Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка програми захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування

Виконав: студент 4 курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

_____ Бучинський Б.В.
(прізвище та ініціали)

Керівник: к.т.н., доц., зав. каф. МБІС
_____ Карпінець В.В.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
_____ Савицька Л.А.
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“_____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту

Бучинському Богдану Васильовичу
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка програми захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування
Керівник роботи Карпінець В.В. к.т.н., доц., зав. каф. МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи "2" червня 2025 року
3. Вихідні дані до роботи Метою даної роботи є розробка програми захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування
4. Зміст текстової частини: У першому розділі проаналізовано предметну область захисту авторських прав на студійні аудіозаписи формату AIFF, наведено класифікацію методів стеганографії та обґрунтовано вибір фазового методу кодування для приховування авторської інформації. Другий розділ присвячений проектуванню та опису алгоритму фазового стеганографічного кодування, адаптованого для роботи з AIFF-файлами. Також у цьому розділі подано блок-схеми, описано принцип дії алгоритму та спроєктовано структуру бази даних для збереження метаінформації. У третьому розділі представлено програмну реалізацію веб-застосунку, який забезпечує кодування та декодування авторських даних у аудіофайлах, включаючи серверну логіку на основі Python та клієнтську частину з інтерфейсом. Реалізовано інтеграцію розробленого модуля з базою даних і проведено тестування функціоналу.
5. Перелік графічного матеріалу:

У першому розділі бакалаврської кваліфікаційної роботи наявно 2 рисунки, у другому – 7 рисунків, у третьому розділі – 5 рисунків.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Карпінець В.В., к.т.н., доц. зав. каф. МБІС		
Розділ II	Карпінець В.В., к.т.н., доц. зав. каф. МБІС		
Розділ III	Карпінець В.В., к.т.н., доц. зав. каф. МБІС		

7. Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент _____ Бучинський Б.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карпінець В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 109 сторінок формату А4, на яких наведено 14 рисунків, 5 таблиць, список використаних джерел містить 33 найменувань.

Метою роботи є розробка програмного засобу захисту авторських прав студійних аудіозаписів формату високої якості AIFF шляхом використання фазового методу стеганографічного кодування.

У загальній частині роботи розглянуто сучасні підходи до цифрової стеганографії, методи приховування інформації в аудіофайлах та обґрунтовано вибір фазового методу як найбільш придатного для роботи з форматом AIFF.

У частині проєктування подано опис структури програмного забезпечення, принцип роботи алгоритму, побудовано схему даних, діаграму класів та описано формат повідомлення, що підлягає приховуванню.

У частині програмної реалізації розроблено вебзастосунок на основі Python і Flask, який дозволяє вбудовувати та витягувати ідентифікаційну інформацію з AIFF-файлів за допомогою методу фазового кодування, а також зберігати результати обробки.

Ключові слова: стеганографія, фазове кодування, AIFF, авторське право, аудіозапис.

ABSTRACT

The bachelor's qualification work consists of 109 pages of A4 format, which contains 14 figures, 5 tables, the list of references contains 33 items.

The aim of the work is to develop a software tool for copyright protection of high quality AIFF studio audio recordings by using the phase steganographic coding method.

In the general part of the paper, we consider modern approaches to digital steganography, methods of hiding information in audio files, and justify the choice of the phase method as the most suitable for working with the AIFF format.

In the design part, we describe the structure of the software, the principle of the algorithm, build a data diagram, a class diagram, and describe the format of the message to be hidden.

As for the software implementation, a web application based on Python and Flask has been developed that allows embedding and extracting identification information from AIFF files using the phase encoding method, as well as saving the processing results.

Keywords: steganography, phase coding, AIFF, copyright, audio recording

ЗМІСТ

ВСТУП 4

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МЕТОДІВ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ 6

1.1 Аналіз предметної області захисту авторських прав у цифрових аудіозаписах 6

1.2 Базові поняття стеганографії та її роль у забезпеченні захисту авторських прав 9

1.3 Порівняльний аналіз аудіоформатів. Обґрунтування вибору формату високої якості AIFF 11

1.4 Аналіз роботи стеганографічного методу фазового кодування для аудіофайлів 20

1.5 Висновки та постановка задачі 22

2 РОЗРОБКА АЛГОРИТМУ РОБОТИ ПРОГРАМИ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ ВИСОКОЇ ЯКОСТІ AIFF НА ОСНОВІ МЕТОДУ ФАЗОВОГО КОДУВАННЯ 23

2.1 Обґрунтування підходу до використання методу фазового кодування .. 23

2.2 Розробка алгоритму інтеграції фазового кодування у формат AIFF 26

2.3 Проєктування архітектури програмного забезпечення 34

2.4 Висновки до розділу 2 39

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ ФОРМАТУ AIFF НА ОСНОВІ ФАЗОВОГО КОДУВАННЯ 41

3.1 Обґрунтування вибору мови програмування та технологій розробки 41

3.2 Реалізація ключових логічних модулів програмного забезпечення 47

3.3 Реалізація UI/UX інтерфейсу програмного забезпечення 62

3.4 Інструкція користувача та рекомендації щодо використання 67

3.5 Висновки до розділу 3 71

ВИСНОВОК 72

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 74

ДОДАТКИ 77

Додаток А. Технічне завдання 78

Додаток Б. Лістинг коду основної логіки програмного забезпечення 82

Додаток В. Лістинг коду графічного інтерфейсу і компонентів взаємодії з користувачем 87

Додаток Г. Ілюстративний матеріал 102

Додаток Д. Протокол перевірки на антиплагіат 109

ВСТУП

Актуальність. У сучасному світі обсяг створюваного цифрового контенту зростає в геометричній прогресії, що створює нові виклики у сфері захисту авторських прав. Зокрема, студійні аудіозаписи високої якості у форматі AIFF, які широко використовуються в професійній звукозаписувальній індустрії, стають об'єктом частих спроб піратства та несанкціонованого копіювання. Особливість цього формату полягає у його відкритій структурі, що дозволяє вільно редагувати аудіофайли, але водночас підвищує ризики порушення прав інтелектуальної власності.

Захист авторських прав на аудіо-контент потребує нових технологічних рішень, які можуть забезпечити стійкість до модифікацій, не порушуючи якості звучання. Серед сучасних підходів стеганографічні методи є одними з найперспективніших. Вони дозволяють приховувати додаткову інформацію (наприклад, авторські дані чи водяні знаки) без видимих змін для кінцевого споживача. Метод фазового кодування є особливо ефективним для роботи з аудіосигналами, оскільки дозволяє зберігати їхню акустичну якість, водночас забезпечуючи надійне приховання інформації.

Таким чином, розробка програмного засобу для стеганографічного захисту аудіозаписів у форматі AIFF є актуальною задачею, яка відповідає сучасним потребам звукозаписувальної індустрії. Ця робота спрямована на вирішення нагальної проблеми захисту авторських прав із використанням сучасних інформаційних технологій.

Мета роботи. Створення програмного модуля для захисту студійних аудіофайлів формату AIFF шляхом приховування текстової інформації у звуковому сигналі. Для демонстрації роботи методу буде реалізовано веб-застосунок, що забезпечує кодування та декодування повідомлень з використанням фазового кодування. Для досягнення поставленої мети необхідно виконати такі завдання:

– провести аналіз предметної області та обґрунтувати доцільність використання фазового кодування для захисту аудіо-файлів;

– ознайомитися з існуючими підходами та програмними рішеннями у сфері аудіо-стеганографії;

– розробити архітектуру модуля фазового кодування;

– обрати оптимальний метод представлення і вбудовування текстової інформації у фазову компоненту сигналу;

– реалізувати веб інтерфейс для зручної взаємодії користувача з програмою;

– провести тестування модуля на прикладах студійних AIFF-записів.

Предметом дослідження є розробка і реалізація веб застосунку для фазового кодування студійних аудіозаписів у форматі AIFF, спрямованого на захист авторських прав шляхом приховування текстової інформації в аудіосигналі. Основна увага приділяється використанню методу фазового кодування як засобу прихованого водяного знаку для забезпечення автентичності та захисту прав інтелектуальної власності.

Об’єктом дослідження є процес захисту студійних аудіозаписів у форматі AIFF з використанням методів приховання інформації. Важливою частиною дослідження є застосування методу фазового кодування для забезпечення конфіденційності, цілісності та доступності вбудованих даних, а також аналіз ефективності цього методу у контексті сучасних засобів обробки аудіосигналів.

Практична цінність. Практична цінність роботи полягає у створенні програмного забезпечення, яке може бути використане у звукозаписувальній індустрії для захисту авторських прав на аудіо-контент. Розроблене рішення забезпечить приховання даних у студійних аудіофайлах без втрати їхньої якості, що є критично важливим для професійного використання. Інструмент, створений у рамках дослідження, може знайти застосування як у професійних студіях звукозапису, так і у компаніях, які займаються дистрибуцією аудіо продукції.

Новизна роботи полягає в адаптації та впровадженні методу фазового кодування до формату AIFF з урахуванням його специфічної структури. На відміну від загальних стеганографічних методів, запропонований підхід дозволяє інтегрувати приховану інформацію в аудіофайл, зберігаючи його студійну якість і стійкість до редагування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МЕТОДІВ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ

1.1 Аналіз предметної області захисту авторських прав у цифрових аудіозаписах

Захист авторських прав у цифрових аудіозаписах є однією з найбільш актуальних тем в умовах швидкого розвитку інформаційних технологій та збільшення обсягу онлайн-розповсюдження контенту. Аудіозаписи, як і будь-які інші цифрові медіа, піддаються ризику несанкціонованого копіювання, розповсюдження та використання, що створює проблеми для власників прав, артистів, студій та інших учасників індустрії. У зв'язку з цим необхідно застосовувати ефективні методи захисту авторських прав, що дозволяють ідентифікувати власника контенту та запобігати його нелегальному використанню.

У минулому захист аудіозаписів від порушень авторських прав здебільшого здійснювався через фізичні носії, такі як платівки, касети або компакт-диски. Однак з розвитком цифрових технологій і появою нових форматів аудіофайлів, таких як MP3, AIFF, WAV, захист контенту став значно складнішим. Проблеми, пов’язані з цифровим копіюванням і тиражуванням, включають можливість створення абсолютно точних копій аудіофайлів, що призводить до масового поширення нелегальних версій творів без участі авторів. Для боротьби з нелегальним копіюванням у цифровому середовищі розроблені різні механізми захисту, серед яких важливими є цифрові водяні знаки, криптографічні методи захисту та стеганографія. Кожен з цих підходів має свої особливості і обмеження, що зумовлює необхідність поєднання декількох методів для досягнення оптимального рівня захисту [1].

Одним з найпоширеніших способів захисту цифрових аудіофайлів є використання цифрових водяних знаків. Ці знаки, що вбудовуються безпосередньо в аудіофайл, містять ідентифікаційну або захисну інформацію, яку важко видалити без істотного впливу на якість звуку. Водяні знаки можуть бути як видимими (для відео та зображень), так і невидимими – для аудіо.

В основному використовуються два типи цифрових водяних знаків: лінійні (які додаються до аудіофайлів у вигляді шуму або малопомітних змін) та нелінійні (які вбудовуються на основі більш складних алгоритмів). Водяні знаки можуть бути вбудовані в спектр звукових хвиль, що дозволяє забезпечити високий рівень прихованості інформації, водночас зберігаючи якість запису [2].

Однак методи водяних знаків мають кілька недоліків, серед яких – можливість їх видалення через спеціалізовані програми, що впливають на аудіофайл. Крім того, існує проблема втрати якості звуку при надмірному використанні водяних знаків, що може бути критичним у професійному середовищі, де кожен звук має важливе значення. Іншим підходом до захисту авторських прав є криптографія. Застосування криптографії дозволяє зашифрувати аудіофайл або його частини, що робить неможливим його прослуховування або зміну без відповідного ключа. Основною метою криптографічного захисту є забезпечення конфіденційності, цілісності та аутентифікації даних. Ці методи використовуються для захисту файлів під час передачі через Інтернет або при зберіганні на цифрових носіях.

Стеганографія, або методи прихованого вбудовування інформації, є однією з найбільш перспективних технологій захисту авторських прав у цифрових аудіозаписах. Основною перевагою стеганографії є те, що вона дозволяє вбудовувати захисну інформацію, не змінюючи суттєво аудіофайл і не впливаючи на його якість. Стеганографічні методи можуть використовуватися для вставки як текстових, так і числових даних, які містять відомості про власника авторських прав, дату створення, чи інші важливі атрибути. Одним з основних методів стеганографії є використання так званого "методу фазового кодування" або "методу прихованих звукових послідовностей", що дозволяє вбудовувати захисну інформацію в аудіофайли без втрати якості та без явних змін, помітних для слухача. Цей метод є досить надійним, оскільки для його виявлення необхідно мати спеціальне програмне забезпечення [3].

Однак, стеганографія також має свої недоліки. Оскільки вона передбачає приховану передачу інформації, існує можливість її виявлення або видалення за

допомогою спеціалізованих інструментів. Тому важливо розробляти нові методи, які поєднують стеганографію з іншими підходами, наприклад, цифровими водяними знаками або криптографією. Хоча існуючі методи захисту авторських прав у цифрових аудіозаписах вже мають певний рівень ефективності, вони не позбавлені обмежень, що викликає потребу в розробці нових, більш досконалих технологій. Перш за все, це стосується пошуку методів, які не тільки забезпечують високий рівень захисту, але й зберігають високу якість звуку, що є критично важливим для студійних записів.

Таким чином, для забезпечення надійного захисту авторських прав на цифрові аудіозаписи необхідно постійно удосконалювати існуючі методи та розробляти нові підходи, які враховують специфіку аудіоформатів та сучасні технологічні виклики. У цьому контексті стеганографія є однією з найбільш перспективних технологій для вирішення проблеми захисту цифрових аудіофайлів.

Враховуючи постійний розвиток цифрових технологій та зростання обсягів цифрового контенту, зокрема аудіозаписів, існує постійна потреба у вдосконаленні існуючих методів захисту авторських прав. Зокрема, одним із найбільш перспективних напрямків є інтеграція декількох технологій у єдину систему захисту. Поєднання криптографії, стеганографії та цифрових водяних знаків може забезпечити високий рівень захисту аудіофайлів від несанкціонованого копіювання, модифікації та поширення, з одночасним збереженням їхньої якості. У цьому контексті важливим є також застосування технологій блокчейн для забезпечення прозорості та довіри до авторських прав. Використання дистрибуційних реєстрів на базі блокчейн-технологій дозволяє створити незмінні записи про володіння авторськими правами на цифрові аудіофайли, що значно ускладнює фальсифікацію інформації про власника та відслідковування прав на певний контент. Блокчейн може використовуватися для реєстрації аудіофайлів та визначення права на їхнє відтворення, розповсюдження або модифікацію [4].

Захист авторських прав на цифрові аудіозаписи є складним та багатогранним завданням, яке потребує використання сучасних технологій, таких як стеганографія, криптографія та цифрові водяні знаки. Розвиток цих технологій

дозволяє створювати ефективні та безпечні методи захисту контенту, що сприяє збереженню прав авторів, зниженню рівня піратства та покращенню загального контролю за використанням цифрових медіа. Водночас, необхідність у постійному вдосконаленні цих методів та їх інтеграції в різні платформи для поширення контенту відкриває нові перспективи для інноваційних розробок у галузі захисту цифрових прав.

1.2 Базові поняття стеганографії та її роль у забезпеченні захисту авторських прав

Стеганографія – це наука і техніка приховування інформації таким чином, щоб її існування було непомітним для сторонніх осіб. Відмінність стеганографії від криптографії полягає в тому, що стеганографія не тільки приховує зміст інформації, але й її сам факт. Мета стеганографії – забезпечити секретність передачі даних, приховуючи їх у найнезначніших деталях інших файлів, таких як зображення, аудіо, відео або навіть текст. Одним з основних принципів стеганографії є вбудовування інформації в контейнер (т. зв. носій), при цьому змінюється лише частина носія, яка є невидимою або нечутною для людського сприйняття. Це дозволяє здійснювати захист інформації без зміни видимих чи відчутних властивостей файлу. У контексті цифрових аудіозаписів, наприклад, це може бути приховування даних у невидимих частинах звукової хвилі або в частотних діапазонах, де людське вухо не здатне виявити зміни.

Основними методами стеганографії є методи на основі зміщення бітів (інформація приховується за допомогою зміщення одиничних бітів в бітовому потоці аудіофайлу), формати сигналів і частот (вкладання інформації відбувається за допомогою маніпуляцій з певними частотами або сигналами, які є невідчутними для слуху), методи на основі хаотичних перетворень (використання алгоритмів, що дозволяють здійснити зміну певних частин аудіофайлу з огляду на математичні моделі, котрі не змінюють чутності для людини). У стеганографії важливою є здатність прихованої інформації зберігати свою непомітність, незважаючи на маніпуляції з носієм. Таким чином, стеганографічний метод повинен бути

високоефективним не тільки з точки зору вбудовування, але й з точки зору витривалості до змін та аналізу [5].

Захист авторських прав є однією з основних задач у сучасному цифровому світі, де обмін контентом відбувається швидко і в величезних масштабах. Аудіозаписи, як і інші види цифрових медіа, піддаються ризику несанкціонованого використання, копіювання, модифікації або поширення без дозволу власника прав. У цьому контексті стеганографія стає важливим інструментом для забезпечення автентичності, захисту та ідентифікації авторських прав. Стеганографія є ефективним інструментом захисту авторських прав на аудіоконтент, зокрема через вбудовування цифрових водяних знаків, які містять інформацію про власника прав, дату створення чи інші ідентифікаційні дані. Такий підхід забезпечує прихованість авторських атрибутів без помітного впливу на якість звуку. Вбудовані дані можуть використовуватись для підтвердження авторства або виявлення випадків нелегального розповсюдження аудіофайлів.

Одна з основних проблем, пов’язаних із стеганографічним вбудовуванням у аудіофайли, – це можливість зміни звукового файлу через компресію або обробку, що може призвести до втрати прихованої інформації. Наприклад, при конвертації файлів у інший формат, чи під час стискання, можуть бути видалені або змінені частини звуку, які містять приховані дані. Важливим аспектом є те, що стеганографічні методи повинні забезпечувати високу якість звуку після вбудовування водяних знаків. Тому використання стеганографії в аудіофайлах вимагає ретельного підходу до вибору технологій та методів, які мінімально змінюють звукову характеристику.

Одним із найбільш ефективних методів стеганографії для захисту авторських прав у цифрових аудіофайлах є фазове кодування. Цей метод використовує особливості фазових змін сигналу для приховування додаткової інформації, що забезпечує високу ступінь непомітності та ефективності стеганографії в аудіофайлах. Фазове кодування полягає в зміні фази звукових хвиль, що лежать в межах одного циклу сигналу. Оскільки людське вухо не дуже чутливе до малих змін фази, цей метод дозволяє вбудовувати приховану інформацію без значних змін у

сприйнятті звуку. Під час кодування інформації в фазу звукової хвилі, зміна відбувається таким чином, що вона не змінює амплітуду чи частоту звуку, що дозволяє зберігати оригінальну якість аудіофайлу [6].

Одним із основних ризиків використання фазового кодування є те, що деякі методи стиснення аудіофайлів (наприклад, через формат MP3) можуть видаляти або змінювати частини звукової хвилі, в тому числі й приховану інформацію, що вбудована за допомогою фазового кодування. Важливо, щоб стеганографічні методи були стійкими до таких змін. Оскільки фазове кодування базується на незначних змінах фази, важливо забезпечити високу точність в відновленні прихованих даних. Невеликі похибки під час вбудовування або вилучення інформації можуть призвести до втрати або пошкодження прихованих даних, що ставить під загрозу захист авторських прав.

Фазове кодування є потужним інструментом стеганографії, який має великий потенціал для захисту авторських прав у цифрових аудіофайлах. Завдяки здатності забезпечувати високу непомітність і зберігати якість звуку, цей метод дозволяє надійно захищати інтелектуальну власність в аудіо-індустрії. Однак для досягнення максимальної ефективності необхідно враховувати обмеження методу, зокрема стійкість до обробки та стиснення файлів. Розвиток адаптивних алгоритмів і комбінування фазового кодування з іншими захисними методами є перспективними напрямками для подальших досліджень у цій галузі.

1.3 Порівняльний аналіз аудіоформатів. Обґрунтування вибору формату високої якості AIFF

Для реалізації програми стеганографічного приховування інформації ключове значення має вибір аудіоформату. Саме формат збереження аудіо визначає не лише якість сигналу, а й технічні можливості обробки, серед яких особливо важливими є доступ до спектральних характеристик, точність представлення амплітудно-частотних та фазових параметрів, а також підтримка додаткових структур даних, зокрема метаданих. Формат має забезпечувати точне збереження сигналу без спотворень, що особливо критично для алгоритмів стеганографії, які

працюють у частотній або часово-частотній області. Також важливо, щоб обраний формат був сумісний з програмними засобами аналізу та синтезу сигналів, забезпечував просту інтеграцію в обчислювальне середовище та дозволяв виконувати спектральну модифікацію без втрат. Стиснення без втрат, можливість роботи з високою розрядністю та частотою дискретизації, прозоре збереження фазових компонент – ці характеристики відіграють визначальну роль під час вибору формату.

З огляду на ці вимоги, до розгляду беруться найбільш поширені формати цифрового аудіо (див. табл. 1.1): MP3, AAC, WAV, FLAC та AIFF. Кожен із них має свої характерні особливості, рівень підтримки в системах обробки сигналів та відповідність завданням високоякісного приховування даних. Порівняння проводиться за критеріями придатності до задач спектральної модифікації, ступеня стиснення, підтримки точності сигналу, наявності артефактів після декодування, і на основі цього формується обґрунтований вибір найбільш оптимального формату для реалізації фазового стеганографічного алгоритму.

Таблиця 1.1 – Загальна структура бінарного формату AIFF

Формат	Стиснення	Якість звуку	Підтримка метаданих	Підтримка платформами
MP3	З втратами	Середня	Так	Висока
AAC	З втратами	Середня	Так	Висока
WAV	Без стиснення	Висока	Обмежена	Висока
FLAC	Без втрат	Висока	Так	Середня
AIFF	Без стиснення	Дуже висока	Так	Середня

Формат MP3 (MPEG Audio Layer III) є одним з найстаріших та найпопулярніших форматів зі стисненням аудіо з втратами. Його головною перевагою є висока ступінь стиснення, що забезпечує невеликий розмір файлів. Це зробило MP3 стандартом де-факто для музичних плеєрів, потокових сервісів та мобільних пристроїв. Проте, MP3 застосовує алгоритми попереднього стиснення, які призводять до втрати частини спектру звукового сигналу, особливо високочастотної складової. Таке стиснення унеможливлює точне відновлення

оригінального сигналу і серйозно обмежує можливість використання MP3 у задачах, що потребують спектральної точності, зокрема в стеганографії, яка базується на модифікації фази сигналу. У цьому контексті формат MP3 є непридатним через незворотні втрати інформації при кодуванні.

Формат AAC (Advanced Audio Coding) був розроблений як покращення MP3 і забезпечує вищу якість звуку при тому ж бітрейті. Він став основним форматом для iTunes, YouTube, iOS-пристроїв та багатьох потокових сервісів. AAC ефективніше використовує спектр частот, краще зберігає динаміку та деталізацію сигналу. Але, попри покращення в алгоритмах, цей формат також є форматом зі стисненням з втратами. Тобто, при збереженні аудіо у форматі AAC деякі дані все одно безповоротно втрачаються, а сама структура сигналу спотворюється. Як і у випадку з MP3, це виключає можливість точної обробки спектру сигналу, що критично для методів, таких як фазове кодування. Таким чином, AAC, хоча і кращий за MP3, також не може використовуватись як повноцінна основа для стеганографії.

Формат WAV (Waveform Audio File Format) є стандартом Microsoft і широко використовується в середовищі Windows. WAV підтримує збереження звуку у вигляді необроблених PCM-даних без стиснення, що робить його придатним для задач, де важлива точність. Його просто обробляти, структура зрозуміла й лінійна. Проте цей формат має деякі обмеження: він не оптимізований для багатоплатформенного використання (особливо в середовищах macOS та Linux), має дещо обмежену підтримку метаданих і менш гнучкий у розширенні функціональності. У порівнянні з AIFF, WAV програє в аспектах універсальності, гнучкості структури та стандартної підтримки розширень, які часто потрібні в сучасних мультимедійних додатках.

Формат FLAC (Free Lossless Audio Codec) дозволяє зберігати звук без втрат, при цьому здійснюючи ефективне стиснення даних. Це означає, що після декодування сигнал ідентичний оригінальному, а розмір файлу менший, ніж у WAV чи AIFF. FLAC користується популярністю в різних спільнотах і серед тих, хто хоче зберігати музику без втрат. Проте, попри його ефективність у збереженні якості,

структура стиснення додає складності до прямої роботи зі спектром сигналу. Обробка фазових характеристик у FLAC значно ускладнюється, і для її виконання потрібне попереднє декодування. Це робить FLAC менш зручним для задач реального часу або інтеграції у прості програмні рішення зі стеганографії.

Формат AIFF (Audio Interchange File Format), який був обраний у межах даної бакалаврської кваліфікаційної роботи, є безкомпресійним форматом, створеним Apple, який зберігає звук у PCM-формі з максимальною точністю. Він забезпечує високу якість звучання, повну відсутність втрат і широкі можливості зберігання допоміжної інформації (метаданих). AIFF підтримується більшістю сучасних професійних аудіоредакторів, а також добре інтегрується в середовищах macOS та Linux. Структура формату дозволяє зручно працювати зі спектром сигналу без проміжних перетворень або втрати якості. Саме ці характеристики роблять AIFF ідеальним для задач стеганографії, зокрема реалізації фазового кодування, де вкрай важливо зберігати точну фазову структуру сигналу. Крім того, формат AIFF є надійним з точки зору повторного використання даних, а його підтримка з боку сучасних бібліотек для роботи з аудіо спрощує програмну реалізацію.

