     }
182
183     if (strncmp(wavHeader.Subchunk1ID, "fmt ", 4))
184     {
185         LOG(LOG_ERR, "Unknown Subchunk1ID");
186
187         internalError = true;
188
189         return WAV_ERROR;
190     }
191
192     if (wavHeader.Subchunk1Size != 16)
193     {
194         LOG(LOG_ERR, "Unknown Subchunk1Size");
195
196         internalError = true;
197
198         return WAV_ERROR;
199     }
200
201     if (wavHeader.AudioFormat != 1)
202     {
203         LOG(LOG_ERR, "Unknown AudioFormat");
204
205         internalError = true;
206
207         return WAV_ERROR;
208     }
209
210     if (strncmp(wavHeader.Subchunk2ID, "data", 4))
211     {
212         LOG(LOG_ERR, "Unknown Subchunk2ID");
213
214         internalError = true;
215
216         return WAV_ERROR;
217     }
218
219     prepareHeader(binFilePath);
220
221     binFileSize = myFile::getFileSize(binFilePath);
222
223     possibleBytesToHideCount = (wavHeader.Subchunk2Size - (myHeader.headerSize * wavHeader.BitsPerSample)) /
224                                wavHeader.BitsPerSample;
225
226     if (possibleBytesToHideCount <= binFileSize)
```

WavFile.cpp

WavFile.cpp > WavFile::readWavHeader(const char *)

```
224                                                     wavHeader.bitsPerSample;
225
226     if (possibleBytesToHideCount <= binFileSize)
227     {
228         char buff[100] = { 0 };
229
230         snprintf(buff, sizeof(buff), "Binary file's size is: (%d) bytes, but you can hide (%d) bytes", binFileSize, possibleBytesToHideCount);
231         LOG(LOG_WARN, "Binary file is too big to hide it into current .wav file");
232         LOG(LOG_WARN, buff);
233
234         internalError = true;
235         return WAV_ERROR;
236     }
237
238     step = possibleBytesToHideCount / (binFileSize + myHeader.headerSize);
239
240     LOG(LOG_INF, "Checking given files done");
241
242     return WAV_SUCCESS;
243 }
244
245 void WavFile::prepareHeader(char* childfile)
246 {
247     LOG(LOG_INF, "Preparing additional header to hide...");
248     if (internalError)
249     {
250         LOG(LOG_ERR, "InternalError");
251         return;
252     }
253
254     myFile temp(childfile, MODE_READ);
255
256     myHeader.fileSize = temp.getSize();
257     myHeader.nameSize = strlen(temp.getName()) + 1;
258
259     myHeader.name = (char *)malloc(myHeader.nameSize);
260     memset(myHeader.name, 0, myHeader.nameSize);
261     strcpy(myHeader.name, temp.getName());
262
263     myHeader.headerSize = sizeof(myHeader.fileSize) + sizeof(myHeader.nameSize) + myHeader.nameSize;
264
265     LOG(LOG_INF, "Preparing additional header to hide done");
266 }
267
268 void WavFile::writeHeader(FILE **parFile, FILE **outFile)
269 {
270     LOG(LOG_INF, "Hiding additional header...");
271     if (internalError)
272     {
273         LOG(LOG_ERR, "InternalError");
```

WavFile.cpp

WavFile.cpp > WavFile::readWavHeader(const char *)