Підсумовуючи, можна зробити висновок, що більшість популярних форматів аудіо (MP3, AAC, FLAC) не забезпечують необхідного рівня точності або зручності обробки для задач стеганографії. Лише формати без втрат, такі як WAV та AIFF, придатні для застосування в подібних системах. Проте саме AIFF надає найбільшу гнучкість, структурованість, точність і підтримку, що робить його найкращим вибором для розробки програмних рішень, орієнтованих на приховування інформації в аудіосигналі за допомогою фазового кодування. Саме тому цей формат було обрано як базовий у даній роботі [7].

Аудіоформат AIFF (Audio Interchange File Format) був розроблений компанією Apple в середині 1980-х років як стандарт для зберігання високоякісних аудіофайлів у незжатому вигляді. Завдяки своїй високій якості відтворення, формат AIFF широко використовується в професійній аудіо- та відеопродукції, звукозапису, пост-продакшн і музичному виробництві. Одна з головних характеристик формату AIFF – це його здатність зберігати аудіо без втрат (див. рис. 1.1), що дає змогу точно

передавати всі нюанси звуку без втрати інформації. Це дозволяє AIFF бути одним із найкращих форматів для професійного використання, де точність відтворення аудіо є критично важливою [8].

Рисунок 1.1 – Вигляд звукової доріжки аудіофайлу формату AIFF

Файл AIFF складається з кількох основних компонентів, зокрема заголовка, який вказує на тип файлу і розмір, а також різних блоків даних, таких як аудіо-інформація та метадані. Ось як виглядає базова структура файлу AIFF:

Заголовок (Header) – перші кілька байтів файлу містять інформацію про тип файлу та загальний розмір. Заголовок необхідний для того, щоб програма, що відкриває файл, могла правильно інтерпретувати його вміст.

Запис FORM – один із основних елементів структури файлу. Він містить інформацію про формат файлу та вказує на основний вміст даних у файлі. Це спеціальний заголовок, що використовується для вказівки на початок основного контейнера даних.

Запис FVER – містить версію формату файлу, що дозволяє програмному забезпеченню правильно обробляти файл, визначаючи його сумісність з різними версіями стандарту AIFF.

Запис SSND – це основний блок даних, що містить самі звукові хвилі. У цьому блоці зберігаються аудіо-дані у вигляді числових значень амплітуди звукової хвилі, які відображають звук у часі. Звукові дані зберігаються в вигляді масивів вибірок (samples), що дозволяють точно відтворювати оригінальний аудіосигнал.

Запис COMM – в цьому блоці зберігається метаінформація про самі звукові дані, така як кількість каналів (моно або стерео), частота дискретизації (кількість вибірок на секунду), та кількість вибірок, що описують тривалість аудіофайлу.

Запис NAME – в ньому можуть зберігатися метадані, такі як ім’я треку, виконавець, альбом, жанр, дата створення та інша інформація, яка може бути корисною для організації та пошуку аудіофайлів [9].

Один із основних аспектів формату AIFF – це зберігання аудіо у вигляді вибірок. Вибірка (sample) – це представлення амплітуди звукової хвилі в певний момент часу. Чим більше вибірок на одиницю часу (частота дискретизації), тим точніше відтворюється звукова хвиля.

Для збереження аудіо AIFF використовує 16 або 32 бітову точність для кожної вибірки. Однак у деяких випадках AIFF може підтримувати ще вищу точність для професійних застосувань. Зазвичай вибірки представляються як цілі числа або числа з плаваючою комою, що дозволяє зберігати більшу кількість інформації про амплітуду звукової хвилі.

Частота дискретизації є ще одним важливим параметром у структурі AIFF. Вона визначає, скільки разів за секунду зчитується амплітуда звукової хвилі. Для форматів високої якості, таких як AIFF, стандартною частотою є 44,1 кГц, що відповідає стандарту CD-аудіо.

Для відео та професійних студійних записів часто використовують частоту 48 кГц або навіть вищу. Завдяки такій точності збереження, AIFF є ідеальним форматом для зберігання даних без втрат, де кожен відтворений звук є максимально близьким до оригіналу. Така точність є критично важливою для аудіофайлів, що використовуються в професійних умовах, де важливо зберегти всі нюанси звучання. У такому методі інформація може бути прихована в фазовому спектрі звукової хвилі без порушення її амплітуди або інших характеристик, що безпосередньо впливають на якість звуку. Таке приховування інформації є дуже надійним і непомітним, оскільки в слуховому відношенні зміни в фазі звукової хвилі не завжди легко сприймаються людським вухом [10].

Таблиця 1.2 – Загальна структура бінарного формату AIFF

Тип блоку	Опис	Розмір
FORM	Заголовок, що вказує на тип файлу та його загальний розмір	4 байти
FVER	Версія формату файлу	4 байти
COMM	Описує основні параметри аудіо, такі як кількість каналів, частота дискретизації та кількість вибірок	18 байт
SSND	Основні аудіо-дані у вигляді вибірок амплітуд звукових хвиль	Змінний розмір
NAME	Метадані, такі як ім’я треку, виконавець, альбом та інша інформація	Змінний розмір

Для впровадження методу фазового кодування в AIFF необхідно враховувати кілька аспектів.

По-перше, важливо правильно вибрати ділянки звукової хвилі, в які буде вбудовуватися інформація. Це може бути зроблено шляхом зміни фази в окремих частинах сигналу, що менш помітно для слухача, але достатньо ефективно для зберігання даних.

По-друге, необхідно враховувати, що AIFF зберігає аудіо дані у вигляді числових вибірок, що дає можливість безпосередньо працювати з їхніми значеннями. Іншим можливим методом стеганографії в AIFF є використання метаданих, які вже присутні в структурі файлу. Оскільки AIFF підтримує збереження метаданих, таких як ім’я треку, виконавець, альбом і інші параметри, ці місця можуть бути використані для приховання невеликих обсягів інформації. Наприклад, невеликий текстовий або бінарний код може бути вставлений у певні поля метаданих, що дозволяє захистити авторські права або додавати інші види захисту.

Детально побачити бінарну структуру формату файла AIFF можна побачити в будь-якому бінарному редакторі (рис. 1.2).

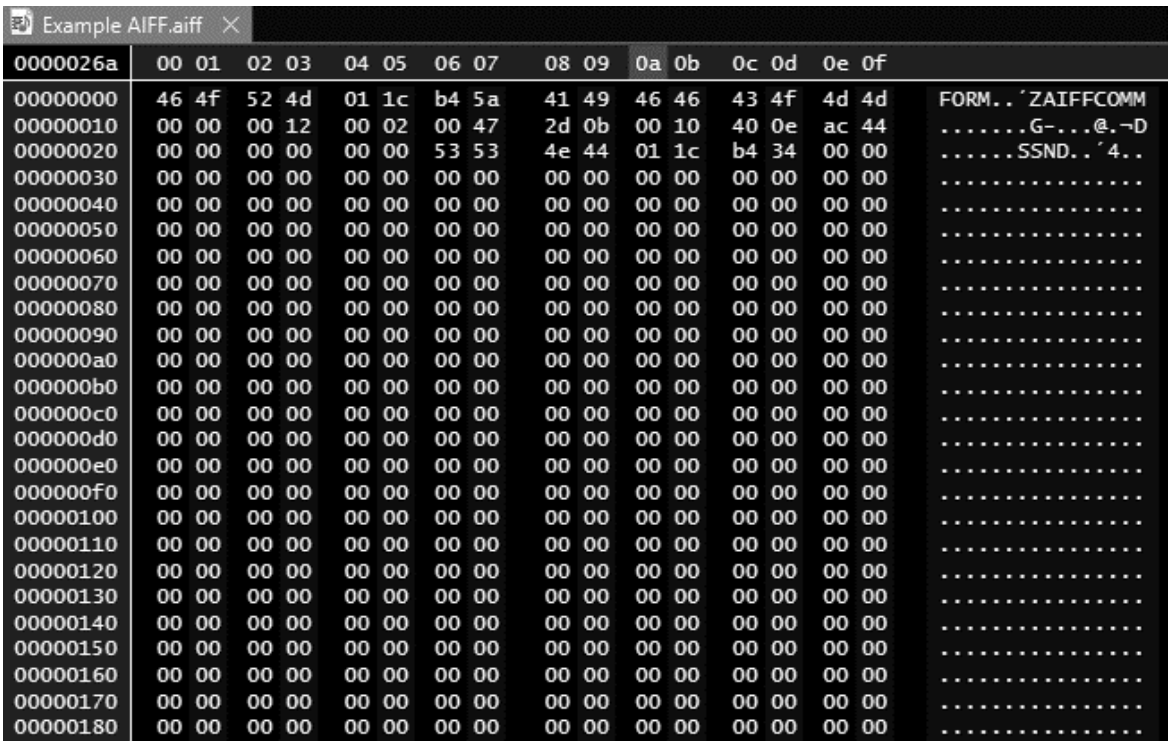


Рисунок 1.2 – Бінарна структура формату аудіо AIFF

Для більш детального розуміння структури та характеристик аудіоформату AIFF варто звернути увагу на деякі додаткові аспекти, що впливають на використання цього формату в професійному середовищі та його придатність для застосування стеганографії.

AIFF використовує FORM для організації даних у своєму файлі. Формат даних у файлі AIFF розбитий на кілька блоків, кожен з яких виконує свою функцію. Блок FORM є головним елементом, який визначає тип даних, що зберігаються в файлі. Він включає в себе кілька підблоків, зокрема блоки FVER (версія формату), COMM (метадані про аудіофайл), SSND (основні звукові дані) і NAME (метадані файлу). Усередині блоку COMM зберігаються базові характеристики аудіофайлу, що дозволяє точніше визначити його параметри, наприклад, кількість каналів (моно, стерео, багато каналів), частоту дискретизації (кількість вибірок на секунду) та кількість вибірок, що відповідають тривалості аудіофайлу. Крім основних аудіо-даних, у файлі AIFF можуть зберігатися додаткові блоки, які дозволяють зберігати корисну інформацію про сам файл, наприклад, ім’я виконавця, альбом, жанр та інші метадані. Це робить формат AIFF дуже зручним для зберігання не тільки аудіо, але й супутніх даних, які можуть використовуватися для пошуку та каталогізації

аудіофайлів. Тому AIFF є ідеальним форматом для застосування в професійних студіях, де важливо мати можливість швидко знайти певний трек за його метаданими.

Однією з основних переваг AIFF є здатність зберігати аудіо в незжатому вигляді, що забезпечує високу якість звуку, а також створює додаткові можливості для використання стеганографії. Стеганографія в аудіофайлах дозволяє приховувати різні типи інформації, таких як текстові повідомлення, ключі доступу або інші важливі дані, у самій аудіо-інформації.

Використання стеганографії в аудіоформатах високої якості AIFF, дозволяє виконувати це приховування без значних втрат у якості відтвореного звуку. Метод фазового кодування, про який згадувалося раніше, є одним з найбільш ефективних для цього формату. Він дозволяє інтегрувати дані у фази звукових хвиль, де зміни будуть практично непомітними для людського слуху. Враховуючи, що AIFF підтримує збереження даних на високому рівні точності, цей формат є ідеальним кандидатом для приховання даних у звукових хвилях без помітних змін в якості. Однак, важливо зазначити, що AIFF має певні обмеження в контексті застосування стеганографії. Оскільки цей формат не використовує стиснення даних, кожен біт аудіо містить максимальну кількість інформації, що дає більше простору для маніпулювання звуковими даними. Проте відсутність стиснення також означає, що файли AIFF займають більше місця на диску, що може стати проблемою для користувачів, яким потрібно зберігати великий обсяг даних [11].

Також, AIFF не є таким популярним у повсякденному використанні, як інші аудіоформати, що обмежує його застосування в широкому спектрі користувацьких пристроїв, таких як смартфони або портативні плеєри. Враховуючи це, методи стеганографії, які використовують AIFF, можуть бути більш корисними в спеціалізованих професійних додатках, де важлива точність і якість звуку.

FLAC, в свою чергу, є популярним форматом для стиснення аудіо без втрат, що дозволяє зберігати високу якість звуку при значному зменшенні обсягу файлу. Хоча FLAC не зберігає стільки метаданих, як AIFF, його основною перевагою є компактний розмір файлів, що робить його привабливим для користувачів, які

прагнуть зберігати великий обсяг аудіофайлів при збереженні високої якості звуку. У зв'язку з тим, що AIFF зберігає аудіо в незжатому вигляді, його перевага полягає у високій точності збереження звукових даних, що є важливим для професійних студій. Цей формат надає більше можливостей для маніпулювання даними, а також для використання стеганографічних методів без значної втрати якості.

Аудіоформат AIFF є одним із найкращих виборів для збереження високоякісних аудіофайлів завдяки його незжатій структурі та високій точності відтворення звуку. Його бінарна структура дозволяє ефективно зберігати як звукові дані, так і метадані, що робить його зручним для застосування в професійних умовах.

Формат AIFF є ідеальним для використання стеганографічних методів, таких як фазове кодування, для приховування даних у аудіофайлах без значних змін у якості звуку. Завдяки своїй універсальності та точності збереження звукових даних, AIFF продовжує залишатися важливим форматом в індустрії, незважаючи на конкуренцію з іншими форматами, такими як WAV і FLAC. Він залишається популярним у сфері професійного звукозапису та пост-продакшн, а також є важливим інструментом для реалізації методів стеганографії для захисту авторських прав та збереження конфіденційної інформації в аудіофайлах.

1.4 Аналіз роботи стеганографічного методу фазового кодування для аудіофайлів

Метод фазового кодування є одним із найбільш ефективних підходів стеганографії для приховування інформації в аудіофайлах. Цей метод працює шляхом зміни фази звукової хвилі для вбудовування прихованих даних, що дозволяє мінімізувати вплив на сприйняття звуку людиною. Завдяки своїй здатності заховати інформацію в аудіофайлі без істотних змін в якості звукового сигналу, фазове кодування набуває особливої популярності у сфері захисту авторських прав, а також у інших додатках, де конфіденційність і цілісність даних є критично важливими. Аудіофайл, зокрема в таких форматах, як AIFF або WAV, складається з числових значень, що відображають амплітуду звукової хвилі в певні моменти часу.

Відмінність методу фазового кодування від інших методів стеганографії полягає в тому, що він не змінює амплітуду або частоту аудіосигналу, а фокусується на зміні фази. Фаза хвилі визначає її точку початку і закінчення в одному циклі, і саме ця величина змінюється для приховування даних. Завдяки тому, що зміни фази не сприймаються людським слухом так чітко, як зміни в амплітуді або частоті, фазове кодування дозволяє ефективно маскувати вбудовану інформацію [12].

Один з важливих аспектів методу полягає в тому, що фаза сигналу в аудіофайлі може бути змінена таким чином, що в результаті зміни не відбудеться суттєвого спотворення звучання. Це особливо важливо в контексті високоякісних звукових записів, де навіть незначні зміни можуть бути помітні слухачеві. Порівняно з іншими методами стеганографії, такими як методи, що маніпулюють амплітудою або частотою, фазове кодування дозволяє зберегти значно вищу якість звуку. Після того, як аудіофайл переведений в цифровий вигляд, наступним етапом є підготовка інформації, яку планується вбудувати. Цю інформацію потрібно перетворити у бінарну форму, тобто у вигляд нулів і одиниць, щоб вона могла бути закодована в аудіосигнал. Для цього вибираються певні сегменти аудіофайлу, які будуть модифіковані для вбудовування цих бітів. Важливою особливістю є те, що зміни фази здійснюються лише в межах тих частин звукової хвилі, де зміни не будуть чутні слухачеві. Це досягається завдяки точному вибору елементів сигналу, в яких мінімальні амплітудні зміни, що дозволяє безболісно вбудовувати дані без помітних спотворень.

Власне вбудовування даних в аудіофайл здійснюється шляхом зміни фази окремих вибірок у файлі. Кожен біт прихованої інформації, що вбудовується, відповідатиме зміні фази в певній вибірці. Для цього використовуються алгоритми, які визначають оптимальні моменти для зміни фази, щоб ці зміни не були помітні для слухача. Водночас система повинна бути розрахована на те, щоб приховані дані можна було ефективно витягнути на етапі декодування [13].

Метод фазового кодування має декілька суттєвих переваг, які роблять його дуже популярним у стеганографії. По-перше, зміна фази звукової хвилі є найменш помітною для людського слуху в порівнянні з іншими видами змін у звуковому

сигналі, такими як зміни амплітуди або частоти. Це дозволяє приховувати велику кількість даних без значних спотворень звукового сигналу. Крім того, використання форматів без стиснення, таких як AIFF чи WAV, дозволяє зберігати високу якість звуку, оскільки ці формати не призводять до втрати інформації через стиснення. Метод фазового кодування є потужним і ефективним способом стеганографії для аудіофайлів, який дозволяє приховувати дані без значних змін у якості звуку. Його основна перевага полягає в тому, що зміни в фазі звукової хвилі не сприймаються слухачем, що робить цей метод ідеальним для застосування в контексті захисту авторських прав на аудіо. Хоча метод має обмеження щодо кількості вбудовуваних даних і може бути чутливим до обробки файлів, його здатність зберігати високу якість звуку робить його ідеальним для використання в стеганографії високоякісних аудіофайлів.

1.5 Висновки та постановка задачі

У цьому розділі було розглянуто основні аспекти захисту авторських прав на цифрові аудіозаписи з використанням стеганографії, а саме метод фазового кодування для аудіоформатів високої якості, таких як AIFF. Досліджено теоретичні основи стеганографії, її роль у забезпеченні конфіденційності та автентичності цифрових медіа-файлів, а також можливості застосування цього методу для захисту авторських прав. Приділено особливу увагу характеристикам формату AIFF, його структурі та бінарним складовим, що дозволяє ефективно впроваджувати стеганографічний метод без втрати якості аудіо.

На основі проведеного аналізу в рамках подальшої роботи необхідно вирішити низку задач, зокрема: розробити детальний алгоритм стеганографічного метод фазового кодування, оптимізуючи його для застосування до аудіоформату AIFF з урахуванням характеристик цього формату. Реалізувати програмний засіб для впровадження методу фазового кодування в аудіофайли високої якості, забезпечивши високу точність і надійність приховування інформації.

2 РОЗРОБКА АЛГОРИТМУ РОБОТИ ПРОГРАМИ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ ВИСОКОЇ ЯКОСТІ AIFF НА ОСНОВІ МЕТОДУ ФАЗОВОГО КОДУВАННЯ

2.1 Обґрунтування підходу до використання методу фазового кодування

Розробка ефективного програмного забезпечення для захисту авторських прав аудіофайлів студійної якості передбачає не лише технічну реалізацію алгоритму, але й глибоке обґрунтування вибору методу, який стане основою для приховання інформації. У сучасних умовах цифрової дистрибуції аудіоматеріалів проблема нелегального копіювання та розповсюдження є надзвичайно гострою. Це обумовлює потребу у створенні таких технологій, які могли б надійно, непомітно й без шкоди для якості сигналу вбудовувати захисну інформацію, зокрема, ідентифікатори правовласника або унікальні маркери. У межах проведеного аналізу доцільно розглянути типові характеристики найпоширеніших методів аудіо-стеганографії, які застосовуються для приховування інформації у звукових сигналах. Аудіо-стеганографічні методи можуть суттєво відрізнятися за принципом дії, рівнем складності, якістю відтворення звуку після модифікації та ступенем захищеності інформації від несанкціонованого вилучення. Саме тому для об’єктивного порівняння необхідно опиратись на комплексну оцінку, що базується на низці ключових метрик.

До основних параметрів, які зазвичай враховуються при порівнянні методів стеганографії, належать:

– непомітність (perceptibility) – здатність алгоритму зберігати високу якість сигналу без помітного впливу на слухове сприйняття;

– стійкість (robustness) – здатність прихованої інформації зберігатись навіть після обробки або часткової деструкції аудіо (наприклад, стисненням, шумом або перетворенням формату);

– обсяг вбудованих даних (payload) – кількість інформації, яку можна вставити без втрати якості сигналу;

– складність реалізації (complexity) – рівень технічних знань, ресурсів та часу, необхідних для розробки і впровадження методу;

– швидкодія (speed) – ефективність алгоритму з точки зору часу кодування та декодування повідомлення.

Метод фазового кодування є одним із найбільш збалансованих варіантів у сфері захисту авторських прав на аудіофайли. Його головна перевага полягає у тому, що модифікації відбуваються в фазовій області сигналу, яка менш помітна для слуху людини. Завдяки цьому досягається висока непомітність навіть при незначному компромісі щодо обсягу вбудованих даних. Крім того, фазове кодування демонструє хорошу стійкість до цифрових атак, зокрема до реверсивного перетворення сигналу або перешкод у спектрі [14].

Для наочності нижче наведено порівняльну таблицю (таблиця 2.1) найпоширеніших методів аудіо в стеганографії, які використовуються в цифрових звукових файлах.

Таблиця 2.1 – Порівняння основних методів стеганографії роботи з аудіо

Метод стеганографії	Непомітність	Стійкість	Обсяг даних	Складність реалізації	Швидкодія
LSB (Least Significant Bit)	Висока при правильному використанні	Низька (вразливий до компресії та фільтрації)	Високий	Низька	Висока
Ехо-кодування	Середня	Середня	Середній	Середня	Середня
Спектральне кодування	Висока	Висока	Низький	Висока	Низька
Фазове кодування	Дуже висока	Висока	Низький–середній	Середня	Середня

З огляду на специфіку задачі – приховане вбудовування авторських міток у студійні аудіофайли без втрати якості – метод фазового кодування виступає одним із найбільш перспективних підходів. Його ключова особливість полягає у використанні фазового спектру звукового сигналу, що, на відміну від амплітудних методів, дозволяє зберігати звукову достовірність навіть при значних

перетвореннях спектру. Це особливо важливо для форматів високої якості, таких як AIFF, де будь-які спотворення можуть бути помітними як на слух, так і при спектральному аналізі.

Метод фазового кодування заснований на властивості людського слуху бути менш чутливим до змін фази у звукових сигналів. Це дає змогу змінювати фазову компоненту сигналу для передавання інформації без помітної деградації якості. В стеганографії цей підхід дозволяє приховати повідомлення, змінюючи фази спектральних компонентів у певних блоках сигналу. Такий підхід забезпечує високий рівень непомітності, що особливо актуально для застосування у сфері захисту авторських прав. На відміну від LSB-методів, які є простими у реалізації, але надзвичайно вразливими до атак, фазове кодування забезпечує значно вищу стійкість. Це досягається за рахунок того, що більшість алгоритмів обробки сигналу (наприклад, компресія або фільтрація) у першу чергу впливають на амплітуду, а не фазу сигналу. Таким чином, вбудована інформація у фазі зберігається навіть після певних змін у сигналі, що робить цей метод надійним для довготривалого зберігання даних. Додатковою перевагою фазового методу є складність його виявлення. Більшість алгоритмів аналізу стеганографії орієнтовані на пошук аномалій в амплітудній складовій сигналу. Зміни у фазі менш передбачувані та не створюють очевидних патернів, які могли б бути виявлені автоматичними аналізаторами. Це підвищує загальний рівень захисту вбудованого повідомлення. Однак при всіх перевагах метод фазового кодування має і певні обмеження. Зокрема, обсяг даних, який може бути вбудований без помітного впливу на якість сигналу, є меншим порівняно з деякими іншими підходами. У випадку із захистом авторських прав це не є критичним недоліком, оскільки зазвичай достатньо передати лише унікальний ідентифікатор або цифровий водяний знак, а не великі обсяги інформації [15].

Ураховуючи всі перелічені фактори, можна зробити висновок, що метод фазового кодування є найбільш доцільним вибором для поставленої задачі. Він забезпечує оптимальний баланс між непомітністю, стійкістю до атак, обсягом вбудованих даних і складністю реалізації. Саме тому даний метод було обрано як

основу для побудови алгоритму захисту авторських прав у студійних аудіозаписах формату AIFF. Далі розглядатиметься детальна розробка алгоритму, що реалізує описаний метод із урахуванням специфіки формату AIFF.

2.2 Розробка алгоритму інтеграції фазового кодування у формат AIFF

Інтеграція цифрового водяного знаку у звуковий сигнал з використанням фазового кодування є складним процесом, який вимагає детального аналізу як характеристик аудіосигналу, так і обраного формату збереження – у цьому випадку, це формат аудіо AIFF.

AIFF (Audio Interchange File Format) – це класичний формат цифрового аудіо, який зберігає звукову інформацію без стиснення з втратами у формі лінійного PCM (Pulse-Code Modulation). Такий підхід дозволяє зберегти точність оригінального сигналу на рівні, наближеному до студійної якості. Це є ключовим для методів стеганографії на основі фазового кодування, оскільки забезпечує спектральні перетворення без втрати деталей і ризику пошкодження фазової інформації. Формати з втратами, як-от MP3, суттєво спотворюють фази гармонік під час стиснення (рис. 2.1), що унеможливлює стабільне вбудовування або вилучення цифрового водяного знаку. Натомість AIFF гарантує стабільність частотних компонент протягом усього життєвого циклу файлу, що є критично важливим для алгоритмів, заснованих на точних розрахунках фазових зсувів.

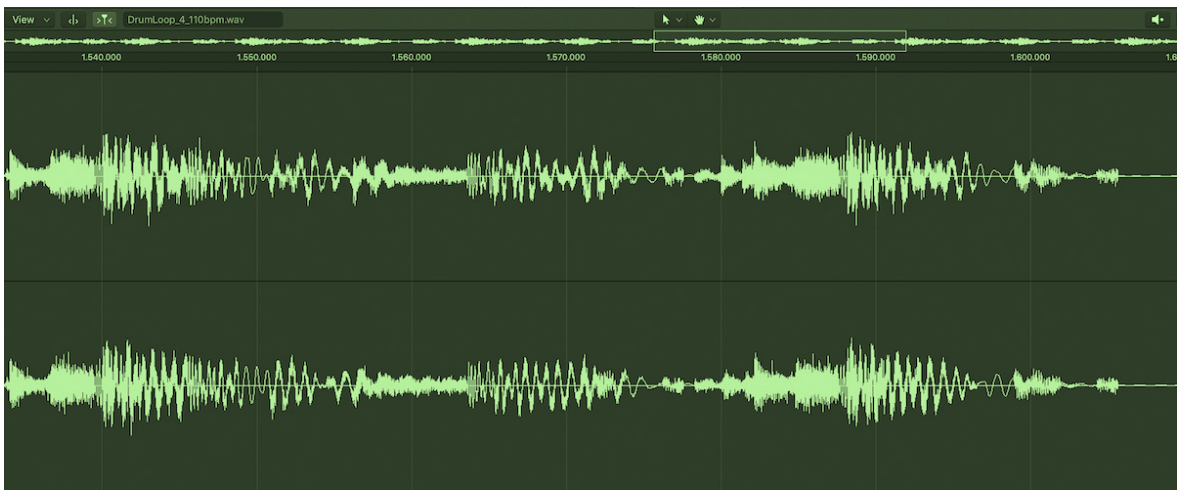


Рисунок 2.1 – Порівняння спотворень після обробки фаз двох звукових хвиль файлів AIFF і MP4 форматів

Крім технічних переваг, AIFF має широке поширення в професійній сфері, особливо у студіях звукозапису, кіноіндустрії, телебаченні та ігровій розробці. Його підтримують популярні цифрові аудіо робочі станції (DAW), такі як Pro Tools, Logic Pro, Cubase та інші. Це робить AIFF логічним і практичним вибором для розробки програмного забезпечення, призначеного для захисту авторських прав на студійні аудіоматеріали.

Першим кроком алгоритму є зчитування звукового сигналу з AIFF-файлу та поділ його на неперервні фрагменти однакової довжини. Це необхідно для обробки сигналу в частотній області, яка здійснюється блочно. Кожен блок аудіо аналізується окремо, що дозволяє рівномірно розподілити вбудовану інформацію та уникнути помітних артефактів. Для кожного з таких блоків виконується дискретне перетворення Фур'є (DFT), яке дозволяє подати сигнал у вигляді набору комплексних коефіцієнтів, що відображають його частотну структуру. Реалізація цього перетворення в програмі здійснюється через швидке перетворення Фур'є (FFT), яке є оптимізованою версією DFT і дозволяє значно зменшити обчислювальні витрати. Це необхідно, оскільки модифікація фази виконується у частотній області (рис. 2.2), а для цього застосовується дискретне перетворення Фур'є (DFT). Для кожного блоку xk[n], де k – індекс блоку, обчислюється перетворення Фур'є:

$$X_k[m] = \sum_{n=0}^{N-1} x_k[n] \cdot e^{-j2\pi mn/N}, \tag{2.1}$$

де N – довжина блоку, m – індекс гармоніки, $X_k[m]$ – комплексне значення спектру.

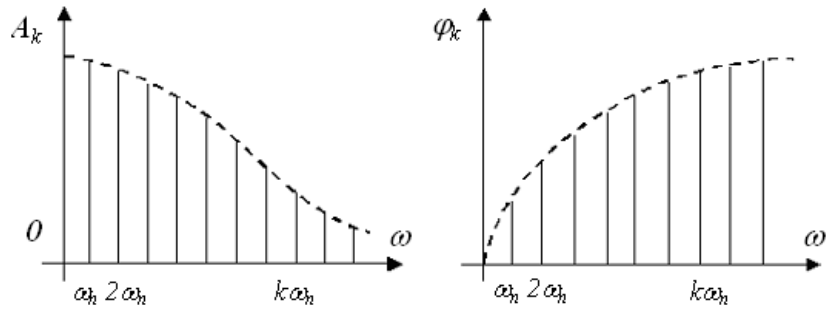


Рисунок 2.2 – Зображення спектра амплітуд і спектра фаз періодичного сигналу

Після обчислення FFT отримано спектр, що складається з модулів і фаз кожної гармоніки. Модуль відповідає інтенсивності звуку на певній частоті, а фаза – за положення хвилі в часі. Саме фазова складова є ключовою в методі фазового кодування. Вона використовується для вбудовування цифрового водяного знаку (ЦВЗ), оскільки людське вухо є менш чутливим до змін фази, ніж до змін амплітуди. Це дозволяє приховано внести додаткову інформацію, не створюючи помітних для слухача змін у звучанні. Спектр розділяється на модуль і фазу:

$$|X_k[m]| = \sqrt{\mathrm{R}^2(X_k[m]) + \mathrm{I}^2(X_k[m])}, \quad \phi_k[m] = \arctan\left(\frac{\mathrm{I}(X_k[m])}{\mathrm{R}(X_k[m])}\right), \qquad (2.2)$$

Перед вбудовуванням повідомлення у фазу сигналу, текст або двійковий потік, який потрібно заховати, проходить попередню обробку. Зазвичай повідомлення кодується у вигляді послідовності бітів (0 та 1). Для підвищення стійкості до змін сигналу або втрат, у повідомлення можуть додаватися контрольні суми, фрейм-синхронізація або навіть застосовуватись елементи кодів виявлення помилок (наприклад, Hamming). Після підготовки даних, вони по одному біту вставляються у фази окремих гармонік аудіоблоків. Для цього використовується принцип керованого фазового зсуву: якщо необхідно передати біт «0», фаза гармоніки залишається незмінною або трохи коригується в заданих межах шуму; якщо ж передається біт «1» – до неї додається фіксоване зміщення $\Delta\phi$, яке попередньо визначається як поріг, достатній для надійного зчитування, але недостатній для того, щоб стати чутливим для людського слуху.

У фазову компоненту $\phi_k[m]$ вбудовується інформація. Кожен біт повідомлення кодується шляхом зміщення фази певної гармоніки на заздалегідь визначене значення. Наприклад, для передачі біта 0 фаза не змінюється, для біта 1 – додається фіксоване зміщення $\Delta\phi$:

$$\phi_k'[m] = \phi_k[m] + b \cdot \Delta\phi, \qquad (2.3)$$

де $b \in \{0,1\}$ – значення біта.

Після внесення змін новий спектр обчислюється як:

$$X_k'[m] = |X_k[m]| \cdot e^{j\phi_k'[m]}, \qquad (2.4)$$

Після модифікації фаз, новий спектр відновлюється шляхом комбінації незмінних модулів і змінених фаз, після чого до кожного блоку застосовується зворотне перетворення Фур'є (inverse FFT). Це дозволяє отримати змінений аудіосигнал, в якому присутня прихована інформація, але який, з акустичної точки зору, залишився незмінним, що дозволяє отримати модифікований блок сигналу:

$$x_k'[n] = \frac{1}{N}\sum_{m=0}^{N-1} X_k'[m] \cdot e^{j2\pi mn/N}, \tag{2.5}$$

Фінальним етапом є об'єднання всіх оброблених блоків у єдиний звуковий потік та збереження результату у новий AIFF-файл. Таким чином, користувач отримує аудіофайл, який є на слух ідентичним оригіналу, але містить прихований ЦВЗ, що може бути використаний для ідентифікації власника, джерела розповсюдження або підтвердження автентичності. Відновлення всієї звукової доріжки здійснюється шляхом збирання всіх модифікованих блоків у єдиний потік. Отриманий AIFF-файл містить приховану інформацію у фазовій компоненті, залишаючись при цьому аудіовізуально ідентичним до оригіналу.

Процес декодування передбачає зворотну послідовність дій. Зчитується модифікований файл, виконується поділ на блоки, кожен з яких перетворюється у спектр. Із фазової компоненти спектру зчитується прихована інформація. Визначення біта базується на величині відхилення фази від очікуваного значення:

$$b = 0, \text{якщо} \phi_k[m] \in [0, \theta], \tag{2.6}$$

$$b = 1, \text{якщо} \phi_k[m] \in [\theta, 2\theta], \tag{2.7}$$

де θ – поріг розмежування між кодуванням 0 і 1 [16].

Якщо було застосовано додаткове кодування чи захист, після зчитування бітів ці процедури також відтворюються, що дозволяє відновити первинне повідомлення.

Для кращого розуміння логіки побудови алгоритму, нижче представлені блок-схеми двох основних процесів: вбудовування та витягання цифрового водяного знаку.

Рисунок 2.3 – Блок-схема алгоритму роботи процесу вбудовування цифрового водяного знаку в AIFF-файл

Процес приховування (вбудовування) цифрового водяного знаку в студійний аудіозапис високої якості (рис. 2.3) складається з 8 кроків.

Крок 1. Зчитування AIFF-файлу. Починається все з етапу завантаження AIFF-файлу. На цьому етапі проводиться перевірка формату, частоти дискретизації та кількості каналів аудіофайлу. Ці характеристики важливі для визначення допустимих меж змін у сигналі, які не порушують якість звуку.

Крок 2. Обчислення хеш-функції початкового AIFF-файлу. Обчислення хеш-функції початкового AIFF-файлу виконується для забезпечення автентичності та цілісності аудіоданих. Перед вбудовуванням водяного знаку програма формує унікальний хеш-код (наприклад, за допомогою алгоритму SHA-256), який генерується на основі вмісту оригінального файлу. Отримане значення додається до водяного знаку у прихованому вигляді. Під час вилучення ЦВЗ хеш дозволяє

перевірити, чи не було змінено аудіофайл з моменту його обробки, що є важливою умовою для захисту авторських прав.

Крок 3. Розбиття інформації на блоки. Після завантаження файл перетворюється на цифровий сигнал, який подається на модуль сегментації. Сегментація означає поділ сигналу на послідовні блоки фіксованої довжини – наприклад, по 1024 або 2048 семплів. Такий поділ дозволяє незалежно обробляти окремі частини аудіо і полегшує подальші спектральні перетворення.

Крок 4. Виконання операції прямого перетворення FFT. Наступним етапом є застосування до кожного блоку прямого швидкого перетворення Фур'є (FFT). Це перетворення дозволяє перейти від часової області до частотної, де сигнал представлений у вигляді набору гармонік з відповідними амплітудами та фазами. У фазовому методі стеганографії саме фази гармонік використовуються як носій інформації.

Крок 5. Модифікація фаз відповідно до бітів ЦВЗ. Далі відбувається кодування повідомлення – цифрового водяного знаку– у фазах певних частотних компонент. Повідомлення попередньо перетворюється у бітову послідовність. Алгоритм обирає певні частотні компоненти в межах низької або середньої частоти, де людське вухо менш чутливе до змін фази. Для кожного біта з повідомлення визначається певний фазовий діапазон. Наприклад, якщо біт дорівнює «0», то фаза відповідної гармоніки може бути змінена у межах від 0 до $\pi/2$; якщо біт – «1», то у межах від π до $3\pi/2$. Ці зміни виконуються таким чином, щоб амплітудна складова сигналу залишалася незмінною або змінювалась мінімально, зберігаючи непомітність модифікацій.

Крок 6. Виконання операції оберненого перетворення FFT. Після вбудовування всіх бітів повідомлення в обрані частотні компоненти кожного блоку виконується обернене швидке перетворення Фур'є (IFFT). Це дозволяє повернутись до часової області, сформувавши модифікований блок сигналу.

Крок 7. Об'єднання блоків у сигнал. Всі оброблені блоки об'єднуються в єдиний потік даних, що створює новий аудіофайл із вже вбудованим водяним знаком.

Крок 8. Збереження нового AIFF-файлу. Фінальним етапом є збереження нового сигналу у форматі AIFF. При цьому забезпечується збереження оригінальної структури заголовку файлу, частоти дискретизації та кількості каналів. Таким чином, отриманий аудіофайл зберігає студійну якість, а вбудований цифровий підпис (ЦВЗ) є невидимим для звичайного прослуховування.

Рисунок 2.4 – Блок-схема алгоритму роботи процесу вилучення цифрового водяного знаку в AIFF-файл

Алгоритм вилучення (зчитування) цифрового водяного знаку (рис. 2.4), який попередньо був вбудований у фазовій області аудіосигналу, складається із 7 послідовних кроків.

Крок 1. Зчитування AIFF-файлу з вбудованим ЦВЗ. Все починається з етапу завантаження AIFF-файлу. Алгоритм перевіряє технічні параметри файлу і підготовлює його до обробки. Як і у схемі вбудовування, сигнал ділиться на блоки фіксованої довжини. Це важливо для того, щоб правильно синхронізуватись із

розміщенням бітів повідомлення у частотній області, оскільки будь-яке зміщення може призвести до спотворення результату декодування.

Крок 2. Розбиття інформації на блоки. Після зчитування аудіофайл трансформується у цифровий сигнал, який передається до модуля сегментації. Аналогічно із алгоритмом вбудовування, сегментація передбачає розбиття сигналу на послідовні блоки фіксованої довжини (наприклад, по 1024 або 2048 семплів), що забезпечує можливість незалежної обробки окремих ділянок сигналу та оптимізує проведення подальших спектральних перетворень.

Крок 3. Виконання операції прямого перетворення FFT. Кожен блок піддається швидкому перетворенню Фур'є (FFT), в результаті чого знову отримується спектральне представлення – набір амплітуд і фаз. Алгоритм визначає ті частотні гармоніки, які використовувались на етапі приховування, та аналізує значення їх фаз.

Крок 4. Зчитування фаз гармонік, які використовувалися для кодування. На основі фазового діапазону для кожної обраної гармоніки визначається відповідний біт: якщо фаза перебуває у межах від 0 до $\pi/2$ – це означає «0», якщо від π до $3\pi/2$ – це «1». Такий аналіз виконується для кожного блоку, послідовно відновлюючи бітову структуру повідомлення.

Крок 5. Порівняння фаз з порогом чутливості. Після вилучення всіх бітів формується бітовий масив, який може бути перетворений назад у текстове повідомлення, числовий ідентифікатор або інший формат залежно від системи автентифікації, що використовується. У разі потреби можуть бути застосовані методи верифікації (наприклад, контрольна сума або криптографічна перевірка), щоб переконатися в цілісності даних.

Крок 6. Об'єднання блоків у сигнал. Після операцій з блоками, файл об'єднується назад в єдиний потік і формується повністю готове приховане повідомлення.

Крок 7. Виведення інформації. Вкінці, після виконання всіх операцій, виводиться кінцевий результат і приховане повідомлення. Схема вилучення багато в чому повторює етапи приховування, але замість модифікації фаз виконується їх

точний аналіз. Основна мета – безпомилково реконструювати первинне повідомлення, зберігаючи при цьому стійкість до можливих спотворень сигналу в результаті передачі або обробки.

Таким чином, детальна реалізація алгоритму дозволяє забезпечити надійне приховування даних у звуковому сигналі з мінімальним впливом на його якість. У наступному підрозділі буде описано, яким чином реалізований алгоритм було інтегровано у програмне забезпечення: як побудовано архітектуру системи, якими засобами здійснено реалізацію та які компоненти взаємодіють між собою під час роботи сервісу.

2.3 Проєктування архітектури програмного забезпечення

Проєктування архітектури дає змогу заздалегідь врахувати важливі аспекти, такі як масштабованість, модульність, повторне використання коду, підтримка, супровід та безпека. У межах даної бакалаврської роботи архітектурне проєктування дозволяє сформувати логічну модель програмного забезпечення, що реалізує процес стеганографічного приховування інформації в аудіофайлах формату AIFF, а також забезпечує взаємодію між модулями обробки сигналу, інтерфейсом користувача, базою даних та алгоритмом фазового кодування [17].

Важливо розуміти як буде відбуватись взаємодія користувача з функціоналом програмного забезпечення, що реалізується через послідовність кроків, відображених у блок-схемі алгоритму користувача (рисунок 2.5). Алгоритм починається з інтерфейсу, який забезпечує вибір дії користувачем залежно від мети використання програми.

Процес вбудовування цифрового водяного знаку розпочинається з вибору відповідного режиму роботи. У разі відсутності файлу з уже вбудованим ЦВЗ користувач здійснює завантаження оригінального аудіофайлу у форматі AIFF. Наступним кроком є введення імені та прізвища автора, які, разом із хеш-сумою вихідного файлу, формують цілісне інформаційне повідомлення для подальшого приховування. Обчислення хеш-функції забезпечує унікальність ідентифікації аудіозапису. Після цього запускається процедура вбудовування з використанням

алгоритму фазового кодування. У результаті формується новий аудіофайл, що містить прихований цифровий водяний знак, який може бути збережений користувачем.

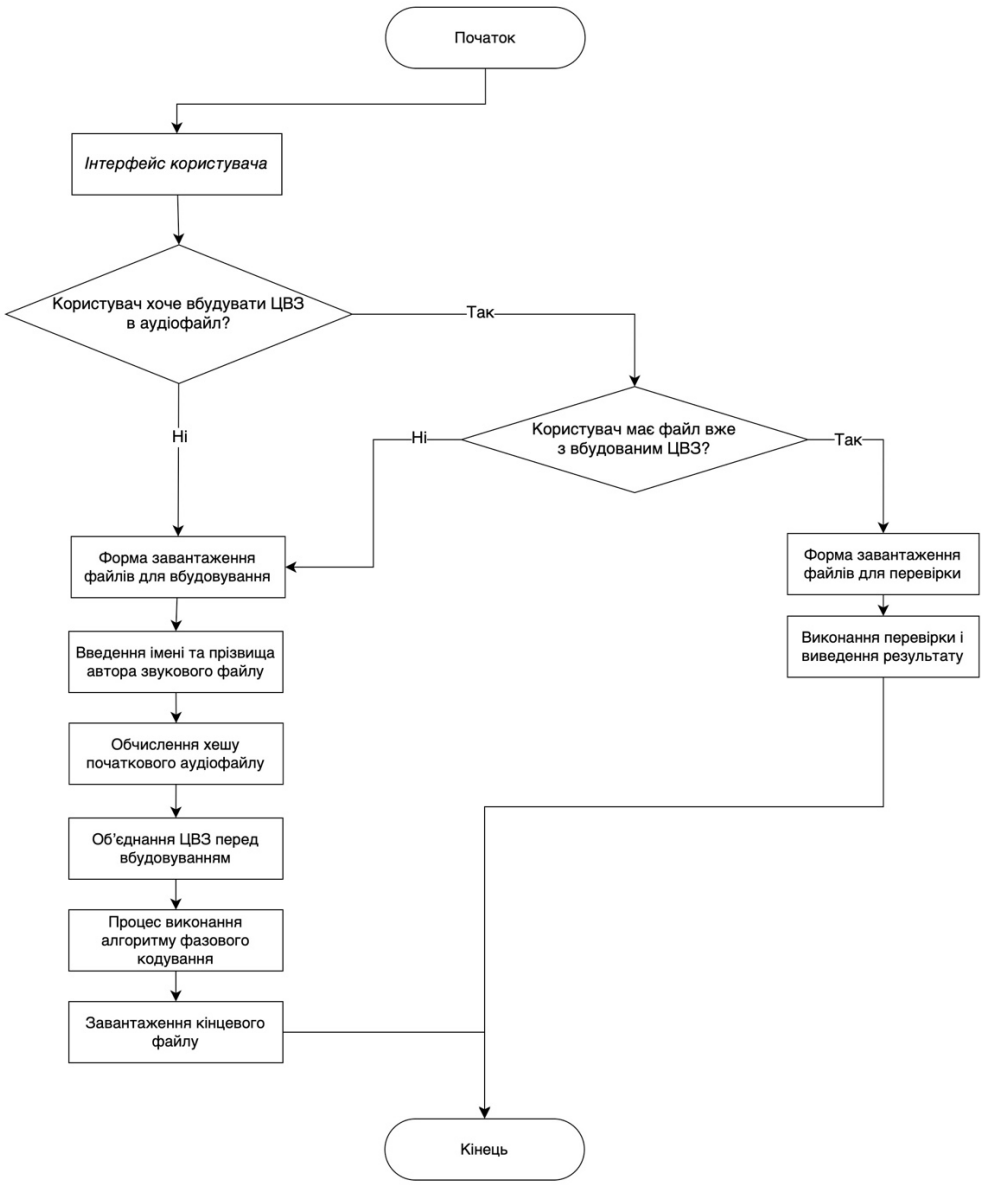


Рисунок 2.5 – Блок-схема алгоритму взаємодії користувача з програмою

У разі наявності у користувача файлу з уже вбудованим цифровим водяним знаком, система переходить до етапу перевірки. Користувач завантажує відповідний аудіофайл, після чого запускається алгоритм аналізу на наявність і коректність цифрового водяного знаку. Результати перевірки виводяться у формі повідомлення, після чого сесія взаємодії з програмою завершується. Таким чином, реалізований алгоритм забезпечує послідовну та зручну роботу як для

вбудовування, так і для перевірки цифрового водяного знаку, сприяючи підвищенню рівня захисту авторських прав. У наступному розділі розглядається архітектура програмного забезпечення та функціонування його ключових компонентів.

Програмне забезпечення складається з кількох основних модулів (рис. 2.6), які виконують специфічні функції, але при цьому перебувають у тісній взаємодії один з одним, що забезпечує цілісність роботи всієї системи. Така модульна архітектура дозволяє не лише чітко розділити обов'язки між компонентами, але й значно полегшує подальший розвиток і підтримку програмного продукту, забезпечуючи при цьому його масштабованість і гнучкість.

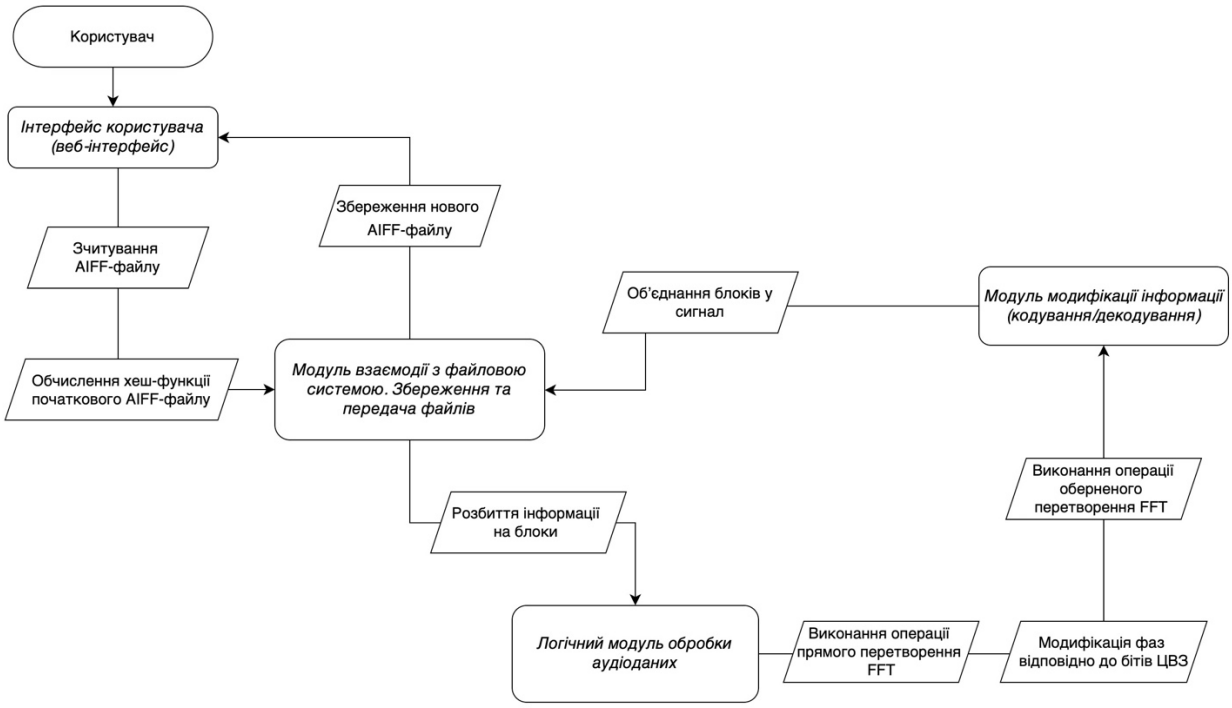


Рисунок 2.6 – Структурна схема взаємодії компонентів архітектури розробленого програмного забезпечення

Перший модуль — веб-інтерфейс, що забезпечує взаємодію користувача з програмою. Він приймає вхідні дані, зокрема AIFF-файли та текстові повідомлення, і дозволяє вибрати режим роботи — кодування або декодування. Інтерфейс розроблено з урахуванням зручності, щоб користувач міг працювати з системою без спеціальних технічних знань. Модуль обробки аудіоданих відповідає за аналіз і

зміну аудіофайлів. Він виконує зчитування файлів, спектральний аналіз через перетворення Фур’є, модифікацію фазової складової сигналу і створення нового файлу зі вбудованою інформацією. Модуль трансформації інформації перетворює текст у бінарний формат, готує дані до вбудовування у фазову складову аудіо і виконує зворотне декодування. Він забезпечує точність і коректність передачі прихованого повідомлення. Модуль роботи з файловою системою та базою даних зберігає оброблені аудіофайли, гарантує їхню цілісність і безпеку, а також веде журнал усіх операцій. Це дозволяє аналізувати ефективність роботи системи і сприяє подальшому розвитку, наприклад, впровадженню автентифікації або збереженню статистики.

Реляційна база даних була розроблена для підтримки функціональності системи стеганографії аудіозаписів, має складну, але водночас чітко структуровану модель, яка забезпечує зберігання та управління всією необхідною інформацією про аудіофайли, алгоритми стеганографії та операції приховування та вилучення даних. Для кращого розуміння структури бази даних та її взаємодії в межах системи була створена UML-діаграма класів (рис. 2.7), яка ілюструє основні сутності, їх атрибути та реляційні зв’язки між ними [19].