```
272     {
273         LOG(LOG_ERR, "InternalError");
274         return;
275     }
276 
277     for (uint32_t i = 0; i < sizeof(uint32_t); i++)
278     {
279         char *headerByte = (char *)&(myHeader.fileSize) + i;
280         hideByte(parFile, outFile, *headerByte);
281     }
282     for (uint32_t i = 0; i < sizeof(uint32_t); i++)
283     {
284         char *headerByte = (char *)&(myHeader.nameSize) + i;
285         hideByte(parFile, outFile, *headerByte);
286     }
287 
288     for (uint32_t i = 0; i < myHeader.nameSize; i++)
289     {
290         hideByte(parFile, outFile, myHeader.name[i]);
291     }
292 
293     LOG(LOG_INF, "Hiding additional header done");
294 }
295 
296 void WavFile::readHeader(FILE **parFile)
297 {
298     LOG(LOG_INF, "Reading additional header...");
299     if (internalError)
300     {
301         LOG(LOG_ERR, "InternalError");
302         return;
303     }
304 
305     uint32_t temp = 0;
306 
307     for (uint32_t i = 0; i < sizeof(uint32_t); i++)
308     {
309         ((char *)&(myHeader.fileSize))[i] = readHiddenByte(parFile);
310     }
311     for (uint32_t i = 0; i < sizeof(uint32_t); i++)
312     {
313         ((char *)&(myHeader.nameSize))[i] = readHiddenByte(parFile);
314     }
315 
316     myHeader.name = (char *)malloc(myHeader.nameSize);
317     memset(myHeader.name, 0, myHeader.nameSize);
318     for (uint32_t i = 0; i < myHeader.nameSize; i++)
319     {
320         myHeader.name[i] = readHiddenByte(parFile);
```

WavFile.cpp

WavFile.cpp > WavFile::readWavHeader(const char *)

```
319     {
320         myHeader.name[i] = readHiddenByte(parFile);
321     }
322
323     myHeader.headerSize = sizeof(myHeader.fileSize) + sizeof(myHeader.nameSize) + myHeader.nameSize;
324
325     possibleBytesToHideCount = (wavHeader.Subchunk2Size - (myHeader.headerSize * wavHeader.BitsPerSample)) /
326                                                         wavHeader.BitsPerSample;
327
328     if (possibleBytesToHideCount <= myHeader.fileSize || !possibleBytesToHideCount || !myHeader.fileSize)
329     {
330         LOG(LOG_WARN, "InternalError. Something wrong with parameters");
331
332         internalError = true;
333     }
334
335     step = possibleBytesToHideCount / (myHeader.fileSize + myHeader.headerSize);
336
337     LOG(LOG_INF, "Reading additional header done");
338 }
339
340 char WavFile::readHiddenByte(FILE **file)
341 {
342     if (internalError)
343     {
344         LOG(LOG_ERR, "InternalError");
345         return 0;
346     }
347
348     char byte = 0, temp = 0;
349
350     for (int i = 0; i < 8; i++, temp = 0)
351     {
352         temp = fgetc(*file);
353         temp = lfsr->getByte(temp);
354         byte |= ((temp & 0x1) << i);
355
356         for (int skipBytes = 0; skipBytes < wavHeader.BitsPerSample / 8 - 1; skipBytes++)
357         {
358             fgetc(*file);
359         }
360     }
361
362     return byte;
363 }
364
365 void WavFile::hideByte(FILE **fileContainer, FILE **fileResult, char byte)
366 {
367     if (internalError)
```

WavFile.cpp

WavFile.cpp > WavFile::hideByte(FILE **, FILE **, char)