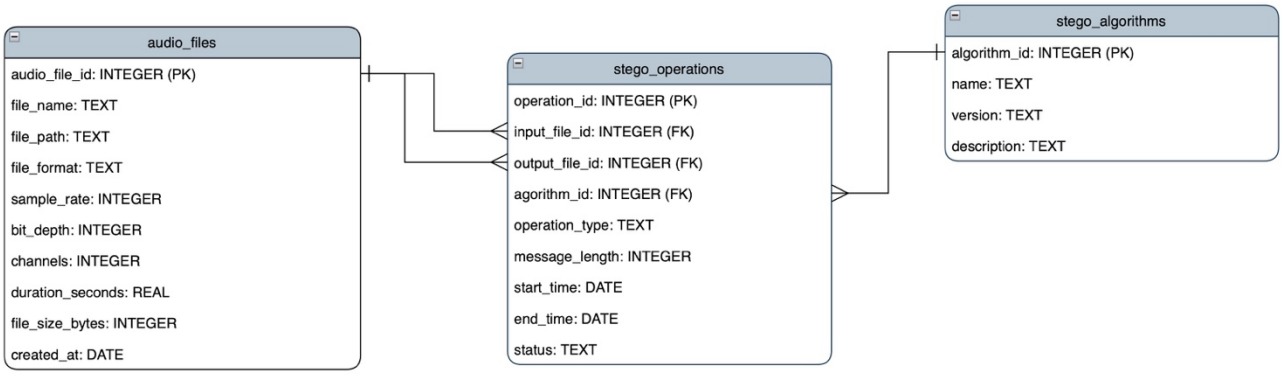


Рисунок 2.7 – UML-діаграма бази даних розроблюваного ПЗ

Першою ключовою сутністю є таблиця audio_files, яка відповідає за збереження метаданих про аудіофайли, що використовуються в процесах стеганографії. Кожен запис у цій таблиці має унікальний ідентифікатор (audio_file_id), який слугує первинним ключем, забезпечуючи однозначну

ідентифікацію файлу в системі. Крім цього, зберігаються такі атрибути, як назва файлу, повний шлях до розташування файлу в файловій системі, формат аудіо (наприклад, AIFF), параметри аудіо (частота дискретизації, бітова глибина, кількість каналів), тривалість аудіозапису у секундах, розмір файлу в байтах, а також часовий штамп створення запису. Така деталізація атрибутів дозволяє системі зберігати повний опис фізичних і логічних властивостей аудіоматеріалів, що є основою для подальшої обробки та аналізу.

Другий основний компонент бази даних – таблиця stego_algorithms, яка описує використовувані в системі методи стеганографії. Кожен алгоритм має власний унікальний ідентифікатор (algorithm_id), а також атрибути, що містять назву алгоритму, версію реалізації та текстовий опис, що пояснює суть або специфіку методу. У поточній реалізації системи основним алгоритмом є фазове кодування (PhaseCoding), що детально відображено в таблиці. Це дозволяє не лише реєструвати, яким способом було виконано приховування інформації, а й забезпечує можливість розширення системи в майбутньому за рахунок додавання нових алгоритмів.

Третім і найважливішим елементом є таблиця stego_operations, яка фіксує історію виконаних операцій кодування і декодування. Кожна операція має унікальний ідентифікатор (operation_id) і містить посилання на вхідний (input_file_id) та вихідний (output_file_id) аудіофайли через зовнішні ключі, що вказують на відповідні записи у таблиці audio_files. Це забезпечує відстеження того, який саме аудіофайл було використано як джерело, а який отримано після застосування стеганографічного алгоритму. Крім того, у таблиці зберігається ідентифікатор алгоритму (algorithm_id), що застосовувався при виконанні операції, тип операції (encode або decode), довжина прихованого повідомлення (у випадку кодування), а також часові позначки початку і завершення операції. Поле статусу дозволяє фіксувати результат виконання (успішно або з помилкою), що є важливим для моніторингу та налагодження системи [20].

В UML-діаграмі класів реляційні зв’язки між сутностями відображені через зовнішні ключі і характеризують логічні залежності та асоціації. Зокрема, зв’язок

між stego_operations та audio_files реалізований у двох напрямках: для вхідних і вихідних файлів операції. Така асоціація, з двох напрямків, забезпечує цілісність даних і дозволяє легко виконувати запити для отримання історії обробки конкретного файлу. Зв’язок між операціями та алгоритмами встановлено через посилання на таблицю stego_algorithms, що дозволяє точно ідентифікувати метод стеганографії, який був використаний. Загалом, ця реляційна модель бази даних забезпечує гнучкість, розширюваність і високу ступінь узгодженості даних. Вона дозволяє зберігати не лише безпосередні аудіоматеріали, але й повний журнал дій зі стеганографії, що є важливою умовою для подальшого аналізу, аудиту та підтримки системи. Водночас чітко визначені зв’язки між сутностями відповідають принципам нормалізації даних, зменшуючи надмірність і запобігаючи аномаліям оновлення.

Розроблена UML-діаграма класів та відповідна база даних виступають фундаментом для побудови надійної, масштабованої та прозорої системи стеганографічного захисту аудіозаписів, що сприяє ефективному керуванню інформацією та забезпеченню цілісності процесів приховування і вилучення даних. Таким чином, взаємодія між цими модулями відбувається за заздалегідь визначеним сценарієм. Користувач через інтерфейс передає необхідні файли і параметри, модулі обробки аудіо та інформації виконують відповідні операції кодування або декодування, а модуль файлової системи та бази даних відповідає за збереження результатів, організацію обміну файлами та ведення логів. Така структурована взаємодія забезпечує стабільність, надійність і гнучкість роботи програмного забезпечення загалом.

2.4 Висновки до розділу 2

У розділі представлено процес розробки алгоритму захисту авторських прав студійних аудіозаписів у форматі AIFF на основі фазового кодування. Метод забезпечує високу непомітність, дозволяючи інтегрувати цифровий водяний знак у фазову складову без втрати якості.

Описано етапи кодування і декодування: перетворення Фур’є, розділення сигналу на амплітуду і фазу, вставлення даних у фазовий компонент і зворотне перетворення. Блок-схема ілюструє логіку алгоритму.

Архітектура ПЗ включає модулі: веб-інтерфейс для завантаження AIFF-файлів і вибору режиму, обробки аудіо (спектральний аналіз, фазова модифікація), трансформації повідомлень у бітову форму, а також роботи з файловою системою і базою даних для збереження й обліку метаданих.

Розділ закладає теоретичну і практичну основу для впровадження і вдосконалення системи захисту авторських прав аудіофайлів у форматі AIFF.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ЗАХИСТУ АВТОРСЬКИХ ПРАВ СТУДІЙНИХ АУДІОЗАПИСІВ ФОРМАТУ AIFF НА ОСНОВІ ФАЗОВОГО КОДУВАННЯ

3.1 Обґрунтування вибору мови програмування та технологій розробки

Програмну реалізацію можна розділити на три чіткі логічні частини: backend-частина, frontend-частина і база даних. Кожна з цих складових виконує свою важливу роль у загальній архітектурі програмного забезпечення, забезпечуючи коректну і ефективну роботу системи в цілому. Backend-частина відповідає за обробку логіки, виконання основних алгоритмів та взаємодію з даними, frontend – за забезпечення інтуїтивного інтерфейсу для користувача, а база даних – за надійне зберігання і організацію інформації. У цьому розділі буде детально розглянуто інструменти і технології, які були використані для реалізації кожної з цих частин, а також пояснено, чому саме вони були обрані для проєкту.

Для розробки backend-частини програмного забезпечення було обрано мову програмування Python, що ґрунтується на кількох важливих критеріях, які визначають її переваги у контексті проєкту. Вибір мови програмування – це фундаментальний крок, який впливає на подальшу архітектуру системи, продуктивність, зручність розробки і підтримки, а також на масштабованість програмного продукту. Тому для обґрунтування цього вибору необхідно розглянути і порівняти основні характеристики Python з іншими популярними мовами, які традиційно використовуються для розробки backend-рішень (табл. 3.1), зокрема Java, C++ та JavaScript (Node.js).

Однією з головних переваг Python є його простота і лаконічність синтаксису, що дозволяє значно прискорити процес розробки. Ця мова відома своєю здатністю працювати з широким спектром завдань: від обробки даних і наукових обчислень до веб-розробки та автоматизації. В контексті проєкту, який передбачає реалізацію алгоритмів фазового кодування, обробки аудіосигналів і взаємодії через веб-інтерфейс, така універсальність є критичною. Python легко інтегрується з різними

компонентами системи, забезпечуючи гладку комунікацію між модулями, що значно спрощує архітектуру і полегшує майбутнє розширення функціоналу.

Таблиця 3.1 – Порівняльна таблиця основних інструментів для backend-розробки

Критерій	Python	C++	Java	JavaScript (Node.js)
Швидкість розробки	Дуже висока	Середня (вимагає багато коду)	Середня	Висока
Простота синтаксису	Дуже проста, читабельна	Складний, багато низькорівневих деталей	Помірна, сувора типізація	Помірна, але асинхронність ускладнює
Підтримка наукових бібліотек	Відмінна	Обмежена	Помірна	Обмежена
Можливості роботи з аудіо	Високі, багато готових рішень	Високі, але потребує значних зусиль	Середні	Помірні
Інтеграція з веб-фреймворками	Легка (Flask, Django)	Складна	Добра (Spring)	Відмінна (Express.js)
Продуктивність	Помірна	Дуже висока	Висока	Помірна
Крива навчання	Низька	Висока	Помірна	Помірна
Спільнота та підтримка	Велика, активна	Велика	Велика, корпоративна	Велика, веб-спільнота

Щодо продуктивності, варто зазначити, що хоча C++ традиційно вважається найшвидшою мовою з наведеного списку, Python, завдяки сучасним інструментам і підходам, демонструє достатній рівень швидкодії для більшості завдань. Для обробки аудіо, що є складним процесом обчислення, використання Python дозволяє поєднати ефективність з простотою реалізації. Водночас можливість інтеграції з низько-рівневими мовами, наприклад через розширення на C або Cython, дозволяє оптимізувати критичні ділянки коду [21].

У порівнянні з Java, яка є більш строго типізованою та вимагає значно більшої кількості коду для реалізації подібного функціоналу, Python забезпечує більш гнучкий підхід, що знижує час розробки і підвищує продуктивність команди. Java, хоча й має хорошу підтримку масштабованості і стійкості, у цьому випадку виявляється надмірно складною, особливо з огляду на те, що основний функціонал не потребує надзвичайно високої продуктивності, а важливіше – швидкість розробки та легкість підтримки [22].

Node.js і JavaScript також є популярними рішеннями для backend-розробки, особливо для веб-сервісів, але вони більше орієнтовані на побудову високонавантажених мережевих застосунків з великою кількістю одночасних підключень. Фокус лежить на складній обробці аудіоданих і математичних операціях, де Python пропонує значно ширші можливості завдяки своїй математичній екосистемі, а також більш зрозумілий і зручний інструментарій для обробки таких задач [23].

Ще одним важливим фактором є масштабованість. Python демонструє середній рівень масштабованості, що цілком достатньо для програмного забезпечення середнього рівня складності. Це означає, що програма зможе ефективно розвиватися і підтримуватися навіть при збільшенні навантаження та розширенні функціоналу без необхідності суттєвих змін у базовій архітектурі. Водночас цей баланс між простотою і масштабованістю забезпечує гнучкість розробки [24].

Таким чином, вибір Python для розробки backend-частини обумовлений комплексом факторів: легкістю і швидкістю розробки, гнучкістю, достатньою продуктивністю для задачі, зручністю реалізації математичних алгоритмів, а також можливістю легко масштабувати і підтримувати програмне забезпечення у подальшому. Всі ці характеристики роблять Python найбільш оптимальним рішенням у порівнянні з іншими мовами програмування, які були розглянуті [26].

Після визначення та обґрунтування вибору інструментів для backend-частини, наступним кроком є розгляд технологій, що використовуються для створення frontend-інтерфейсу. Від якості та зручності користувацького інтерфейсу

значною мірою залежить загальне враження від роботи з програмою. Frontend-частина відповідає за взаємодію користувача з системою, прийом вхідних даних і відображення результатів у зрозумілому вигляді. Тому вибір відповідних технологій для розробки цієї частини є ключовим для досягнення високої юзабіліті і стабільності роботи програми.

Для створення frontend-частини було прийнято рішення використовувати стандартний набір веб-технологій – HTML, CSS та JavaScript. Цей вибір був зумовлений прагненням забезпечити максимально простий, водночас гнучкий та кросплатформений інтерфейс користувача. HTML як мова розмітки дозволяє структуровано подати інформацію на сторінці, CSS відповідає за стилізацію, завдяки чому можна формувати приємний для очей та логічно зрозумілий дизайн. JavaScript додає інтерактивність, роблячи інтерфейс живим і адаптивним до дій користувача без необхідності оновлювати сторінку повністю. Використання цих технологій забезпечує найвищий рівень сумісності з усіма сучасними браузерами і пристроями, що дозволяє охопити широку аудиторію користувачів. Користувачам не потрібно встановлювати додаткові програми або розширення – достатньо зайти на веб-сторінку, і вони отримують повний функціонал програми. Така гнучкість підвищує зручність користування програмою в різних умовах, не обмежуючи аудиторію [27].

JavaScript, як мова програмування на стороні клієнта, забезпечує багатий функціонал взаємодії: можливість швидкої перевірки введених даних без додаткових звернень до сервера, динамічне оновлення сторінки, управління виведенням інформації та елементів інтерфейсу у відповідь на дії користувача. Це створює більш комфортний користувацький досвід, роблячи роботу з програмою швидкою та інтуїтивною.

Для зв’язку frontend-частини з було обрано фреймворк Flask на мові Python. Flask є легким і гнучким засобом для створення веб-додатків, який не нав’язує жорсткої структури, що дає можливість адаптувати архітектуру під конкретні потреби проекту. Така простота дозволяє швидко впроваджувати нові функції, підтримувати і розширювати програму без зайвих ускладнень. Використання Flask

у поєднанні з класичними веб-технологіями frontend забезпечує ефективний розподіл ролей: клієнтська частина відповідає за взаємодію з користувачем, а серверна – за обробку аудіофайлів, виконання алгоритмів кодування та декодування, а також взаємодію з базою даних. Такий підхід дає змогу зберігати логіку обробки і важкі обчислення на сервері, при цьому роблячи користувацький інтерфейс швидким і чуйним.

Загалом, вибір frontend на базі HTML, CSS і JavaScript у поєднанні з Flask на backend є збалансованим рішенням, яке поєднує простоту, ефективність, сумісність та гнучкість, що важливо для довготривалої підтримки та подальшого розвитку програмного забезпечення. Такий підхід дозволяє створити сучасний веб-додаток з приємним користувацьким інтерфейсом, що ефективно виконує поставлені задачі.

Що стосується вибору бази даних, у проєкті було прийнято рішення використовувати SQLite, яка є вбудованою реляційною системою управління базами даних. Це рішення базується на багатьох факторах, які були ретельно проаналізовані у контексті особливостей розроблюваного програмного забезпечення. SQLite має низку переваг, що роблять її оптимальним вибором для даної задачі [28].

Перш за все, SQLite не потребує окремого серверного процесу або конфігурації, оскільки вона зберігає всю базу у вигляді одного файлу на диску. Це значно спрощує розгортання і використання системи, оскільки відсутня необхідність встановлювати додаткове серверне ПЗ, налаштовувати мережеву взаємодію, а також керувати складною архітектурою. Головною метою є обробка відносно невеликого обсягу метаданих (інформація про аудіофайли, дані користувача, цифрові підписи), цей аспект стає суттєвим плюсом. Простота інтеграції SQLite з бекендом, написаним на Python, дозволяє зосередитись на реалізації основної функціональності без зайвих технічних ускладнень [29].

Якщо порівнювати SQLite з іншими популярними системами управління базами даних (табл. 3.2), такими як MySQL, PostgreSQL або MongoDB, то варто зазначити кілька ключових відмінностей. MySQL і PostgreSQL – це повноцінні серверні СУБД, які орієнтовані на обробку великих обсягів даних, складні запити

та одночасну роботу багатьох користувачів. Вони вимагають розгортання серверної інфраструктури, налаштування прав доступу, підтримки безпеки на рівні сервера, а також адміністрування. Для проекту, що розробляється, де кількість одночасних користувачів не є великою, а обсяги даних помірні, така складність є зайвою і неефективною.

Таблиця 3.2 – Порівняльна таблиця існуючих аналогів БД

Характеристика	SQLite	MySQL	PostgreSQL	MongoDB
Легкість налаштування	Відмінна	Середня	Середня	Середня
Розмір	Дуже малий	Великий	Великий	Великий
Швидкодія для малих і середніх навантажень	Відмінна	Відмінна	Відмінна	Добра
Масштабованість	Низька	Висока	Висока	Висока
Підтримка транзакцій	Відмінна	Відмінна	Відмінна	Часткова
Складність адміністрування	Дуже низька	Середня	Середня	Середня
Вбудованість у додаток	Відмінна	Немає	Немає	Немає

Щодо MongoDB, яка є документно-орієнтованою базою даних, вона добре підходить для зберігання неструктурованих або напівструктурованих даних. Проте для задач, де потрібно зберігати відносно просту структуровану інформацію, пов’язану з аудіофайлами та їх метаданими, реляційна модель є більш логічною та зрозумілою. Крім того, MongoDB вимагає окремого сервера, що ускладнює налаштування і збільшує ресурси, необхідні для підтримки роботи програми [30].

SQLite підтримує повний набір функцій реляційної бази даних, включно з транзакціями, що гарантують цілісність даних навіть у випадку збоїв або аварійного завершення роботи. Це особливо важливо для програми, яка працює з цифровими підписами та іншими критично важливими даними, де втрата або пошкодження інформації неприпустимі. У разі потреби у підтримці більшої кількості користувачів або збільшення обсягів оброблюваних даних, завжди існує можливість переходу на більш потужні серверні СУБД. На початковому етапі ж

вибір SQLite забезпечує оптимальне співвідношення між простотою розробки, продуктивністю і надійністю [31].

Таким чином, вибір SQLite був зумовлений її простотою впровадження, мінімальними вимогами до системних ресурсів, високою надійністю, продуктивністю для невеликих обсягів даних, а також зручністю інтеграції з бекендом, написаним на Python. Всі ці фактори в сукупності роблять SQLite найбільш доцільним та ефективним рішенням для збереження та обробки інформації в межах розробленої системи.

3.2 Реалізація ключових логічних модулів програмного забезпечення

Реалізація програмного забезпечення поділена на кілька логічних частин, кожна з яких виконує певний набір функцій, що у сукупності формують повний цикл обробки аудіоданих. Це дозволяє підвищити модульність системи, полегшити її подальшу підтримку та розширення функціоналу. У цьому підрозділі розглядаються такі основні модулі: користувацький інтерфейс, модуль обробки аудіо, модуль трансформації інформації для вбудовування та вилучення цифрового водяного знаку, а також модуль взаємодії з базою даних.

Backend-програми реалізований на мові Python з використанням мікрофреймворку Flask, що забезпечує ефективну обробку запитів користувача і керування логікою програми. У центрі бекенд-частини лежать функції кодування і декодування аудіофайлів формату AIFF за допомогою методу фазового кодування.

```
app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
ALLOWED_EXTENSIONS = {'aiff'}

def allowed_file(filename):
   return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS
if not os.path.exists(UPLOAD_FOLDER):
   os.makedirs(UPLOAD_FOLDER)
```

Ініціалізація початку роботи програми. У цьому фрагменті коду ініціалізується серверна частина програми з використанням фреймворку Flask, який

забезпечує роботу веб-додатку. Визначається папка для збереження завантажених користувачем файлів – «uploads».

Конфігурація застосунку налаштовує цю директорію як основне місце для збереження. Функція allowed_file виконує перевірку розширення файлу, що завантажується, забезпечуючи прийнятність лише аудіоформату AIFF, який є об’єктом обробки програми. Це запобігає завантаженню файлів небажаних типів і підвищує безпеку та стабільність роботи. Також, перед початком роботи програми, здійснюється перевірка існування папки для зберігання файлів. Якщо директорія не існує, вона створюється автоматично. Це дозволяє уникнути помилок при збереженні та обробці файлів і гарантує коректну роботу серверної частини в процесі прийому і збереження даних від користувача.

Цей фрагмент коду відповідає за обробку HTTP-запиту, який надходить на маршрут /aiffForm-data/<filename>. Його призначення – читання аудіофайлу у форматі AIFF, обробка даних сигналу та формування спрощеного представлення звукової хвилі у вигляді усереднених значень. Першим кроком здійснюється пошук файлу у папці для завантажень.

```
@app.route('/aiffForm-data/<filename>')
def aiffForm_data(filename):
    filepath = os.path.join(UPLOAD_FOLDER, filename)
    if not os.path.exists(filepath):
        return jsonify({'error': 'Файл не знайдено'}), 404
    data, samplerate = sf.read(filepath)
    if len(data.shape) > 1:
        data = data[:, 0]
    samples_per_point = max(len(data) // 500, 1)
    aiffForm = []
    for i in range(0, len(data), samples_per_point):
        chunk = data[i:i+samples_per_point]
        avg = float(np.abs(chunk).mean())
        aiffForm.append(avg)
    return jsonify({'aiffForm': aiffForm})
```

Якщо файл існує, він зчитується, що дозволяє отримати масив аудіоданих і частоту дискретизації. Для обробки підтримується випадок із багатоканальним аудіо, де вибирається лише перший канал, що спрощує подальшу обробку. Щоб зменшити обсяг оброблюваних даних і сформувати зручний для візуалізації або

аналізу масив, аудіодані розбиваються на приблизно 500 сегментів, усереднених по абсолютному значенню. Таке представлення дає змогу відобразити загальну форму сигналу без детального навантаження на ресурси. Результат у вигляді списку середніх значень повертається клієнту у форматі JSON, що забезпечує просту інтеграцію з фронтендом для подальшої візуалізації чи аналізу. Цей модуль виконує роль проміжного перетворення сирого аудіосигналу у зручний для користувача та системи формат.

Далі відбувається реалізація маршруту /encode, який приймає POST-запит для кодування аудіофайлу. Спочатку перевіряється, чи надіслані всі необхідні дані: сам аудіофайл, ім’я та прізвище автора. Якщо якісь дані відсутні, користувач повертається на головну сторінку. Перевіряється, чи файл має коректне ім’я і чи відповідає формат AIFF. Якщо формат неправильний або ім’я файлу порожнє, повертається повідомлення про помилку.

```
@app.route('/encode', methods=['POST'])
def encode():
    if 'audio_file' not in request.files or 'author_name' not in request.form or 'author_surname' not in request.form:
        return redirect(url_for('index'))
    audio_file = request.files['audio_file']
    author_name = request.form['author_name'].strip()
    author_surname = request.form['author_surname'].strip()
    if audio_file.filename == '' or not allowed_file(audio_file.filename):
        return 'Invalid file format. Only AIFF files are allowed.'
    filename = secure_filename(audio_file.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    audio_file.save(filepath)
    data, rate = sf.read(filepath)
    duration = len(data) / rate
    file_size = os.path.getsize(filepath)
    input_file_id = add_audio_file(
        file_name=filename,
        file_path=filepath,
        file_format='AIFF',
        sample_rate=rate,
        bit_depth=None,
        channels=data.shape[1] if len(data.shape) > 1 else 1,
        duration_seconds=duration,
        file_size_bytes=file_size
    )
```

Після цього файл зберігається у заздалегідь визначену папку для завантажень. Наступним кроком аудіодані зчитуються з файлу, обчислюється тривалість запису та його розмір у байтах. Отримані метадані (ім’я файлу, шлях, формат, частота дискретизації, кількість каналів, тривалість і розмір файлу) передаються у функцію, яка зберігає цю інформацію в базі даних і повертає унікальний ідентифікатор запису. Таким чином реалізується обробка вхідних даних і підготовка їх для подальшої роботи з програмою.

У наступному фрагменті коду здійснюється основна частина процесу кодування аудіофайлу за допомогою методу фазового кодування. Спочатку відкривається файл у двійковому режимі, і обчислюється його хеш за алгоритмом SHA-256. Це робиться для формування унікального цифрового підпису файлу, що забезпечує перевірку цілісності та автентичності.

```
with open(filepath, 'rb') as f:
    file_hash = hashlib.sha256(f.read()).hexdigest()
full_message = f"{author_name}{author_surname}&&{file_hash}"

name, ext = os.path.splitext(filename)
unique_suffix = str(int(time.time() * 1000))  # мілісекунди
encoded_filename = f"{name}_s{unique_suffix}{ext}"
encoded_path = os.path.join(app.config['UPLOAD_FOLDER'], encoded_filename)
stego = PhaseEncodingAudioStego()
start_time = datetime.now()
stego.encodeAudio(filepath, full_message, outputPath=encoded_path)
end_time = datetime.now()
data_out, rate_out = sf.read(encoded_path)
duration_out = len(data_out) / rate_out
file_size_out = os.path.getsize(encoded_path)
output_file_id = add_audio_file(
    file_name=encoded_filename,
    file_path=encoded_path,
    file_format='AIFF',
    sample_rate=rate_out,
    bit_depth=None,
    channels=data_out.shape[1] if len(data_out.shape) > 1 else 1,
    duration_seconds=duration_out,
    file_size_bytes=file_size_out
)
```

Далі формується повне повідомлення, яке включає ім’я та прізвище автора, розділені спеціальним символом, та отриманий хеш файлу. Це повідомлення буде заховане у фазовому спектрі аудіосигналу. Після цього створюється унікальна назва для нового закодованого файлу, що містить часовий суфікс у мілісекундах, щоб уникнути перезаписування існуючих файлів.

Ініціалізується об’єкт класу, що відповідає за фазове кодування, і запускається процес вбудовування повідомлення у аудіофайл. Для оцінки продуктивності зберігається час початку і завершення операції.

Після кодування новий файл зчитується для визначення його основних параметрів, таких як тривалість, частота дискретизації і розмір. Ці метадані також зберігаються у базі даних, аналогічно як і для вхідного файлу. Далі визначається ідентифікатор алгоритму кодування і створюється запис у логах про виконану операцію, включно з часом початку та завершення, кількістю вбудованих символів та статусом успіху.

Наступний фрагмент коду відповідає за розшифрування прихованого повідомлення з аудіофайлу формату AIFF, отриманого від користувача. Спочатку перевіряється наявність файлу у запиті та його формат, після чого файл зберігається у вказаній директорії. Ініціалізується об’єкт класу, що реалізує фазове кодування, та виконується операція декодування, при якій вилучається приховане повідомлення з аудіо. Також фіксується час початку і завершення операції для подальшого логування.

```
@app.route('/decode', methods=['POST'])
def decode():
    if 'audio_file' not in request.files:
        return redirect(url_for('index'))

    audio_file = request.files['audio_file']

    if audio_file.filename == '' or not allowed_file(audio_file.filename):
        return 'Invalid file format. Only AIFF files are allowed.'

    filename = secure_filename(audio_file.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    audio_file.save(filepath)
```

Після декодування аудіофайл зчитується для отримання параметрів (тривалість, частота дискретизації, розмір файлу), які зберігаються у базі даних як інформація про вхідний файл. Також створюється запис у логах про виконання операції декодування, де вказується статус успіху та основні метрики. В кінці функції користувачу повертається веб-сторінка з результатом розшифрування, що містить витягнуте повідомлення та назву розшифрованого файлу.

```
        stego = PhaseEncodingAudioStego()
        start_time = datetime.now()
        decoded_message = stego.decodeAudio(filepath)
        end_time = datetime.now()

        data, rate = sf.read(filepath)
        duration = len(data) / rate
        file_size = os.path.getsize(filepath)

        input_file_id = add_audio_file(
            file_name=filename,
            file_path=filepath,
            file_format='AIFF',
            sample_rate=rate,
            bit_depth=None,
            channels=data.shape[1] if len(data.shape) > 1 else 1,
            duration_seconds=duration,
            file_size_bytes=file_size
        )

        algorithm_id = get_algorithm_id_by_name('PhaseCoding')

        log_operation(
            input_file_id=input_file_id,
            output_file_id=None,
            algorithm_id=algorithm_id,
            operation_type='decode',
            message_length=len(decoded_message),
            start_time=start_time.isoformat(sep=' ', timespec='milliseconds'),
            end_time=end_time.isoformat(sep=' ', timespec='milliseconds'),
            status='success'
        )

        return render_template('index.html', decoded_message=decoded_message,
decoded_file=filename)
```

Далі основна логіка фазового кодування. Клас PhaseEncodingAudioStego є спадкоємцем базового класу AudioStego і реалізує метод фазового кодування

аудіоданих для приховування інформації в аудіофайлах. Метод encodeAudio починається з перетворення вхідного аудіофайлу у масив байтів, що є необхідним кроком для подальшої обробки.

```
class PhaseEncodingAudioStego(AudioStego):

    def encodeAudio(self, audioLocation, stringToEncode, outputPath=None) -> str:
        self.convertToByteArray(audioLocation)
        stringToEncode = stringToEncode.ljust(100, '~')
        textLength = 8 * len(stringToEncode)

        blockLength = int(2 * 2 ** np.ceil(np.log2(2 * textLength)))
        blockNumber = int(np.ceil(self.audioData.shape[0] / blockLength))

        if len(self.audioData.shape) == 1:
            self.audioData.resize(blockNumber * blockLength, refcheck=False)
            self.audioData = self.audioData[np.newaxis]
        else:
            self.audioData.resize((blockNumber * blockLength, self.audioData.shape[1]),
refcheck=False)
            self.audioData = self.audioData.T
```

Рядок, який потрібно закодувати, доповнюється символами ~ до довжини 100, щоб забезпечити фіксований розмір вбудованої інформації. Обчислюється довжина тексту у бітах, що впливає на вибір розміру блоку кодування. Для кращої оптимізації застосовується розмір блоку, який є степенем двійки, із запасом для коректної роботи алгоритму. Далі відбувається переформатування масиву аудіоданих, його доповнення до необхідної довжини та транспонування у разі багатоканального запису. Це готує дані для подальшого етапу фазового кодування.

```
        blocks = self.audioData[0].reshape((blockNumber, blockLength))
        blocks = np.fft.fft(blocks)
        magnitudes = np.abs(blocks)
        phases = np.angle(blocks)
        phaseDiffs = np.diff(phases, axis=0)

        textInBinary = np.ravel([[int(y) for y in format(ord(x), "08b")] for x in stringToEncode])
        textInPi = textInBinary.copy()
        textInPi[textInPi == 0] = -1
        textInPi = textInPi * -np.pi / 2

        blockMid = blockLength // 2
        phases[0, blockMid - textLength: blockMid] = textInPi
        phases[0, blockMid + 1: blockMid + 1 + textLength] = -textInPi[::-1]
```

Наступним кроком у методі encodeAudio аудіодані розбиваються на блоки фіксованої довжини, які потім перетворюються за допомогою швидкого перетворення Фур'є (FFT). Це дозволяє перейти у частотну область, де аудіосигнал представлений через амплітуди та фази кожної частотної компоненти. Амплітуди отримують через модуль комплексних чисел, а фази – через їхній аргумент.

Далі обчислюються різниці фаз між сусідніми блоками, що важливо для подальшого кодування інформації у фазових змінних. Повідомлення, яке потрібно закодувати, перетворюється у двійковий формат: кожен символ переводиться у послідовність бітів. Потім біти перетворюються на множину значень +1 і -1, які масштабуються на величину $-\pi/2$ для кодування у фазі.

Закодування відбувається у центральній частині спектру блоку. В обчислених фазах замінюються значення на нові, що відповідають повідомленню: в одному напрямку і дзеркально у зворотному. Це дозволяє приховати інформацію у фазовій структурі сигналу без суттєвого спотворення аудіо.

```
for i in range(1, len(phases)):
    phases[i] = phases[i - 1] + phaseDiffs[i - 1]

blocks = (magnitudes * np.exp(1j * phases))
blocks = np.fft.ifft(blocks).real
self.audioData[0] = blocks.ravel().astype(np.int16)

return self.saveToLocation(self.audioData.T, audioLocation, outputPath)
```

Наступним етапом є відновлення фазових значень для кожного блоку аудіоданих. Це відбувається через послідовне додавання обчислених раніше різниць фаз, що забезпечує коректну фазову структуру сигналу після кодування. Завдяки цьому зберігається зв'язність фаз між сусідніми блоками, що мінімізує спотворення звуку.

Після цього нові фази комбінуються з амплітудами, і виконується обернене швидке перетворення Фур'є (IFFT), щоб повернути сигнал у часову область. Отримані дані округлюються до цілочисельного формату 16-бітного аудіо, після чого збереження відбувається у заданому форматі і шляху. Таким чином, метод не тільки приховує повідомлення у фазовій структурі аудіо, але й повертає

модифікований аудіофайл у звичайний формат, що можна відтворити стандартними плеєрами.

```
def decodeAudio(self, audioLocation) -> str:
    self.convertToByteArray(audioLocation)
    textLength = 800
    blockLength = 2 * int(2 ** np.ceil(np.log2(2 * textLength)))
    blockMid = blockLength // 2

    if len(self.audioData.shape) == 1:
        secret = self.audioData[:blockLength]
    else:
        secret = self.audioData[:blockLength, 0]

    secretPhases = np.angle(np.fft.fft(secret))[blockMid - textLength:blockMid]
    secretInBinary = (secretPhases < 0).astype(np.int8)
    secretInIntCode = secretInBinary.reshape((-1, 8)).dot(1 << np.arange(8 - 1, -1, -1))

    return "".join(np.char.mod("%c", secretInIntCode)).replace("~", "")
```

Метод decodeAudio відповідає за витягування прихованого повідомлення з аудіофайлу. Спочатку відбувається перетворення аудіо в масив чисел, що зручно для обробки. Далі визначається довжина тексту і параметри блоків, які використовувалися під час кодування.

Потім із виділеного блоку аудіоданих обчислюються фази, з яких вибирається частина, де міститься приховане повідомлення. Ці фази конвертуються у двійковий код, де знак фази відображає біт повідомлення. Після цього двійкові дані перетворюються назад у символи, а спеціальні символи-падінки, що використовувалися для вирівнювання, видаляються. В результаті метод повертає оригінальний рядок, який було зашифровано у фазовій структурі аудіо.

```
def convertToByteArray(self, audio):
    try:
        self.audioData, self.rate = sf.read(audio, dtype='int16')
    except Exception as e:
        print("Error reading AIFF:", e)

    self.audioData = self.audioData.copy()
```

Метод convertToByteArray відповідає за зчитування аудіофайлу у форматі AIFF і перетворення його в масив числових даних для подальшої обробки. Він

використовує спеціальну бібліотеку для роботи з аудіо, щоб завантажити файл у вигляді цілочисельного масиву 16-бітного формату. Якщо під час читання виникає помилка, вона фіксується і виводиться у консоль. Після успішного зчитування, аудіодані копіюються у внутрішній стан об’єкта для подальших операцій, щоб зберегти оригінал незмінним.

```
def saveToLocation(self, audioArray, location, outputPath=None) -> str:
    if outputPath is not None:
        output_path = outputPath
    else:
        dir = os.path.dirname(location)
        output_path = os.path.join(dir, "output-pc.aiff")

    if len(audioArray.shape) > 1:
        audioArray = audioArray[:, 0]

    sf.write(output_path, audioArray, self.rate, format='AIFF', subtype='PCM_16')
    return output_path
```

Метод saveToLocation відповідає за збереження оброблених аудіоданих у файл на диску. Він приймає масив аудіосемплів, вихідний шлях до оригінального файлу та необов’язковий параметр для збереження у задане місце. Якщо конкретний шлях не вказаний, файл зберігається у тій же директорії, де знаходиться оригінал, під ім’ям за замовчуванням. Перед записом, якщо аудіодані мають кілька каналів, метод обирає лише перший канал для збереження. Запис виконується у форматі AIFF з 16-бітним PCM кодуванням. Після успішного збереження повертається шлях до створеного файлу.

```
@app.route('/history')
def history():
    encodings = get_all_operations()
    return render_template('history.html', encodings=encodings)

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

Маршрут /history, відповідає за відображення сторінки з історією виконаних операцій кодування та декодування. При зверненні до цього маршруту викликається функція history, яка отримує всі записи з бази даних через функцію get_all_operations(). Отримані дані передаються у шаблон history.html для

виведення користувачу, що дозволяє переглянути журнал виконаних дій у програмі. Далі у блоці if __name__ == '__main__': відбувається ініціалізація бази даних викликом init_db(), після чого запускається сам вебсервер Flask з увімкненим режимом відладки (debug=True). Це забезпечує автоматичне перезавантаження сервера при зміні коду та полегшує тестування і розробку програми.

Ініціалізація бази даних SQLite. Змінна DB_NAME містить назву файлу бази даних – 'app_data.db', де будуть зберігатися всі необхідні дані програми.

```
DB_NAME = 'app_data.db'

def init_db():
  conn = sqlite3.connect(DB_NAME)
  c = conn.cursor()
```

Функція init_db() встановлює з’єднання з цією базою даних, використовуючи бібліотеку sqlite3. Вона створює об’єкт курсора c, який слугує для виконання SQL-запитів до бази даних. Цей початковий крок є базовим для подальшої роботи з базою, зокрема для створення таблиць та інших операцій з даними.

Створення в базі даних таблиці stego_operations, для прикладу, яка зберігає інформацію про виконані операції кодування та декодування аудіофайлів.

```
  c.execute('''
    CREATE TABLE IF NOT EXISTS stego_operations (
      operation_id INTEGER PRIMARY KEY AUTOINCREMENT,
      input_file_id INTEGER NOT NULL,
      output_file_id INTEGER,
      algorithm_id INTEGER NOT NULL,
      operation_type TEXT NOT NULL,
      message_length INTEGER,
      start_time TEXT NOT NULL,
      end_time TEXT,
      status TEXT,
      FOREIGN KEY(input_file_id) REFERENCES audio_files(audio_file_id),
      FOREIGN KEY(output_file_id) REFERENCES audio_files(audio_file_id),
      FOREIGN KEY(algorithm_id) REFERENCES stego_algorithms(algorithm_id)
    )
  ''')
```

Таблиця має такі поля: operation_id – унікальний ідентифікатор операції, який автоматично збільшується; input_file_id – посилання на вхідний аудіофайл, output_file_id – посилання на файл, отриманий після обробки (якщо він є);

algorithm_id – ідентифікатор алгоритму, що був використаний; operation_type – тип операції (кодування чи декодування); message_length – довжина прихованого повідомлення; start_time та end_time – часові позначки початку і завершення операції; status – статус виконання.

Ключі FOREIGN KEY забезпечують зв’язок між цими полями та відповідними записами в інших таблицях бази, зокрема з таблицями аудіофайлів та алгоритмів, що забезпечує цілісність даних у базі. Це дозволяє відстежувати повну історію роботи з файлами та алгоритмами.

Ця частина коду відповідає за перевірку наявності запису про алгоритм фазового кодування в таблиці stego_algorithms.

```
        c.execute("SELECT algorithm_id FROM stego_algorithms WHERE name = ?",
('PhaseCoding',))
        if c.fetchone() is None:
            c.execute('''
                INSERT INTO stego_algorithms (name, version, description)
                VALUES (?, ?, ?)
            ''', ('PhaseCoding', '1.0', 'Phase coding steganography method'))

        conn.commit()
        conn.close()
```

Спочатку виконується пошук алгоритму за назвою 'PhaseCoding'. Якщо такий запис відсутній, то додається новий запис з іменем алгоритму, версією '1.0' і коротким описом. Таким чином забезпечується, що база даних містить інформацію про використовуваний алгоритм, що дозволяє зв’язувати операції кодування та декодування з конкретною реалізацією методу. В кінці операції зміни зберігаються за допомогою commit(), а з’єднання з базою закривається. Це дозволяє уникнути дублювання записів і підтримувати структуру даних у базі.

Add_audio_file() функція відповідає за додавання інформації про аудіофайл до таблиці audio_files у базі даних. Вона приймає параметри, які описують основні характеристики файлу: ім’я, шлях розташування, формат, частоту дискретизації, глибину бітності, кількість каналів, тривалість у секундах та розмір файлу в байтах.

Функція встановлює з’єднання з базою даних SQLite, після чого формує SQL-запит для вставки нового запису з переданими параметрами. Час створення запису

фіксується автоматично у форматі дати і часу. Після успішного додавання даних у таблицю, функція отримує унікальний ідентифікатор доданого файлу, який потім повертає.

```
def add_audio_file(file_name, file_path, file_format=None, sample_rate=None,
bit_depth=None, channels=None, duration_seconds=None, file_size_bytes=None):
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()

    created_at = datetime.now().isoformat(sep=' ', timespec='seconds')

    c.execute('''
      INSERT INTO audio_files (file_name, file_path, file_format, sample_rate, bit_depth,
channels, duration_seconds, file_size_bytes, created_at)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
    ''', (file_name, file_path, file_format, sample_rate, bit_depth, channels, duration_seconds,
file_size_bytes, created_at))

    audio_file_id = c.lastrowid
    conn.commit()
    conn.close()
    return audio_file_id
```

Цей ідентифікатор використовують для подальшої роботи з файлами у базі, зокрема для логування операцій кодування і декодування, що дозволяє відслідковувати пов’язану з аудіофайлами інформацію і підтримувати цілісність даних.

Наступна функція призначена для отримання унікального ідентифікатора алгоритму стеганографії за його назвою з таблиці stego_algorithms у базі даних. Вона встановлює з’єднання з базою SQLite, виконує SQL-запит, який шукає запис з відповідною назвою алгоритму, та отримує перший результат.

```
def get_algorithm_id_by_name(name):
  conn = sqlite3.connect(DB_NAME)
  c = conn.cursor()
  c.execute("SELECT algorithm_id FROM stego_algorithms WHERE name = ?", (name,))
  row = c.fetchone()
  conn.close()
  if row:
    return row[0]
  return None
```

Якщо такий запис існує, функція повертає ідентифікатор алгоритму. У випадку, коли алгоритм з заданою назвою не знайдений, функція повертає значення

None. Це дає змогу програмі перевірити наявність алгоритму у базі перед виконанням операцій, пов’язаних із ним, і уникнути помилок, пов’язаних з відсутністю відповідного запису. Після отримання результату відбувається закриття з’єднання з базою даних.

Функція log_operation відповідає за запис інформації про операції стеганографії у таблицю stego_operations бази даних. Вона приймає параметри, що описують конкретну операцію: ідентифікатори вхідного і вихідного аудіофайлів, ідентифікатор алгоритму, тип операції (наприклад, кодування або декодування), довжину повідомлення, час початку і закінчення операції, а також статус виконання.

```
    def log_operation(input_file_id, output_file_id, algorithm_id, operation_type,
message_length, start_time=None, end_time=None, status='success'):
        conn = sqlite3.connect(DB_NAME)
        c = conn.cursor()

        if start_time is None:
            start_time = datetime.now().isoformat(sep=' ', timespec='seconds')

        c.execute('''
            INSERT INTO stego_operations (input_file_id, output_file_id, algorithm_id,
operation_type, message_length, start_time, end_time, status)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', (input_file_id, output_file_id, algorithm_id, operation_type, message_length, start_time,
end_time, status))

        conn.commit()
        conn.close()
```

Якщо час початку операції не переданий, він автоматично встановлюється на поточний момент. Після підготовки даних виконується SQL-запит для вставки нового запису у таблицю, що дозволяє зберігати історію виконаних операцій. Це важливо для подальшого аналізу, моніторингу та відстеження процесів кодування і декодування аудіо. Після внесення запису log_operation завершує роботу.

Функція get_all_operations виконує вибірку всіх записів про виконані операції стеганографії з бази даних. Вона з’єднує таблиці stego_operations, audio_files (для вхідних і вихідних файлів) та stego_algorithms, щоб отримати повну інформацію про кожну операцію: ідентифікатор операції, назви вхідного та вихідного файлів,

назву алгоритму, тип операції, довжину повідомлення, час початку і закінчення, а також статус виконання.

```
def get_all_operations():
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()
    c.execute('''
        SELECT
            so.operation_id,
            af_in.file_name AS input_file,
            af_out.file_name AS output_file,
            sa.name AS algorithm,
            so.operation_type,
            so.message_length,
            so.start_time,
            so.end_time,
            so.status
        FROM stego_operations so
        JOIN audio_files af_in ON so.input_file_id = af_in.audio_file_id
        LEFT JOIN audio_files af_out ON so.output_file_id = af_out.audio_file_id
        JOIN stego_algorithms sa ON so.algorithm_id = sa.algorithm_id
        ORDER BY so.start_time DESC
    ''')
    result = c.fetchall()
    conn.close()
    return result
```

Запити до бази відсортовані за часом початку операції у спадному порядку, що дозволяє спочатку отримувати найсвіжіші записи. Результат роботи функції – це список кортежів з даними про всі операції, який можна використовувати для відображення історії кодувань і декодувань у веб-інтерфейсі.

У результаті реалізації бекенду та бази даних було створено серверну частину, яка приймає аудіофайли формату AIFF, виконує фазове кодування для приховування інформації та розкодування захованих даних. Бекенд забезпечує збереження оригінальних і оброблених файлів, а також реєструє всі операції в базі даних.

База даних організована так, щоб зберігати інформацію про аудіофайли, використовувані алгоритми стеганографії та історію операцій з детальною інформацією про кожне кодування або декодування. Це дозволяє відстежувати виконані дії та управляти збереженими файлами через веб-інтерфейс.

3.3 Реалізація UI/UX інтерфейсу програмного забезпечення

Інтерфейс складається з головної сторінки, на якій розташовані форми для вибору аудіофайлу та введення імені й прізвища автора при кодуванні, а також кнопки для запуску процесів кодування і декодування. Результати операцій відображаються на тій самій сторінці у вигляді повідомлень або посилань для завантаження готових файлів. Крім того, реалізовано сторінку історії операцій, що дозволяє користувачам переглядати інформацію про всі попередні кодування та декодування. Такий підхід забезпечує простоту у використанні і високу зручність взаємодії з програмою.

```
@app.route('/')
def index():
   return render_template('index.html')
```

Кореневий App.route визначає маршрут кореневого URL-адресу вебдодатку, тобто сторінку за замовчуванням. Коли користувач заходить на головну сторінку, виконується функція index, яка повертає HTML-шаблон index.html. Цей шаблон відповідає за відображення інтерфейсу користувача у браузері. Таким чином, цей маршрут забезпечує первинний доступ до вебінтерфейсу програми, з якого користувач може виконувати подальші дії, наприклад, завантажувати аудіофайли для кодування або декодування.

Далі HTML-код описує частину інтерфейсу для кодування аудіофайлів AIFF за допомогою фазового кодування. Заголовок повідомляє про суть програми, а далі розміщено форму, в якій користувач може вибрати AIFF-файл для кодування. Після вибору файлу показується його назва, хеш і графік (через canvas) форми аудіо. Нижче є два поля для введення імені та прізвища автора, які обов’язкові для заповнення. Кнопка відправляє форму на сервер для обробки [32].

Якщо файл було успішно закодовано, під формою відображається повідомлення з ім’ям збереженого файлу. Такий інтерфейс робить процес простим і зрозумілим для користувача, даючи можливість контролювати введені дані та бачити попередній огляд аудіо.

```
<div class="box-encrypt">
        <h2 class="subheader-encrypt">Кодування</h2>
        <form method="post" action="/encode" enctype="multipart/form-data">
                <div class="custom-file-upload-encrypt">
                <label for="encodeFile">Вибрати AIFF файл для кодування</label>
                <input type="file" name="audio_file" id="encodeFile" required
onchange="handleFileUpload(this)" accept=".aiff">
                <div class="output">
                <span class="file-name" id="encodeFileName">Файл не вибрано</span>
                <div id="fileHash" class="file-hash"></div>
                <canvas id="aiffFormCanvas" width="600" height="120"></canvas>
                        </div>
                </div>
                <input type="text" name="author_name" placeholder="Ім’я автора" required>
                <input type="text" name="author_surname" placeholder="Прізвище автора"
required>
                <button class="submit-button" type="submit">Закодувати</button>
        </form>
        {% if encoded_file %}
        <div class="output-result">Файл збережено: {{ encoded_file }}</div>
        {% endif %}
</div>
```

Далі блок коду відповідає за реалізацію інтерфейсу користувача для функції декодування аудіофайлів формату AIFF. У ньому визначено форму, яка дозволяє користувачу вибрати файл для розкодування, а також кнопку для відправлення запиту на сервер.

```
<div class="box-decrypt">
        <h2 class="subheader-decrypt">Декодування 🔓</h2>
        <form method="post" action="/decode" enctype="multipart/form-data">
        <div class="custom-file-upload-decrypt">
                <label for="decodeFile">Вибрати AIFF файл для декодування</label>
                <input type="file" name="audio_file" id="decodeFile" required
onchange="updateFileName(this)" accept=".aiff">
                <span class="file-name" id="decodeFileName">Файл не вибрано</span>
                </div>
                <button class="submit-button" type="submit">Розкодувати</button>
                <a href="/history" class="submit-button submit-button-history">Історія
операцій</a>
        </form>
        {% if decoded_message %}
        <div class="output">
                Прихований текст: <br> {{ decoded_message }}
                <canvas id="decodeaiffFormCanvas" width="600" height="120"></canvas>
        </div> {% endif %}
    </div>
</div>
```

Після обробки файлу та отримання прихованого повідомлення, воно виводиться у відповідному полі разом із графічною візуалізацією аудіоформи за допомогою елемента canvas. Також передбачено посилання для переходу до сторінки історії операцій.

Функція drawaiffForm асинхронно отримує дані хвильової форми AIFF файлу з сервера за вказаним ім’ям файлу. Вона перевіряє коректність відповіді і наявність помилок, після чого обробляє отриманий масив амплітудних значень. Далі функція налаштовує параметри «канваса» для малювання, враховуючи роздільну здатність екрану, і готує полотно для візуалізації звукової форми.

```
async function drawaiffForm(filename, canvasId = 'aiffFormCanvas') {
    const response = await fetch(`/aiffForm-data/${filename}`)
    if (!response.ok) {
        console.error('Не вдалося завантажити дані хвильової форми')
        return
    }
        const data = await response.json()
        if (data.error) {
        console.error(data.error)
        return
    }
        const aiffForm = data.aiffForm
        const canvas = document.getElementById(canvasId)
    if (!canvas) {
        console.error(`Canvas з ID ${canvasId} не знайдено`)
        return
    }
        const ctx = canvas.getContext('2d')
        const dpr = window.devicePixelRatio || 1
        const cssWidth = canvas.width
        const cssHeight = canvas.height
        canvas.style.width = cssWidth + 'px'
        canvas.style.height = cssHeight + 'px'
        ctx.scale(dpr, dpr)
        const width = cssWidth
        const height = cssHeight
        const middle = height
        const step = width / aiffForm.length
```

Потім визначається ширина та висота канваса, середня лінія, а також крок між точками візуалізації, виходячи з довжини масиву амплітуд. Ці підготовчі дії

дозволяють коректно відобразити графік хвильової форми на веб-сторінці, забезпечуючи візуальне представлення аудіосигналу для користувача.

У цьому фрагменті коду реалізовано функцію drawFrame, яка відповідає за анімаційне малювання хвильової форми аудіосигналу на «канвасі». Встановлено масштаб посилення амплітуди, що дозволяє зробити візуалізацію більш виразною, а також інтенсивність анімації для плавного коливального ефекту.

```
const amplitudeScale = 2.5
        const animationIntensity = 0.0015
        const color = 'rgba(0, 123, 255, 0.9)'
        let frame = 0
        function drawFrame() {
                ctx.clearRect(0, 0, width, height)
                ctx.beginPath()
                for (let i = 0; i < aiffForm.length; i++) {
                        const x = i * step
                        const waveShift = Math.sin((frame * 0.025) + i * 0.05) *
animationIntensity
                        const sample = aiffForm[i] + waveShift

                        const y = middle - (sample * middle * amplitudeScale)
                        if (i === 0) {
                                ctx.moveTo(x, y)
                        } else {
                                ctx.lineTo(x, y)
                        }
                }

                        ctx.strokeStyle = color
                        ctx.lineWidth = 2.8
                        ctx.shadowColor = 'rgba(0, 0, 0, 0.25)'
                        ctx.shadowBlur = 1.5
                        ctx.stroke()
```

В межах циклу для кожної точки масиву амплітуд розраховується горизонтальна координата і додаткове зміщення по вертикалі, яке створює ефект руху хвилі. Центрування хвильової форми відбувається шляхом зміщення від середини полотна «канваса». Лінія хвилі малюється послідовним з’єднанням точок, а стиль лінії включає заданий колір, товщину та тінь, що додає глибини візуальному представленню.

Цей код відповідає за виклик функції drawaiffForm, яка відображає хвильову форму аудіофайлу після виконання операції кодування або декодування. Якщо

змінна encoded_file містить ім’я файлу, що було збережено після кодування, відповідний скрипт автоматично викликає drawaiffForm для відображення хвильової форми на основному полотні.