```
366     }
367     if (internalError)
368     {
369         LOG(LOG_ERR, "InternalError");
370         return;
371     }
372
373     char fileByte = 0;
374     for (int bit = 0; bit < 8; bit++)
375     {
376         fileByte = fgetc(*fileContainer);
377         fileByte = (fileByte & 0xFE) | ((char)((byte >> bit) & 1));
378         fputc(lfsr->getByte(fileByte), *fileResult);
379
380         for (int skipBytes = 0; skipBytes < wavHeader.BitsPerSample / 8 - 1; skipBytes++)
381         {
382             fputc(fgetc(*fileContainer), *fileResult);
383         }
384     }
385 }
386
387 int WavFile::encryptFile(char* contFilePath, char* binFilePath, char* outFilePath)
388 {
389     if (internalError)
390     {
391         LOG(LOG_ERR, "InternalError");
392         return WAV_ERROR;
393     }
394
395     FILE *contFile = NULL, *binFile = NULL, *outFile = NULL;
396     unsigned char header[WAV_HEADER_SIZE];
397
398     // check and Initialize parent & txt files...
399     if (checkFilesForHiding(contFilePath, binFilePath) != WAV_SUCCESS)
400     {
401         LOG(LOG_ERR, "InternalError");
402         return WAV_ERROR;
403     }
404
405     LOG(LOG_INF, "Hiding binary file...");
406
407     contFile = fopen(contFilePath, "rb");
408     binFile = fopen(binFilePath, "rb");
409     outFile = fopen(outFilePath, "w+b");
410
411     fread(header, WAV_HEADER_SIZE, 1, contFile);        // read WAV header
412     fwrite(header, WAV_HEADER_SIZE, 1, outFile);        // write WAV header
413
414     //for case of LIST presence
415     for (uint32_t i = 0; i < wavHeader.listSize; i++)
```

WavFile.cpp

WavFile.cpp > WavFile::hideByte(FILE **, FILE **, char)

```
414     //for case of LIST presence
415     for (uint32_t i = 0; i < wavHeader.listSize; i++)
416     {
417         fputc(fgetc(contFile), outFile);
418     }
419
420     writeHeader(&contFile, &outFile);
421
422     // main hiding process
423     while (!feof(binFile))
424     {
425         hideByte(&contFile, &outFile, fgetc(binFile));
426         if (!feof(binFile))
427         {
428             for (uint32_t skipByte = 0; skipByte < step * (wavHeader.BitsPerSample / 8); skipByte++)
429             {
430                 fputc(fgetc(contFile), outFile);
431             }
432         }
433     }
434
435     // write remaing wav bytes into the new file.
436     while (!feof(contFile))
437     {
438         fputc(fgetc(contFile), outFile);
439     }
440
441     // close all file handlers
442     fclose(contFile);
443     fclose(binFile);
444     fclose(outFile);
445
446     LOG(LOG_INF, "Hiding binary file done");
447
448     return WAV_SUCCESS;
449 }
450
451 int WavFile::decryptFile(char* encFilePath)
452 {
453     if (internalError)
454     {
455         LOG(LOG_ERR, "InternalError");
456         return WAV_ERROR;
457     }
458
459     LOG(LOG_INF, "Revealing binary file...");
460
461     FILE *encFile = NULL, *binFile = NULL;
462
```

WavFile.cpp

WavFile.cpp > WavFile::hideByte(FILE **, FILE **, char)

```
446     LOG(LOG_INF, "Hiding binary file done");
447
448     return WAV_SUCCESS;
449 }
450
451 int WavFile::decryptFile(char* encFilePath)
452 {
453     if (internalError)
454     {
455         LOG(LOG_ERR, "InternalError");
456         return WAV_ERROR;
457     }
458
459     LOG(LOG_INF, "Revealing binary file...");
460
461     FILE *encFile = NULL, *binFile = NULL;
462
463     encFile = fopen(encFilePath, "rb");
464
465     // skip WAV header
466     fseek(encFile, WAV_HEADER_SIZE + wavHeader.listSize, SEEK_SET);
467
468     readHeader(&encFile);
469     binFile = fopen(myHeader.name, "w+b");
470
471     uint32_t i = 0;
472     while (!feof(encFile) && i < myHeader.fileSize)
473     {
474         fputc(readHiddenByte(&encFile), binFile);
475         i++;
476         for (uint32_t skipByte = 0; skipByte < step * (wavHeader.BitsPerSample / 8); skipByte++)
477         {
478             fgetc(encFile);
479         }
480     }
481
482     fclose(encFile);
483     fclose(binFile);
484
485     LOG(LOG_INF, "Revealing binary file done");
486
487     return WAV_SUCCESS;
488 }
489
```