```
{% if encoded_file %}
<script>
        drawaiffForm("{{ encoded_file }}")
</script>
{% endif %}
```

Аналогічно, у випадку наявності змінної decoded_file, хвильова форма аудіо відображається на окремому полотні з ідентифікатором decodeaiffFormCanvas. Це забезпечує динамічне оновлення інтерфейсу користувача і надає візуальний зворотний зв’язок про результати фазового кодування та декодування аудіосигналу.

```
{% if decoded_file %}
<script>
        drawaiffForm("{{ decoded_file }}", "decodeaiffFormCanvas")
</script>
{% endif %}
```

Функція updateFileName оновлює текстовий лейбл поруч із полем вибору файлу, відображаючи ім’я обраного файлу або повідомлення «Файл не вибрано», якщо файл не обрано. Ідентифікатор лейблу формується на основі id елемента введення.

```
function updateFileName(input) {
        const fileName = input.files.length > 0 ? input.files[0].name : "Файл не вибрано"
        const labelId = input.id + "Name"
        document.getElementById(labelId).textContent = fileName
}
```

Функція handleFileUpload виконує кілька завдань: спочатку оновлює ім’я файлу за допомогою updateFileName, потім читає вміст вибраного файлу у вигляді масиву байтів.

За допомогою Web Crypto API обчислюється SHA-256 хеш цього масиву, який відображається у інтерфейсі для підтвердження цілісності файлу. Після короткої затримки викликається функція drawaiffForm для побудови графічного відображення хвильової форми аудіо.

```
function handleFileUpload(input) {
	updateFileName(input)
	const file = input.files[0]
	if (!file) return

	const reader = new FileReader()
	reader.onload = async function (event) {
		const arrayBuffer = event.target.result
		const hashBuffer = await crypto.subtle.digest('SHA-256', arrayBuffer)
		const hashArray = Array.from(new Uint8Array(hashBuffer))
		const hashHex = hashArray.map(b => b.toString(16).padStart(2, '0')).join('')
		document.getElementById('fileHash').textContent = "SHA-256: " + hashHex
		setTimeout(() => drawaiffForm(file.name), 500)
	}
	reader.readAsArrayBuffer(file)
}
```

У результаті реалізації frontend частини інтерфейсу було створено зручну веб-сторінку для роботи з фазовим кодуванням аудіофайлів формату AIFF. Користувач отримує можливість завантажувати файли для кодування та декодування, вводити інформацію про автора, а також переглядати результат операцій безпосередньо у браузері. Для покращення взаємодії було реалізовано динамічне відображення назви вибраного файлу, а також обчислення та показ SHA-256 хешу аудіофайлу, що забезпечує перевірку цілісності даних [33].

Крім того, на сторінці відображається графічне представлення хвильової форми аудіо, що додає візуальний контекст і підвищує зручність користування програмою. Таким чином, інтерфейс поєднує простоту, функціональність і естетику, забезпечуючи інтуїтивно зрозумілий досвід для користувача.

3.4 Інструкція користувача та рекомендації щодо використання

Програмне забезпечення має дві основні функції: кодування (вбудовування прихованої інформації в аудіофайл) та декодування (витягнення прихованого повідомлення з файлу). Для початку роботи користувач переходить до запуску програми, де через інтуїтивно зрозумілий інтерфейс (рис. 3.1) може завантажити аудіофайл у форматі AIFF. Цей формат обрано через його популярність у професійному аудіо-виробництві, а також через високу якість збереження звуку, що є важливим для коректного застосування методу фазового кодування.

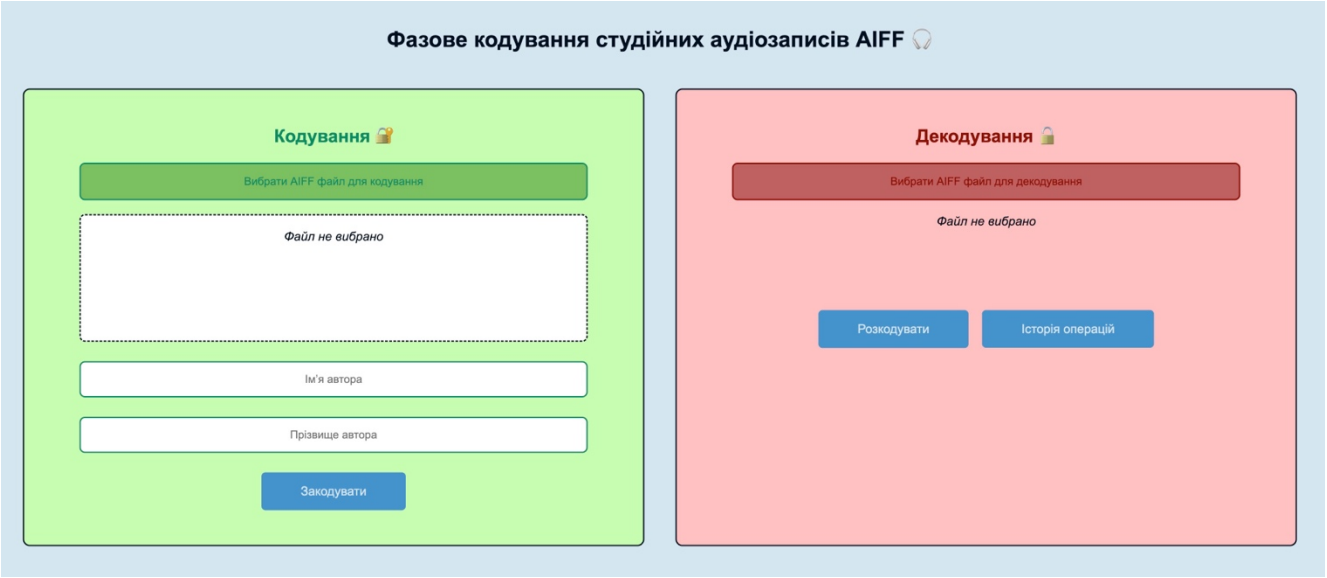


Рисунок 3.1 – Вигляд основної частини інтерфейсу програмного забезпечення

Після завантаження файлу користувач вводить в окремі поля ім’я та прізвище автора, які будуть зашифровані та інтегровані у розмітку аудіо разом із хешем файлу. Це дозволяє підтвердити авторство та перевірити цілісність даних у подальшому. Інтерфейс також відображає хвильову форму аудіофайлу (рис. 3.2), що допомагає користувачу візуально переконатися в правильності вибору файлу.

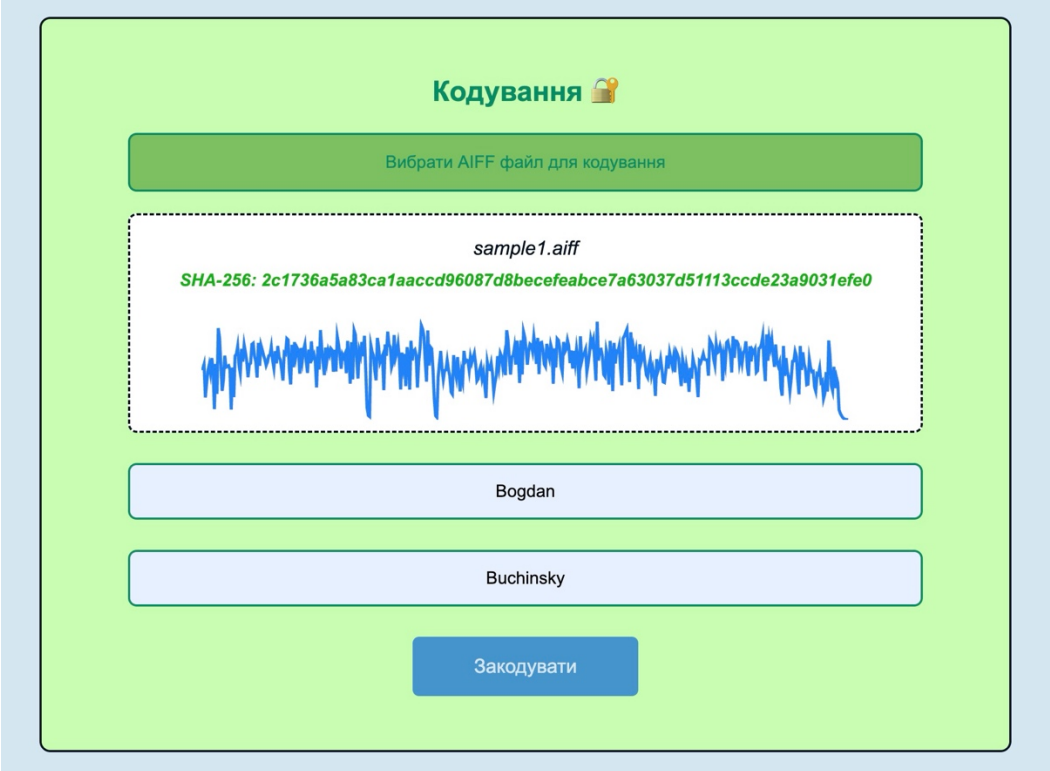


Рисунок 3.2 – Вигляд частини інтерфейсу кодування. Заповнення форми

Після заповнення необхідних полів користувач активує процес кодування, натискаючи відповідну кнопку. Програма виконує фазове кодування, модифікуючи фазову складову спектру аудіосигналу з урахуванням закодованого повідомлення. По завершенню операції генерується новий AIFF-файл, який містить приховану інформацію (рис. 3.3). Користувачу надається повідомлення з інформацією про місце збереження цього файлу, що дає змогу легко його знайти і використовувати надалі.

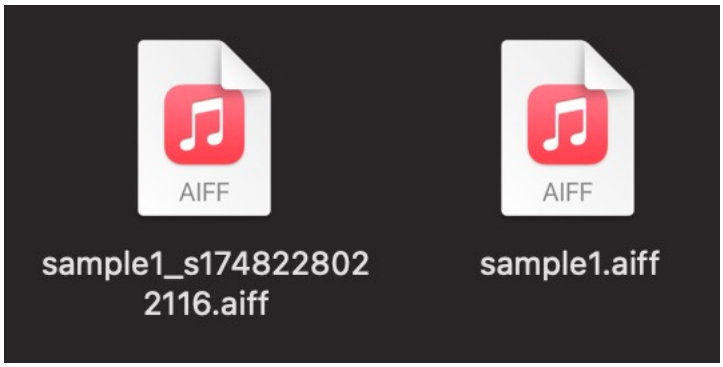


Рисунок 3.3 – Створений новий файл .aiff

Функція декодування передбачає вибір зашифрованого файлу через інтерфейс і натискання кнопки для розшифрування прихованого тексту. Після обробки відображається текстове повідомлення, що було заховано, що дає змогу швидко отримати необхідну інформацію без додаткових технічних дій.

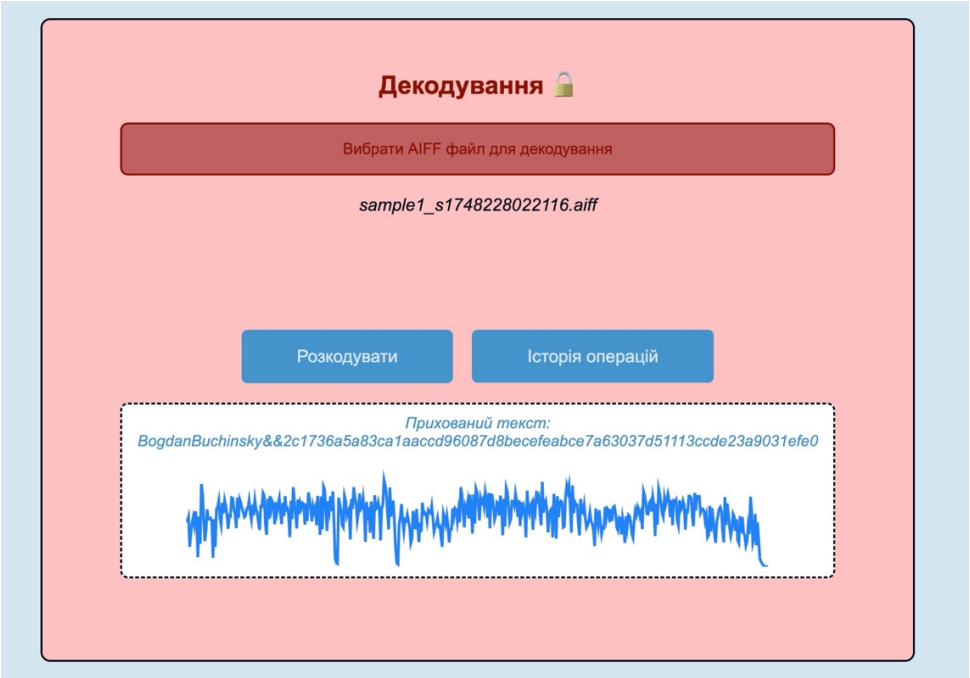


Рисунок 3.4 – Вигляд частини інтерфейсу декодування. Заповнення форми

Важливою частиною програмного забезпечення є модуль збереження історії операцій (рис. 3.5). Кожне кодування чи декодування автоматично записується до бази даних із деталями про файли, обраний алгоритм, тривалість операції та статус виконання. Це дозволяє користувачу відстежувати всі проведені дії, аналізувати ефективність та повторно використовувати результати без ризику втрати інформації.

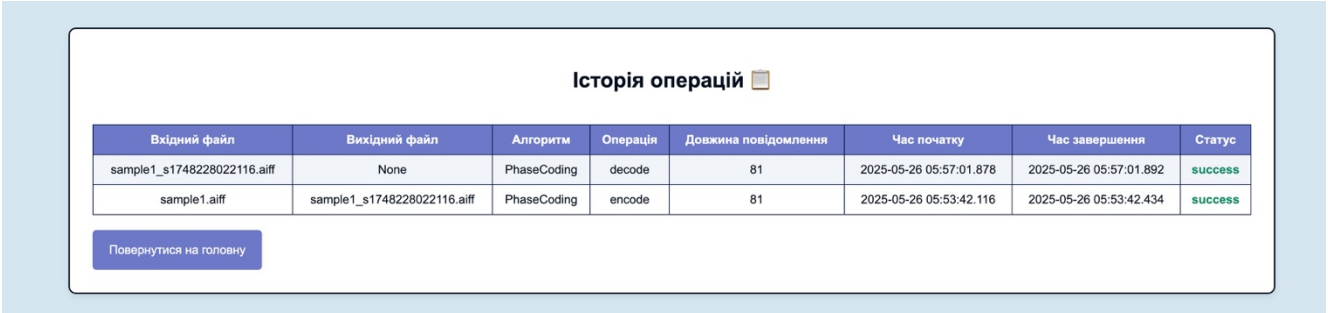


Рисунок 3.5 – Вигляд інтерфейсу таблиці перегляду операцій

Для забезпечення якісної роботи програми рекомендується використовувати аудіофайли з високою роздільною здатністю та стандартними параметрами, такими як 16-бітова глибина та частота дискретизації 44,1 кГц або вище. Перед початком роботи слід перевірити коректність обраних файлів, оскільки пошкоджені чи нестандартні файли можуть спричинити помилки при кодуванні чи декодуванні.

Також важливо враховувати, що вбудоване повідомлення має обмежену довжину через особливості методу фазового кодування. Тому при введенні тексту слід дотримуватися рекомендованих обсягів інформації, які забезпечують надійне приховування без погіршення якості звуку.

Після завершення операцій рекомендується зберігати оригінальні файли для можливості порівняння або повторної обробки. Це підвищує безпеку роботи та дозволяє за необхідності відновити первинний аудіозапис.

Таким чином, розроблене програмне забезпечення має зручний і зрозумілий користувацький інтерфейс, який дозволяє виконувати складні операції фазового кодування аудіо без спеціальних технічних знань. Інструкції та рекомендації, викладені в цьому розділі, допомагають користувачу максимально ефективно застосовувати програму та уникати типових помилок під час роботи.

3.5 Висновки до розділу 3

У розділі було розглянуто комплексний процес розробки програмного забезпечення для захисту авторських прав студійних аудіозаписів формату AIFF на основі методу фазового кодування. Спочатку було обґрунтовано вибір мови програмування Python та супутніх технологій, що забезпечують необхідний функціонал, зручність розробки та підтримку роботи з аудіофайлами високої якості. Обрані інструменти дозволили ефективно реалізувати алгоритми стеганографії та інтегрувати їх у програмний продукт. Далі описано реалізацію ключових модулів системи, зокрема роботи з базою даних для збереження інформації про аудіофайли, операції кодування та декодування, а також алгоритму фазового кодування. Було створено структуру бази даних, що забезпечує цілісність даних і можливість детального журналювання операцій, що підвищує надійність і прозорість роботи системи.

Також було приділено увагу розробці інтуїтивного і функціонального користувацького інтерфейсу, реалізованого за допомогою сучасних веб-технологій. Інтерфейс забезпечує зручність вибору файлів, введення необхідних даних, відображення візуалізації аудіохвиль та отримання результатів кодування і декодування, що робить програму доступною навіть для користувачів без спеціалізованих технічних знань.

Завершальним етапом стала підготовка детальної інструкції користувача та рекомендацій щодо ефективного використання програмного забезпечення. Це сприяє правильному застосуванню системи, зменшує ризики виникнення помилок і забезпечує максимальну користь від реалізованого функціоналу.

Отже, розроблений комплекс програмних компонентів, що включає вибір технологій, реалізацію алгоритмів, створення зручного інтерфейсу та підготовку користувацьких інструкцій, становить цілісне й ефективне рішення для захисту авторських прав аудіозаписів за допомогою фазового кодування.

ВИСНОВОК

У ході виконання дипломної роботи було проведено всебічне дослідження, розробку та реалізацію програмного забезпечення для захисту авторських прав студійних аудіозаписів високої якості у форматі AIFF на основі методу фазового кодування. Робота включала теоретичний аналіз предметної області, розробку алгоритмів, проєктування архітектури та практичну реалізацію системи із зручним інтерфейсом користувача.

На початковому етапі було проведено детальний аналіз предметної області захисту авторських прав у цифрових аудіозаписах, що включало огляд існуючих методів та технологій захисту інформації. Встановлено, що використання стеганографії як методу прихованого вбудовування інформації має значні переваги для забезпечення цілісності та достовірності аудіоматеріалів без помітного впливу на якість звуку. В рамках аналізу було розглянуто основні аудіоформати, їх властивості і обґрунтовано вибір формату AIFF як оптимального варіанту для високоякісних студійних записів через відсутність стиснення та високу точність збереження звукової інформації. Подальша увага була зосереджена на вивченні методу фазового кодування як одного з перспективних стеганографічних підходів. Проведений аналіз показав його ефективність у забезпеченні надійного приховання даних у фазовій складовій аудіосигналу, що мінімізує вплив на акустичні характеристики та підвищує стійкість до різних видів обробки файлів.

Наступний етап роботи полягав у розробці алгоритму інтеграції методу фазового кодування до формату AIFF. Було створено детальний алгоритмічний опис, включно з процедурами кодування та декодування, а також розроблено архітектуру програмного забезпечення, що забезпечує модульність, масштабованість та можливість подальшого розвитку системи. Визначено основні компоненти, такі як модулі роботи з аудіоданими, базою даних для збереження метаданих і операцій, а також інтерфейс користувача.

Практична реалізація програмного продукту здійснювалася з використанням мови програмування Python та супутніх бібліотек, що надали необхідний

функціонал для обробки аудіофайлів AIFF та реалізації стеганографічного алгоритму. Особливу увагу було приділено розробці бази даних для збереження інформації про аудіофайли, операції кодування та декодування, що дозволило забезпечити прозорість та контроль процесів захисту авторських прав. Було створено веб-інтерфейс із зручним UI/UX, який забезпечує інтуїтивно зрозуміле керування програмою, можливість вибору файлів, введення даних автора, відображення візуалізації аудіохвиль та перегляду результатів операцій. Також було розроблено детальну інструкцію користувача, що містить покрокові рекомендації щодо роботи з програмою, що значно полегшує її застосування як для фахівців, так і для користувачів без спеціальних технічних знань. Це сприяє підвищенню ефективності використання розробленого продукту в реальних умовах.

В результаті виконаної роботи створено повноцінну систему захисту авторських прав студійних аудіозаписів, яка поєднує теоретичні напрацювання у сфері стеганографії та практичну реалізацію сучасних методів кодування. Розроблений продукт забезпечує надійне приховання інформації у аудіофайлах без втрати якості, що робить його цінним інструментом для аудіостудій, виконавців та правовласників.

Отже, дипломна робота внесла вагомий внесок у розвиток засобів цифрового захисту аудіоконтенту, підтвердила ефективність застосування методу фазового кодування у форматі AIFF та продемонструвала можливості сучасних програмних технологій для створення зручних і функціональних рішень у цій галузі. Результати роботи можуть бути використані для подальших досліджень, удосконалення алгоритмів та впровадження в комерційні продукти для захисту інтелектуальної власності.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іващенко В. П., Пройдак Ю. С., Фріман І. М., Фріман Є. М. Авторське право: поняття, завдання і принципи: навч. посіб. для студентів. Дніпро: Акцент ПП, 2021.

2. Michaylov K. D., Sarmah D. K. Steganography and steganalysis for digital image enhanced forensic analysis and recommendations. Journal of Cyber Security Technology. 2024.

3. Murthy S. V. N., Reddy G. J. P., Sharanya R., Shirisha S., Yashawini S. Image, audio and video steganography. International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal. 2023.

4. Rahman S., Uddin J., Zakarya M. A comprehensive study of digital image steganographic techniques. IEEE. 2023.

5. Alher Z. A., Al Imran B. M., Al Ali I. LSB as a steganography tool in information security. University of Kerbala. 2024.

6. Хорошко В. О., Яремчук Ю. Є., Карпінець В. В. Комп'ютерна стеганографія: навч. посіб. Вінниця: ВНТУ, 2017. 156 с.

7. Чепок О. Л. Механічні коливання і хвилі. Основи акустики. Одеса, 2021.

8. Кононов С. П., Макогон В. І. Основи радіомовлення та звукотехніки: навч. посіб. Вінниця: ВНТУ, 2018.

9. Шевчук Р. П. Програмне забезпечення мультимедіа: навч. посіб. для студентів напряму «Програмна інженерія». Тернопіль, 2011. 174 с.

10. Найда С. А., Іванько К. О. Акустика слуху: навч. посіб. Київ, 2020.

11. Юдін О. К., Конахович Г. Ф., Корченко О. Г. Захист інформації в мережах передачі даних: підручник. Київ: Інтерсервіс, 2009. 714 с.

12. Ковальчук С. О., Кухарська Н. П. Використання методу фазового кодування для приховування конфіденційної інформації в аудіофайлах. Львів, 2015.

13. Романюк М. І., Власюк Г. Г. Основи теорії інформації та кодування: навч. посібник. Київ: КПІ ім. Ігоря Сікорського, 2018. 81 с.

14. Денбновецький С. В., Мельник І. В., Писаренко Л. Д. Кодування сигналів в електронних системах. Способи кодування сигналів: Натуральні, ефективні та лінійні коди: навч. посібник. Київ: КПІ ім. Ігоря Сікорського, 2021.

15. Рудий Є. М. Технології передачі дискретних повідомлень: Стиснення телефонних сигналів: навч. посіб. Одеса: ОНАЗ ім. О. С. Попова, 2013. 102 с.

16. Ужинський М. Ю. Основи цифрового звукозапису. Рівне: РДГУ, 2022. 37 с.

17. Ушенко Ю. О., Олар О. В., Галочкін О. В., Д'яченко Л. І. Сучасні технології розробки web-додатків: Фронтенд розробка: навч. посіб. Чернівці: Чернівецький нац. ун-т, 2022. 222 с.

18. Коновалов В. С., Радоуцький К. Є. Сучасні принципи і методи проектування програмного забезпечення: конспект лекцій. Харків: УкрДАЗТ, 2015. Ч. 2. 109 с.

19. Ахромов М. О. Організація баз даних та знань. Краматорськ: МК ДГМА, 2017. 148 с.

20. Згуровська Л. П., Киричук Ю. В., Назаренко Н. М. Бази даних. Комп'ютерний практикум: навч. посіб. Київ: КПІ ім. Ігоря Сікорського, 2021. 241 с.

21. Бєлов Ю. А., Карнаух Т. О., Коваль Ю. В., Ставровський А. Б. Вступ до програмування мовою С++. Організація обчислень : навч. посіб. Київ : Видавничо-поліграфічний центр "Київський університет", 2012. 175 с.

22. Java-програмування: комп'ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спец. 122 «Комп'ютерні науки» / уклад. Ю. А. Тарнавський ; КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2021. 95 с. Електронні текстові дані (1 файл: 686 Кбайт).

23. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек-енд) : навч. посіб. Львів : Видавництво Львівської політехніки, 2018. 248 с. (протокол № 34 від 15.03.2018 р.)

24. Яковенко А. В. Основи програмування. Python [Електронний ресурс] : підруч. для студ. спец. 122 «Комп'ютерні науки» / КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2018. 195 с. Електронні текстові дані (1 файл: 1,59 Мбайт).

25. Анісімов А. В., Дорошенко А. Ю., Погорілий С. Д., Дорогий Я. Ю. Програмування числових методів мовою Python : підруч. / за ред. А. В. Анісімова. Київ : Видавничо-поліграфічний центр “Київський університет”, 2014. 640 с.

26. Duckett J. HTML and CSS: Design and Build Websites : навч. посіб. Hoboken, NJ : John Wiley & Sons, 2014. 89 с.

27. Жарікова М. В. Розробка ігрових web додатків : електрон. навч. посібник. Херсон : ФОП Вишемирський В. С., 2018. 218 с.

28. Балик Н. Р., Мандзюк В. І. Бази даних MySQL : навч. посіб. Тернопіль, 2010. 160 с.

29. Павловський В. І., Петрашенко А. В., Побєда Д. В. Бази даних та засоби управління. Практикум [Електронний ресурс] : навч. посіб. для студ. спец. 123 – Комп’ютерна інженерія / КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2021. 112 с. Електронні текстові дані (1 файл: 7,7 Мбайт).

30. Крилов Є. В., Анікін В. К. Технології проєктування NoSQL баз даних [Електронний ресурс] : навч. посіб. для здобувачів ступеня магістра спец. 126 «Інформаційні системи та технології» / КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2023. 117 с. Електронні текстові дані (1 файл: 7,5 Мбайт).

31. Лосєв М. Ю., Федько В. В. Бази даних : навч.-практ. посібник [Електронний ресурс]. Харків : ХНЕУ ім. С. Кузнеця, 2018. 233 с.

32. Чемерис Г. Ю. UX/UI дизайн : навч. посіб. для здобувачів ступеня бакалавра спец. «Дизайн», освіт.-проф. програми «Графічний дизайн». Запоріжжя : ЗНУ, 2021. 290 с.

33. Гаврил М. UX/UI дизайн: основні принципи розробки ефективних інтерфейсів. Дніпро : ФСЗМК, 2023.

ДОДАТКИ

Вінницький національний технічний університет

Факультет менеджменту та інформаційної безпеки

Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ

Голова секції "Управління інформаційною безпекою" кафедри МБІС

д.т.н., професор

Юрій ЯРЕМЧУК

" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

Розробка програми захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування

08-72.БКР.003.00.109 ТЗ

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доц., зав. каф. МБІС

Карпінець В.В.

" " _____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб для захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування.

Область застосування: захист авторських прав на студійні аудіозаписи шляхом прихованого вбудовування ідентифікаційної інформації у звукові дані з метою протидії несанкціонованому доступу та незаконному розповсюдженню в системах інформаційної безпеки.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета: розробка програмного засобу для захисту авторських прав на студійні аудіозаписи високої якості шляхом застосування методу фазового кодування.

3.2 Призначення: розроблений програмний засіб забезпечує приховане вбудовування ідентифікаційної інформації в аудіофайли формату AIFF з метою підтвердження авторства та унеможливлення несанкціонованого використання.

4. Джерела розробки

4.1 Іващенко В. П., Пройдак Ю. С., Фріман І. М., Фріман Є. М. Авторське право: поняття, завдання і принципи: навч. посіб. для студентів. Дніпро: Акцент ПП, 2021.

4.2 Michaylov K. D., Sarmah D. K. Steganography and steganalysis for digital image enhanced forensic analysis and recommendations. Journal of Cyber Security Technology. 2024.

4.3 Murthy S. V. N., Reddy G. J. P., Sharanya R., Shirisha S., Yashawini S. Image, audio and video steganography. International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal. 2023.

4.4 Rahman S., Uddin J., Zakarya M. A comprehensive study of digital image steganographic techniques. IEEE. 2023.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Інтерфейс програми має бути інтуїтивно зрозумілим та адаптованим для взаємодії;

5.1.2 Реалізація функціоналу не повинна вимагати використання комерційного або ліцензійного програмного забезпечення;

5.1.3 Програмний засіб повинен забезпечувати можливість вбудовування і вилучення прихованих повідомлень в аудіофайли формату AIFF із застосуванням методу фазового кодування.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен стабільно працювати під час обробки аудіофайлів, а у випадку виникнення помилок – інформувати користувача відповідними повідомленнями;

5.2.2 База даних, у якій зберігається інформація про кодування, повинна підтримувати резервне копіювання;

5.2.3 Програма повинна коректно виконувати всі визначені функції: вбудовування, зчитування повідомлень і збереження інформації в базу даних.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – не нижче Intel Pentium 1500 МГц або еквівалентний;

– оперативна пам’ять – не менше 512 МБ;

– середовище функціонування – будь-яка сучасна операційна система з підтримкою Python та браузера (наприклад, Windows, macOS, Linux);

– вимоги до техніки безпеки при роботі з програмним засобом повинні відповідати чинним стандартам безпеки користування комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4.

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист програмного засобу від несанкціонованого копіювання та модифікації шляхом вбудовування авторських даних у аудіофайл формату AIFF;

7.2 Необхідно унеможливити несанкціоноване вилучення або підміну
хованої інформації в аудіофайлі. 81

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати
трати на його реалізацію;

8.2 Програмний засіб має бути реалізований таким чином, щоб його можна було
користовувати широким спектром користувачів без потреби у дорогому
бладнанні чи спеціалізованому програмному забезпеченні.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025
2.	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3.	Розробка алгоритму роботи	14.04.2025	27.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	08.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

- ПЗ до бакалаврської кваліфікаційної роботи;
- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____ Бучинський Б.В.

Додаток Б. Лістинг коду основної логіки програмного забезпечення

```
from phase_stegnography.AudioStegnographyAlgorithm import AudioStego


class PhaseEncodingAudioStego(AudioStego):

    def encodeAudio(self, audioLocation, stringToEncode, outputPath=None) -> str:
        self.convertToByteArray(audioLocation)

        stringToEncode = stringToEncode.ljust(100, '~')
        textLength = 8 * len(stringToEncode)

        blockLength = int(2 * 2 ** np.ceil(np.log2(2 * textLength)))
        blockNumber = int(np.ceil(self.audioData.shape[0] / blockLength))

        if len(self.audioData.shape) == 1:
            self.audioData.resize(blockNumber * blockLength, refcheck=False)
            self.audioData = self.audioData[np.newaxis]
        else:
            self.audioData.resize((blockNumber * blockLength, self.audioData.shape[1]),
refcheck=False)
            self.audioData = self.audioData.T

        blocks = self.audioData[0].reshape((blockNumber, blockLength))

        blocks = np.fft.fft(blocks)
        magnitudes = np.abs(blocks)
        phases = np.angle(blocks)
        phaseDiffs = np.diff(phases, axis=0)

        textInBinary = np.ravel([[int(y) for y in format(ord(x), "08b")] for x in stringToEncode])
        textInPi = textInBinary.copy()
        textInPi[textInPi == 0] = -1
        textInPi = textInPi * -np.pi / 2

        blockMid = blockLength // 2
        phases[0, blockMid - textLength: blockMid] = textInPi
        phases[0, blockMid + 1: blockMid + 1 + textLength] = -textInPi[::-1]

        for i in range(1, len(phases)):
            phases[i] = phases[i - 1] + phaseDiffs[i - 1]

        blocks = (magnitudes * np.exp(1j * phases))
        blocks = np.fft.ifft(blocks).real

        self.audioData[0] = blocks.ravel().astype(np.int16)

        return self.saveToLocation(self.audioData.T, audioLocation, outputPath)
```

```
    def decodeAudio(self, audioLocation) -> str:
        self.convertToByteArray(audioLocation)
        textLength = 800
        blockLength = 2 * int(2 ** np.ceil(np.log2(2 * textLength)))
        blockMid = blockLength // 2

        if len(self.audioData.shape) == 1:
            secret = self.audioData[:blockLength]
        else:
            secret = self.audioData[:blockLength, 0]

        secretPhases = np.angle(np.fft.fft(secret))[blockMid - textLength:blockMid]
        secretInBinary = (secretPhases < 0).astype(np.int8)
        secretInIntCode = secretInBinary.reshape((-1, 8)).dot(1 << np.arange(8 - 1, -1, -1))

        return "".join(np.char.mod("%c", secretInIntCode)).replace("~", "")

    def convertToByteArray(self, audio):
        try:
            self.audioData, self.rate = sf.read(audio, dtype='int16')
        except Exception as e:
            print("Error reading AIFF:", e)

        self.audioData = self.audioData.copy()

    def saveToLocation(self, audioArray, location, outputPath=None) -> str:
        if outputPath is not None:
            output_path = outputPath
        else:
            dir = os.path.dirname(location)
            output_path = os.path.join(dir, "output-pc.aiff")

        if len(audioArray.shape) > 1:
            audioArray = audioArray[:, 0]

        sf.write(output_path, audioArray, self.rate, format='AIFF', subtype='PCM_16')
        return output_path

from abc import ABC, abstractmethod
class AudioStego(ABC):
    def __init__(self):
        pass

    @abstractmethod
    def encodeAudio(self, audioLocation, stringToEncode) -> str:
        pass

    @abstractmethod
```

```
    def decodeAudio(self, audioLocation) -> str:
        pass

    @abstractmethod
    def convertToByteArray(self, audio):
        pass

    @abstractmethod
    def saveToLocation(self, audioArray, location) -> str:
        pass

// database.py
import sqlite3
from datetime import datetime

DB_NAME = 'app_data.db'

def init_db():
    conn = sqlite3.connect(DB_NAME)
    c = conn.cursor()

    c.execute('''
        CREATE TABLE IF NOT EXISTS audio_files (
            audio_file_id INTEGER PRIMARY KEY AUTOINCREMENT,
            file_name TEXT NOT NULL,
            file_path TEXT NOT NULL,
            file_format TEXT,
            sample_rate INTEGER,
            bit_depth INTEGER,
            channels INTEGER,
            duration_seconds REAL,
            file_size_bytes INTEGER,
            created_at TEXT NOT NULL
        )
    ''')

    c.execute('''
        CREATE TABLE IF NOT EXISTS stego_algorithms (
            algorithm_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL UNIQUE,
            version TEXT,
            description TEXT
        )
    ''')

    c.execute('''
        CREATE TABLE IF NOT EXISTS stego_operations (
            operation_id INTEGER PRIMARY KEY AUTOINCREMENT,
            input_file_id INTEGER NOT NULL,
```

```
                output_file_id INTEGER,
                algorithm_id INTEGER NOT NULL,
                operation_type TEXT NOT NULL,
                message_length INTEGER,
                start_time TEXT NOT NULL,
                end_time TEXT,
                status TEXT,
                FOREIGN KEY(input_file_id) REFERENCES audio_files(audio_file_id),
                FOREIGN KEY(output_file_id) REFERENCES audio_files(audio_file_id),
                FOREIGN KEY(algorithm_id) REFERENCES stego_algorithms(algorithm_id)
            )
        ''')

        c.execute("SELECT algorithm_id FROM stego_algorithms WHERE name = ?",
('PhaseCoding',))
        if c.fetchone() is None:
            c.execute('''
                INSERT INTO stego_algorithms (name, version, description)
                VALUES (?, ?, ?)
            ''', ('PhaseCoding', '1.0', 'Phase coding steganography method'))

        conn.commit()
        conn.close()

    def add_audio_file(file_name, file_path, file_format=None, sample_rate=None,
bit_depth=None, channels=None, duration_seconds=None, file_size_bytes=None):
        conn = sqlite3.connect(DB_NAME)
        c = conn.cursor()

        created_at = datetime.now().isoformat(sep=' ', timespec='seconds')

        c.execute('''
            INSERT INTO audio_files (file_name, file_path, file_format, sample_rate, bit_depth,
channels, duration_seconds, file_size_bytes, created_at)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (file_name, file_path, file_format, sample_rate, bit_depth, channels, duration_seconds,
file_size_bytes, created_at))

        audio_file_id = c.lastrowid
        conn.commit()
        conn.close()
        return audio_file_id

    def get_algorithm_id_by_name(name):
        conn = sqlite3.connect(DB_NAME)
        c = conn.cursor()
        c.execute("SELECT algorithm_id FROM stego_algorithms WHERE name = ?", (name,))
        row = c.fetchone()
        conn.close()
```

```
        if row:
            return row[0]
        return None

    def log_operation(input_file_id, output_file_id, algorithm_id, operation_type,
message_length, start_time=None, end_time=None, status='success'):
        conn = sqlite3.connect(DB_NAME)
        c = conn.cursor()

        if start_time is None:
            start_time = datetime.now().isoformat(sep=' ', timespec='seconds')

        c.execute('''
            INSERT INTO stego_operations (input_file_id, output_file_id, algorithm_id,
operation_type, message_length, start_time, end_time, status)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        ''', (input_file_id, output_file_id, algorithm_id, operation_type, message_length, start_time,
end_time, status))

        conn.commit()
        conn.close()

    def get_all_operations():
        conn = sqlite3.connect(DB_NAME)
        c = conn.cursor()
        c.execute('''
            SELECT
                so.operation_id,
                af_in.file_name AS input_file,
                af_out.file_name AS output_file,
                sa.name AS algorithm,
                so.operation_type,
                so.message_length,
                so.start_time,
                so.end_time,
                so.status
            FROM stego_operations so
            JOIN audio_files af_in ON so.input_file_id = af_in.audio_file_id
            LEFT JOIN audio_files af_out ON so.output_file_id = af_out.audio_file_id
            JOIN stego_algorithms sa ON so.algorithm_id = sa.algorithm_id
            ORDER BY so.start_time DESC
        ''')
        result = c.fetchall()
        conn.close()
        return result
```

Додаток В. Лістинг коду графічного інтерфейсу і компонентів взаємодії з користувачем

```
//app.py
from flask import Flask, jsonify, render_template, request, redirect, url_for
import os
import soundfile as sf
import hashlib
from datetime import datetime
import time

from phase_stegnography.PhaseEncodingAudioStego import PhaseEncodingAudioStego
from database_stegnography.database import init_db, add_audio_file,
get_algorithm_id_by_name, log_operation, get_all_operations

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
ALLOWED_EXTENSIONS = {'aiff'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

@app.route('/aiffForm-data/<filename>')
def aiffForm_data(filename):
    filepath = os.path.join(UPLOAD_FOLDER, filename)
    if not os.path.exists(filepath):
        return jsonify({'error': 'Файл не знайдено'}), 404

    data, samplerate = sf.read(filepath)
    if len(data.shape) > 1:
        data = data[:, 0]

    samples_per_point = max(len(data) // 500, 1)
    aiffForm = []
    for i in range(0, len(data), samples_per_point):
        chunk = data[i:i+samples_per_point]
        avg = float(np.abs(chunk).mean())
        aiffForm.append(avg)

    return jsonify({'aiffForm': aiffForm})

@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/encode', methods=['POST'])
def encode():
    if 'audio_file' not in request.files or 'author_name' not in request.form or 'author_surname' not in request.form:
        return redirect(url_for('index'))

    audio_file = request.files['audio_file']
    author_name = request.form['author_name'].strip()
    author_surname = request.form['author_surname'].strip()

    if audio_file.filename == '' or not allowed_file(audio_file.filename):
        return 'Invalid file format. Only AIFF files are allowed.'

    filename = secure_filename(audio_file.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    audio_file.save(filepath)

    data, rate = sf.read(filepath)
    duration = len(data) / rate
    file_size = os.path.getsize(filepath)

    input_file_id = add_audio_file(
        file_name=filename,
        file_path=filepath,
        file_format='AIFF',
        sample_rate=rate,
        bit_depth=None,
        channels=data.shape[1] if len(data.shape) > 1 else 1,
        duration_seconds=duration,
        file_size_bytes=file_size
    )

    with open(filepath, 'rb') as f:
        file_hash = hashlib.sha256(f.read()).hexdigest()

    full_message = f"{author_name}{author_surname}&&{file_hash}"

    name, ext = os.path.splitext(filename)
    unique_suffix = str(int(time.time() * 1000))  # мілісекунди
    encoded_filename = f"{name}_s{unique_suffix}{ext}"
    encoded_path = os.path.join(app.config['UPLOAD_FOLDER'], encoded_filename)

    stego = PhaseEncodingAudioStego()
    start_time = datetime.now()
    stego.encodeAudio(filepath, full_message, outputPath=encoded_path)
    end_time = datetime.now()

    data_out, rate_out = sf.read(encoded_path)
```

```
    duration_out = len(data_out) / rate_out
    file_size_out = os.path.getsize(encoded_path)

    output_file_id = add_audio_file(
        file_name=encoded_filename,
        file_path=encoded_path,
        file_format='AIFF',
        sample_rate=rate_out,
        bit_depth=None,
        channels=data_out.shape[1] if len(data_out.shape) > 1 else 1,
        duration_seconds=duration_out,
        file_size_bytes=file_size_out
    )

    algorithm_id = get_algorithm_id_by_name('PhaseCoding')

    log_operation(
        input_file_id=input_file_id,
        output_file_id=output_file_id,
        algorithm_id=algorithm_id,
        operation_type='encode',
        message_length=len(full_message),
        start_time=start_time.isoformat(sep=' ', timespec='milliseconds'),
        end_time=end_time.isoformat(sep=' ', timespec='milliseconds'),
        status='success'
    )

    return render_template('index.html', encoded_file=encoded_filename)

@app.route('/decode', methods=['POST'])
def decode():
    if 'audio_file' not in request.files:
        return redirect(url_for('index'))

    audio_file = request.files['audio_file']

    if audio_file.filename == '' or not allowed_file(audio_file.filename):
        return 'Invalid file format. Only AIFF files are allowed.'

    filename = secure_filename(audio_file.filename)
    filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    audio_file.save(filepath)

    stego = PhaseEncodingAudioStego()
    start_time = datetime.now()
    decoded_message = stego.decodeAudio(filepath)
    end_time = datetime.now()

    data, rate = sf.read(filepath)
```

```
            duration = len(data) / rate
            file_size = os.path.getsize(filepath)

            input_file_id = add_audio_file(
                file_name=filename,
                file_path=filepath,
                file_format='AIFF',
                sample_rate=rate,
                bit_depth=None,
                channels=data.shape[1] if len(data.shape) > 1 else 1,
                duration_seconds=duration,
                file_size_bytes=file_size
            )

            algorithm_id = get_algorithm_id_by_name('PhaseCoding')

            log_operation(
                input_file_id=input_file_id,
                output_file_id=None,
                algorithm_id=algorithm_id,
                operation_type='decode',
                message_length=len(decoded_message),
                start_time=start_time.isoformat(sep=' ', timespec='milliseconds'),
                end_time=end_time.isoformat(sep=' ', timespec='milliseconds'),
                status='success'
            )

            return render_template('index.html', decoded_message=decoded_message,
decoded_file=filename)

        @app.route('/history')
        def history():
            encodings = get_all_operations()
            return render_template('history.html', encodings=encodings)

        if __name__ == '__main__':
            init_db()
            app.run(debug=True)

        //index.html
        <!DOCTYPE html>
        <html lang="ua">

        <head>
                <meta charset="UTF-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0">
                <meta name="author" content="Bogdan Buchinskiy">
                <title>Stego-AIFF Web</title>
                <link rel="stylesheet" href="/static/style.css">
```

```
    </head>

    <body>
        <h1 class="header">Фазове кодування студійних аудіозаписів AIFF 🎧</h1>
        <div class="container">
    <div class="box-encrypt">
        <h2 class="subheader-encrypt">Кодування</h2>
        <form method="post" action="/encode" enctype="multipart/form-data">
            <div class="custom-file-upload-encrypt">
                <label for="encodeFile">Вибрати AIFF файл для кодування</label>
                <input type="file" name="audio_file" id="encodeFile" required
onchange="handleFileUpload(this)"
                    accept=".aiff">
                <div class="output">
                    <span class="file-name" id="encodeFileName">Файл не
вибрано</span>
                    <div id="fileHash" class="file-hash"></div>
                    <canvas id="aiffFormCanvas" width="600"
height="120"></canvas>
                </div>
            </div>

            <input type="text" name="author_name" placeholder="Ім’я автора" required>
            <input type="text" name="author_surname" placeholder="Прізвище автора"
required>

            <button class="submit-button" type="submit">Закодувати</button>
        </form>

        {% if encoded_file %}
        <div class="output-result">
            Файл збережено: {{ encoded_file }}
        </div>
        {% endif %}
    </div>

    <div class="box-decrypt">
        <h2 class="subheader-decrypt">Декодування</h2>
        <form method="post" action="/decode" enctype="multipart/form-data">
            <div class="custom-file-upload-decrypt">
                <label for="decodeFile">Вибрати AIFF файл для декодування</label>
                <input type="file" name="audio_file" id="decodeFile" required
onchange="updateFileName(this)"
                    accept=".aiff">
                <span class="file-name" id="decodeFileName">Файл не
вибрано</span>
            </div>

            <button class="submit-button" type="submit">Розкодувати</button>
```

```
                <a href="/history" class="submit-button submit-button-history">Історія операцій</a>
            </form>

            {% if decoded_message %}
            <div class="output">
                Прихований текст: <br> {{ decoded_message }}
                <canvas id="decodeaiffFormCanvas" width="600" height="120"></canvas>
            </div>
            {% endif %}
        </div>
        </div>

            <script>
                async function drawaiffForm(filename, canvasId = 'aiffFormCanvas') {
                    const response = await fetch(`/aiffForm-data/${filename}`)
                    if (!response.ok) {
                        console.error('Не вдалося завантажити дані хвильової форми')
                        return
                    }
                    const data = await response.json()
                    if (data.error) {
                        console.error(data.error)
                        return
                    }
                    const aiffForm = data.aiffForm

                    const canvas = document.getElementById(canvasId)
                    if (!canvas) {
                        console.error(`Canvas з ID ${canvasId} не знайдено`)
                        return
                    }

                    const ctx = canvas.getContext('2d')

                    const dpr = window.devicePixelRatio || 1
                    const cssWidth = canvas.width
                    const cssHeight = canvas.height
                    canvas.width = cssWidth * dpr
                    canvas.height = cssHeight * dpr
                    canvas.style.width = cssWidth + 'px'
                    canvas.style.height = cssHeight + 'px'
                    ctx.scale(dpr, dpr)

                    const width = cssWidth
                    const height = cssHeight
                    const middle = height

                    const step = width / aiffForm.length
```

```
                const amplitudeScale = 2.5
                const animationIntensity = 0.0015

                const color = 'rgba(0, 123, 255, 0.9)'

                let frame = 0

                function drawFrame() {
                        ctx.clearRect(0, 0, width, height)
                        ctx.beginPath()

                        for (let i = 0; i < aiffForm.length; i++) {
                                const x = i * step
                                const waveShift = Math.sin((frame * 0.025) + i * 0.05) *
animationIntensity
                                const sample = aiffForm[i] + waveShift
                                const y = middle - (sample * middle * amplitudeScale)
                                if (i === 0) {
                                        ctx.moveTo(x, y)
                                } else {
                                        ctx.lineTo(x, y)
                                }
                        }

                        ctx.strokeStyle = color
                        ctx.lineWidth = 2.8
                        ctx.shadowColor = 'rgba(0, 0, 0, 0.25)'
                        ctx.shadowBlur = 1.5
                        ctx.stroke()

                        frame++
                        requestAnimationFrame(drawFrame)
                }

                drawFrame()
        }
    </script>

{% if encoded_file %}
<script>
    drawaiffForm("{{ encoded_file }}")
</script>
{% endif %}

{% if decoded_file %}
<script>
    drawaiffForm("{{ decoded_file }}", "decodeaiffFormCanvas")
</script>
```

```
{% endif %}

        <script>
function updateFileName(input) {
        const fileName = input.files.length > 0 ? input.files[0].name : "Файл не вибрано"
        const labelId = input.id + "Name"
        document.getElementById(labelId).textContent = fileName
}

function handleFileUpload(input) {
        updateFileName(input)
        const file = input.files[0]
        if (!file) return

        const reader = new FileReader()
        reader.onload = async function (event) {
                const arrayBuffer = event.target.result
                const hashBuffer = await crypto.subtle.digest('SHA-256', arrayBuffer)
                const hashArray = Array.from(new Uint8Array(hashBuffer))
                const hashHex = hashArray.map(b => b.toString(16).padStart(2, '0')).join('')
                document.getElementById('fileHash').textContent = "SHA-256: " + hashHex
                setTimeout(() => drawaiffForm(file.name), 500)
        }
        reader.readAsArrayBuffer(file)
}
        </script>
</body>
</html>

//history.html
<!DOCTYPE html>
<html lang="ua">

<head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <meta name="author" content="Bogdan Buchinskiy">
        <title>Stego-AIFF History</title>
        <link rel="stylesheet" href="/static/style.css" />
</head>

<body>
        <div class="container-history">
                <h1 class="header">Історія операцій</h1>

                {% if encodings %}
                <div class="table-container">
                        <table class="history-table">
                                <thead>
```

```
                                        <tr>
                                                <th>Вхідний файл</th>
                                                <th>Вихідний файл</th>
                                                <th>Алгоритм</th>
                                                <th>Операція</th>
                                                <th>Довжина повідомлення</th>
                                                <th>Час початку</th>
                                                <th>Час завершення</th>
                                                <th>Статус</th>
                                        </tr>
                                </thead>
                                <tbody>
                                        {% for op in encodings %}
                                        <tr>
                                                <td>{{ op[1] }}</td>
                                                <td>{{ op[2] }}</td>
                                                <td>{{ op[3] }}</td>
                                                <td>{{ op[4] }}</td>
                                                <td>{{ op[5] }}</td>
                                                <td>{{ op[6] }}</td>
                                                <td>{{ op[7] if op[7] else '–' }}</td>
                                                <td class="status-{{ 'success' if op[8] == 'success'
else 'failed' }}">{{ op[8] }}</td>
                                        </tr>
                                        {% endfor %}
                                </tbody>
                        </table>
                </div>
                {% else %}
                <p class="no-data">Немає записів для відображення.</p>
                {% endif %}
                <a href="/" class="back-button">Повернутися на головну</a>
        </div>
</body>

</html>

//style.css
:root {
        --basic-text-color: #011020;
        --encrypt-text-color: #008d66;
        --decrypt-text-color: #8d0000;
        --history-text-color: #000c39;

        --basic-input-color: #4394cd;
        --encrypt-input-color: #7cc061;
        --decrypt-input-color: #c06161;
        --history-input-color: #6c79c8;
```

```
        --basic-background-color: #d4e7f0;
        --encrypt-background-color: #c8ffb2;
        --decrypt-background-color: #ffc1c1;
        --history-background-color: #ffffff;
}

* {
        box-sizing: border-box;
}

body {
        font-family: Arial, sans-serif;
        padding: 30px;
        color: var(--basic-text-color);
        background-color: var(--basic-background-color);
}

a {
        text-decoration: none;
}

input[type='text'] {
        width: 100%;
        border-radius: 8px;
        font-size: 16px;
        border: 2px solid var(--encrypt-text-color);
        font-size: 16px;
        padding: 15px 10px;
        text-align: center;
        &::placeholder {
                font-size: 16px;
                transition: 0.2s all;
        }
        &:active {
                border: 2px solid var(--encrypt-text-color);
        }
        cursor: text;
        transition: 0.5s all;
        &:hover {
                &::placeholder {
                        color: var(--encrypt-text-color);
                        transform: translateY(-2px);
                }
        }
}

canvas {
        display: block;
        margin: 0 auto;
```

```
}

.submit-button {
	border: 2px solid var(--basic-input-color);
	padding: 15px 55px;
	background-color: var(--basic-input-color);
	color: var(--basic-background-color);
	border-radius: 6px;
	cursor: pointer;
	font-size: 18px;
	transition: 0.5s all;
	&:hover {
		background-color: var(--basic-background-color);
		color: var(--basic-input-color);
		border: 2px solid var(--basic-input-color);
		transform: translateY(-3px);
	}
}

.submit-button-history {
	margin-left: 15px;
}

.container {
	display: flex;
	flex-direction: row;
	justify-content: center;
	gap: 45px;
	margin-top: 50px;
}

.header {
	text-align: center;
	font-size: 32px;
}

.box-encrypt {
	width: 45%;
	background-color: var(--encrypt-background-color);
	border: 2px solid;
	border-radius: 10px;
	padding: 50px 80px;
	text-align: center;
	button,
	input[type='text'] {
		margin-top: 28px;
	}
}
```

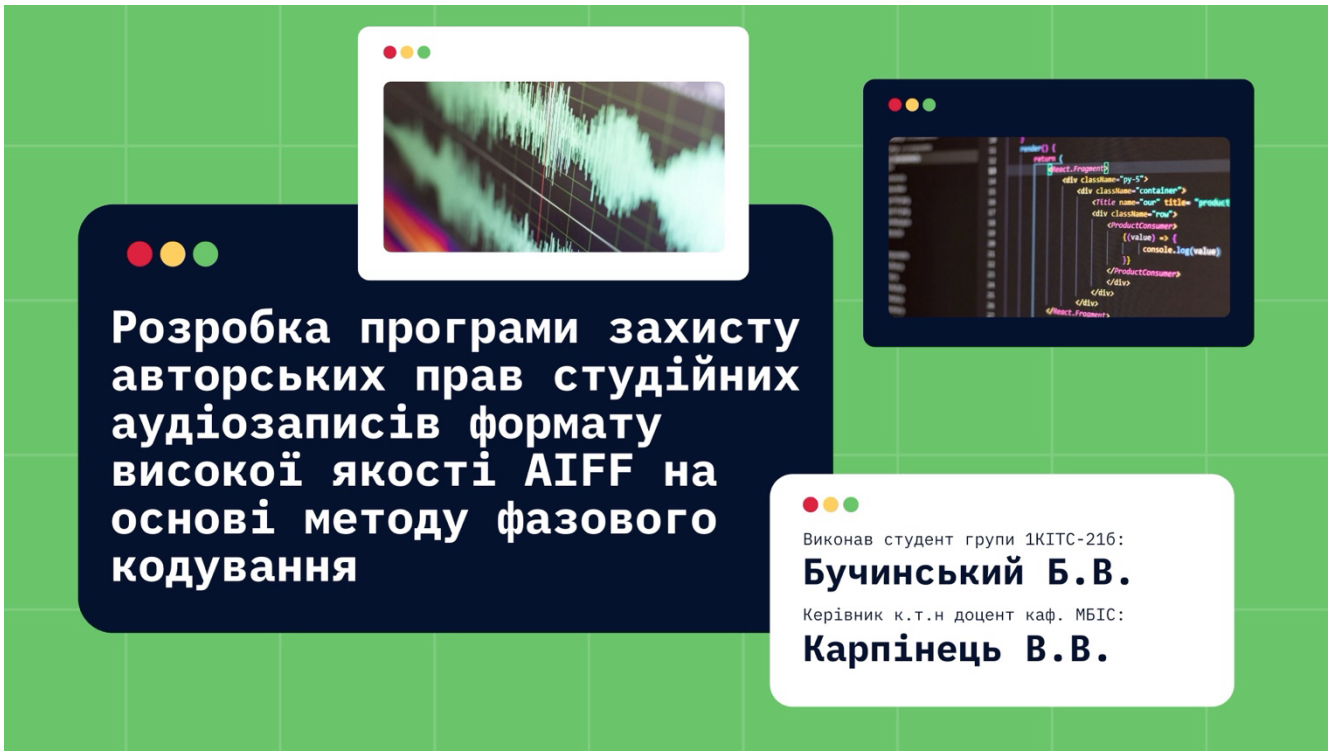
```
.subheader-encrypt {
	text-align: center;
	font-size: 26px;
	margin-top: 0;
	color: var(--encrypt-text-color);
}

.custom-file-upload-encrypt label {
	display: flex;
	align-items: center;
	justify-content: center;
	gap: 10px;
	margin-bottom: 20px;
	background-color: var(--encrypt-input-color);
	color: var(--encrypt-text-color);
	border: 2px solid var(--encrypt-text-color);
	padding: 16px 20px;
	border-radius: 8px;
	cursor: pointer;
	font-size: 16px;
	transition: 0.5s all;
	&:hover {
		background-color: var(--encrypt-text-color);
		color: var(--encrypt-input-color);
		transform: translateY(-2px);
	}
}

/* Decrypt styles */

.box-decrypt {
	width: 45%;
	background-color: var(--decrypt-background-color);
	border: 2px solid;
	border-radius: 10px;
	padding: 50px 80px;
	text-align: center;
	button {
		margin-top: 108px;
	}
}

.subheader-decrypt {
	text-align: center;
	font-size: 26px;
	margin-top: 0;
	color: var(--decrypt-text-color);
}
```

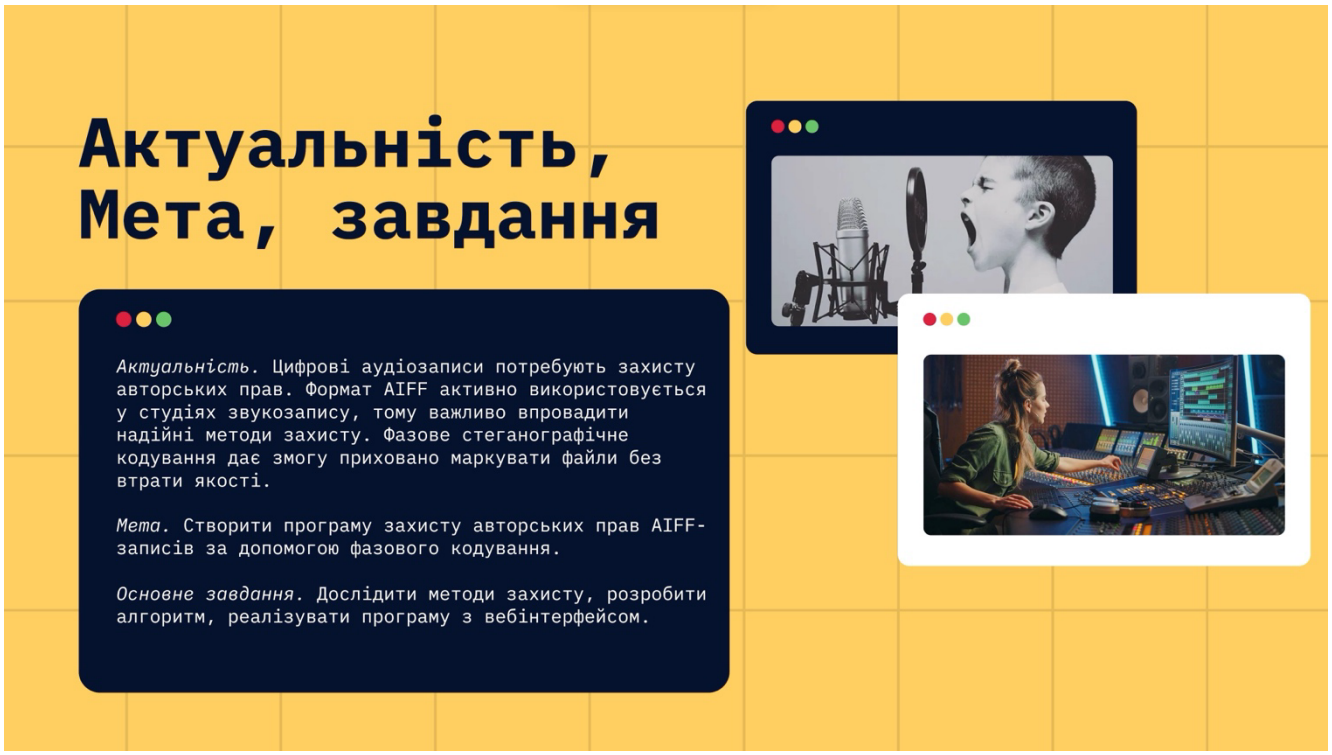
```
.custom-file-upload-decrypt label {
        display: flex;
        align-items: center;
        justify-content: center;
        gap: 10px;
        margin-bottom: 20px;
        background-color: var(--decrypt-input-color);
        color: var(--decrypt-text-color);
        border: 2px solid var(--decrypt-text-color);
        padding: 16px 20px;
        border-radius: 8px;
        cursor: pointer;
        font-size: 16px;
        transition: 0.5s all;
        &:hover {
                background-color: var(--decrypt-text-color);
                color: var(--decrypt-input-color);
                transform: translateY(-2px);
        }
}

.output {
        margin-top: 20px;
        padding: 10px;
        background-color: var(--history-background-color);
        border-radius: 8px;
        font-size: 16px;
        font-style: italic;
        color: var(--basic-input-color);
        border: 2px dashed var(--basic-text-color);
}

.output-result {
        margin-top: 20px;
        padding: 10px;
        background-color: var(--basic-background-color);
        border-radius: 8px;
        font-size: 16px;
        font-style: italic;
        color: var(--basic-input-color);
        border: 2px dashed var(--basic-input-color);
}

.file-name {
        font-size: 18px;
        font-style: italic;
        color: var(--basic-text-color);
        text-align: center;
        display: block;
```

```
        margin: 10px auto;
}

.file-hash {
        font-style: italic;
        font-weight: bold;
        color: rgb(16, 175, 16);
}

.container-history {
        background-color: var(--history-background-color);
        border: 2px solid var(--basic-text-color);
        border-radius: 10px;
        padding: 30px;
        margin: 50px auto;
        width: 90%;
        max-width: 1600px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}

.container-history h1 {
        text-align: center;
        font-size: 28px;
        color: var(--basic-text-color);
        margin-bottom: 20px;
}

.table-container {
        overflow-x: auto;
}

.history-table {
        width: 100%;
        border-collapse: collapse;
        border-radius: 5px;
        margin-top: 20px;
}

.history-table th,
.history-table td {
        border: 1px solid var(--history-text-color);
        padding: 10px;
        text-align: center;
        font-size: 16px;
}

.history-table th {
        background-color: var(--history-input-color);
        color: var(--history-background-color);
```

```
}

.history-table tr:nth-child(even) {
	background-color: #ffffff;
}

.history-table tr:nth-child(odd) {
	background-color: #f2f4fa;
}

.history-table .status-success {
	color: var(--encrypt-text-color);
	font-weight: bold;
}

.history-table .status-failed {
	color: var(--decrypt-text-color);
	font-weight: bold;
}

.no-data {
	text-align: center;
	font-size: 18px;
	color: var(--basic-text-color);
	margin-top: 20px;
}

.back-button {
	display: inline-block;
	margin-top: 20px;
	text-decoration: none;
	color: var(--history-background-color);
	background-color: var(--history-input-color);
	font-size: 16px;
	padding: 15px 20px;
	border: 2px solid var(--history-input-color);
	border-radius: 5px;
	transition: 0.3s;
	&:hover {
		background-color: var(--history-background-color);
		color: var(--history-input-color);
		transform: translateY(-2px);
	}
}

#encodeFile,
#decodeFile {
	display: none;
}
```

Додаток Г. Ілюстративний матеріал

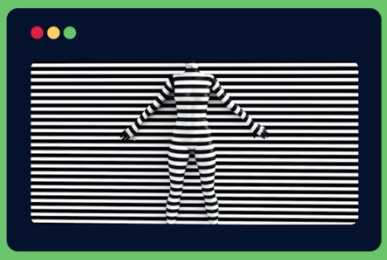




Аналіз предметної області

Захист авторських прав в цифрових аудіофайлах є актуальним завданням у музичній індустрії, особливо для студійних записів високої якості. Одним із перспективних напрямів є застосування методів стеганографії.

Стеганографія — це спосіб приховання інформації всередині інших даних, наприклад, в аудіо. На відміну від криптографії, вона не шифрує повідомлення, а маскує сам факт його існування.



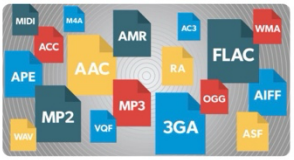
Цифрові водяні знаки (ЦВЗ) є різновидом стеганографії, де в медіафайл вбудовується службова інформація про автора, що дозволяє ідентифікувати правовласника навіть після копіювання або розповсюдження.

Аудіо та вибір формату

Аудіо — це цифрове подання звукової інформації, яке зберігається у файлах різних форматів. Формат визначає спосіб кодування, якість звучання та розмір файлу.

Серед популярних форматів аудіо:
MP3 і AAC — це стиснені з втратами формати. Вони мають невеликий розмір, але значно змінюють спектр сигналу, що робить їх непридатними для стеганографії на основі фазового кодування.
WAV — хоч і зберігає сигнал без втрат, часто використовується з PCM-кодуванням, яке не завжди підходить для точного контролю фазових компонентів.
FLAC — стискає без втрат, але має складнішу структуру і вносить обмеження на безпосередню обробку фази сигналу.

AIFF — формат без втрат, який зберігає повну звукову інформацію, включаючи фазові характеристики. Саме тому він був обраний для реалізації методу фазового кодування в системі захисту авторських прав.

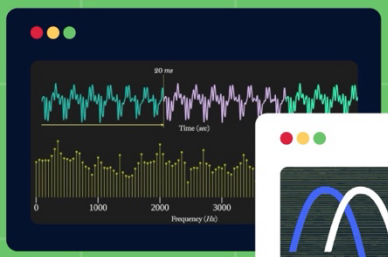


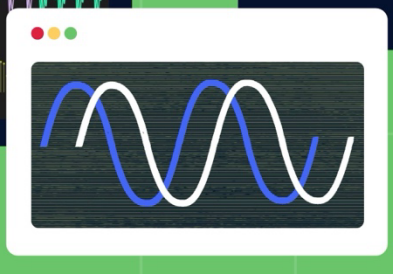
Метод фазового кодування

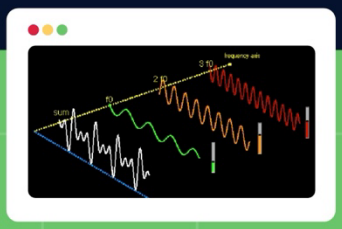
Фазове кодування — це стеганографічний метод, який вбудовує приховану інформацію в аудіосигнал шляхом зміни фазових компонентів його спектру.

Особливість цього підходу полягає в тому, що людське вухо майже не сприймає змін у фазі сигналу, тому модифікація не впливає на якість звучання. Це робить метод малопомітним і придатним для захисту авторських прав.

Алгоритм виконує перетворення сигналу у частотну область (за допомогою FFT), вбудовує дані у фазу спектру та зберігає змінений сигнал у форматі AIFF. При цьому зберігається висока якість звучання та забезпечується стійкість до перетворень, які не змінюють фазу.







Порівняння інших методів стеганографії

Метод стеганографії	Непомітність	Стійкість	Обсяг даних	Складність реалізації	Швидкодія
LSB (Least Significant Bit)	Висока при правильному використанні	Низька (вразливий до компресії та фільтрації)	Високий	Низька	Висока
Ехо-кодування	Середня	Середня	Середній	Середня	Середня
Спектральне кодування	Висока	Висока	Низький	Висока	Низька
Фазове кодування	Дуже висока	Висока	Низький–середній	Середня	Середня

Метод фазового кодування було обрано з огляду на його найвищі показники непомітності та високий рівень стійкості. Порівняно з іншими стеганографічними методами, він забезпечує найменше спотворення сигналу, що особливо важливо для аудіозаписів формату AIFF.

Наприклад, метод LSB має високу швидкодію та дозволяє вбудовувати великі обсяги даних, однак вразливий до компресії та фільтрації, через що не підходить для надійного захисту авторських прав. Ехо-кодування є простішим, але поступається за всіма ключовими параметрами. Спектральне кодування забезпечує високу стійкість, однак складне у реалізації та менш ефективне за швидкодією.

На цьому фоні фазове кодування показує збалансовані характеристики — дуже високу непомітність, високу стійкість, середній рівень складності та достатню швидкодію. Саме тому воно було обране як основа для реалізації захисту студійних аудіозаписів.

Метод фазового кодування

Метод фазового кодування починається з розбиття AIFF-аудіосигналу на блоки однакової довжини. Це дозволяє працювати з сигналом у частотній області окремо для кожного фрагмента. Далі до кожного блоку застосовується дискретне перетворення Фур'є, яке реалізується через швидке перетворення Фур'є (FFT).

$$X_k[m] = \sum_{n=0}^{N-1} x_k[n] \cdot e^{-j2\pi mn/N}$$

У результаті перетворення кожен спектр представляється через модуль і фазу. Модуль відповідає інтенсивності звуку на певній частоті, а фаза – за положення хвилі в часі. Саме фазова складова є ключовою в методі фазового кодування. Вона використовується для вбудовування цифрового водяного знаку (ЦВЗ), оскільки людське вухо є менш чутливим до змін фази, ніж до змін амплітуди.

$$|X_k[m]| = \sqrt{\mathrm{R}^2(X_k[m]) + \mathrm{I}^2(X_k[m])},$$
$$\phi_k[m] = \arctan\left(\frac{\mathrm{I}(X_k[m])}{\mathrm{R}(X_k[m])}\right)$$

Метод фазового кодування

Саме фазова складова використовується для вбудовування інформації, оскільки вона менш помітна для людського слуху. Кожен біт вбудовується у фазу гармоніки: для біта «0» фаза залишається незмінною, для біта «1» до неї додається фіксоване зміщення дельти.

$$\phi_k'[m] = \phi_k[m] + b \cdot \Delta\phi$$
$$X_k'[m] = |X_k[m]| \cdot e^{j\phi_k'[m]}$$

На основі змінених фаз формується новий спектр.Після цього до кожного спектра застосовується зворотне швидке перетворення Фур'є для повернення сигналу у часову область. Це дає змінений блок звуку.

$$x_k'[n] = \frac{1}{N}\sum_{m=0}^{N-1} X_k'[m] \cdot e^{j2\pi mn/N}$$

Блоки об'єднуються у фінальний аудіофайл, який містить приховане повідомлення у фазі. Для зчитування інформації алгоритм повторює ті самі кроки у зворотному порядку. Визначення біта відбувається на основі величини зміщення фази.

$$b = 0, \quad \text{якщо } \phi_k[m] \in [0, \theta]$$
$$b = 1, \quad \text{якщо } \phi_k[m] \in [\theta, 2\theta]$$

Вбудовування ЦВЗ починається з завантаження AIFF-файлу та перевірки його параметрів (формат, частота дискретизації).

Сигнал розбивається на блоки (наприклад, по 1024 семпли) і до кожного застосовується FFT для переходу в частотну область.

Фази гармонік використовуються для приховування повідомлення. Кожен біт вбудовується шляхом зміни фази у визначених діапазонах (наприклад, 0–π/2 для «0», π–3π/2 для «1»).

Після вбудовування всіх бітів виконується IFFT, і блоки збираються у новий аудіосигнал.

Готовий сигнал зберігається у форматі AIFF без втрати якості. Вбудований ЦВЗ залишається непомітним при прослуховуванні.

Алгоритм роботи процесу вбудовування ЦВЗ

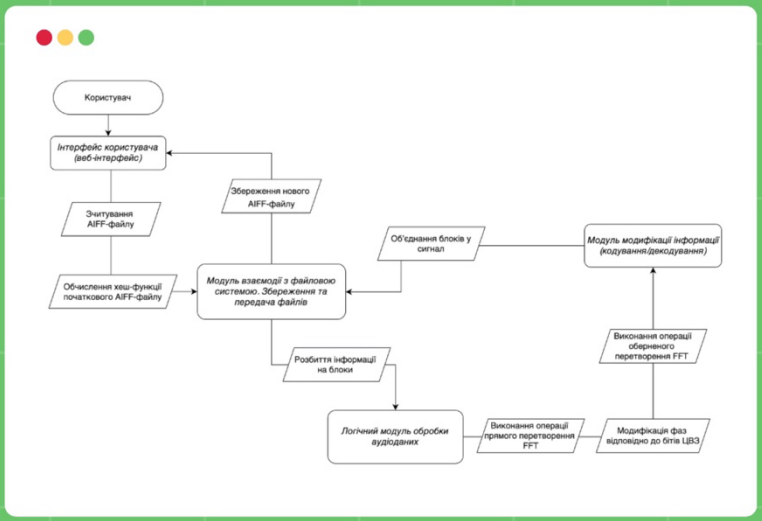
Схема взаємодії компонентів архітектури ПЗ

Інтерфейс – приймає AIFF-файли, ім'я, прізвище, повідомлення; запускає кодування або декодування.

Логічний аудіомодуль – зчитує файл, виконує FFT/IFFT, змінює або зчитує фази.

Модуль кодування або декодування – перетворює текст у біти, вбудовує або витягує їх.

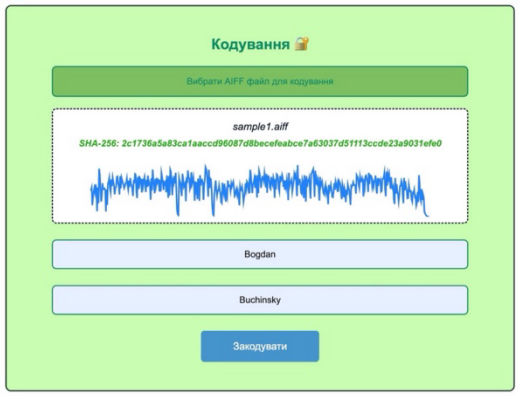
Модуль взаємодії з файловою системою – зберігає файли, веде журнал, працює з базою даних.



Інструменти розробки

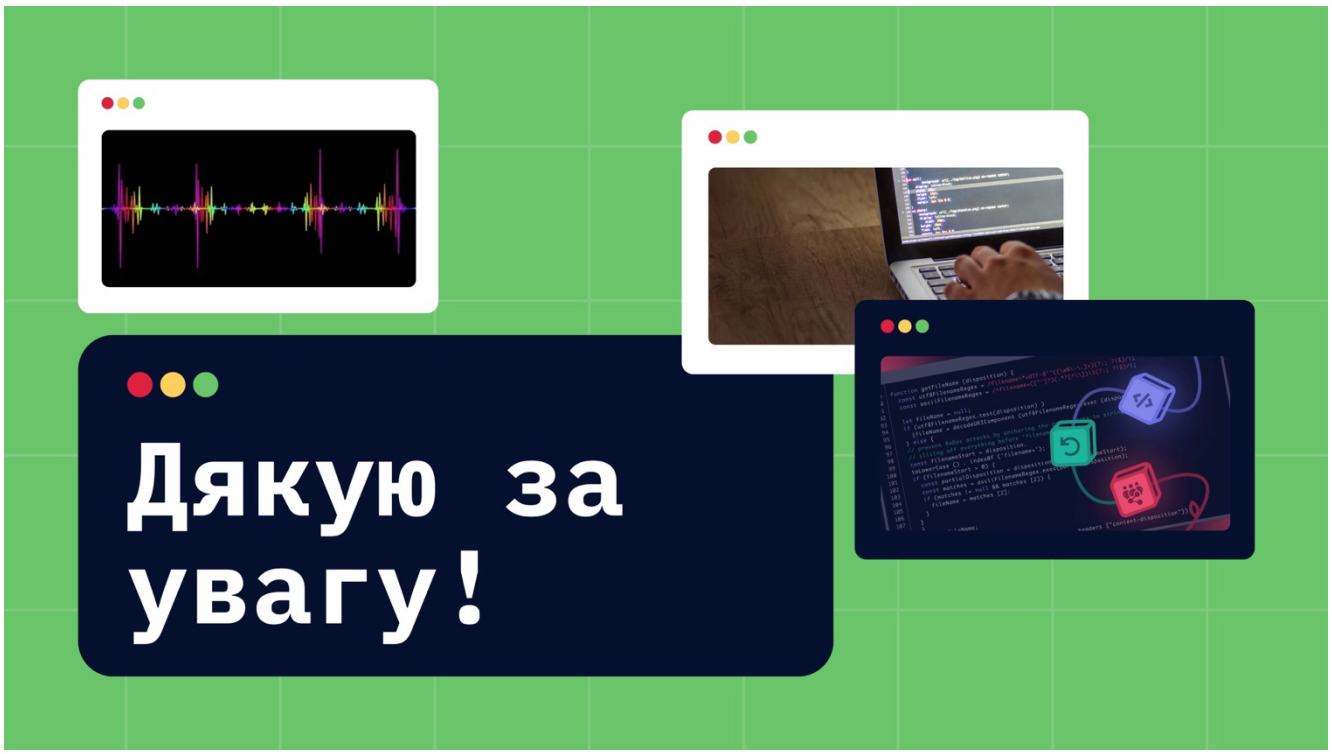
Для розробки системи використовувався *Flask* — для створення серверної частини і обробки запитів. Веб-інтерфейс реалізовано за допомогою *HTML* і *CSS*, що забезпечує зручну взаємодію користувача з програмою. Основна логіка коду написана на *Python*. Для збереження інформації про оброблені файли та операції застосовується база даних *SQLite*.

Інтерфейс користувача



Кодування повідомлення

Декодування повідомлення


Дякую за
увагу!

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка програми захисту авторських прав студійних аудіозаписів формату високої якості AIFF на основі методу фазового кодування

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Карпінець В.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	98,78%
Загальна схожість	1,22%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Бучинський Б.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

☐ **Допустити до захисту**

_____

_____

_____

_____

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(підпис) (прізвище, ініціали, посада)