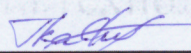


Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

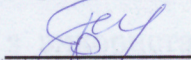
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи магістра
на тему: «Методи та технології забезпечення
конфіденційності та непомітності передавання даних на
основі стеганографії і симетричного шифрування»

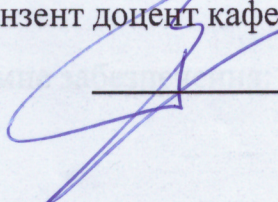
Виконав студент 2-го курсу групи КБм-24-1
спеціальності 125 «Кібербезпека та захист інформації»

 Іван КУЛИКІВСЬКИЙ

Керівник старший викладач кафедри КІ та КБ.

 Олександра ПОКОТИЛО

Рецензент доцент кафедри КІ та КБ, к.т.н.,

 Олександр ПІРОГ

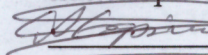
Житомир – 2025

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека та захист інформації»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
6 жовтня 2025 р.

ЗАВДАННЯ

на випускн кваліфікаційну роботу магістра

Студент: Куликівський Іван Дмитрович

Тема роботи: «Методи та технології забезпечення конфіденційності та непомітності передавання даних на основі стеганографії і симетричного шифрування»

затверджена Наказом університету від 6 жовтня 2025 р. № 566/с

Термін здачі студентом закінченої роботи 24 грудня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):

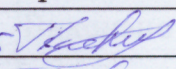
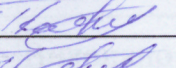
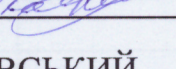
- Процес прихованої передачі та захисту конфіденційної інформації в незахищених каналах зв'язку з використанням цифрових мультимедійних контейнерів;

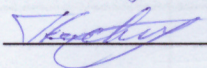
- методи, алгоритми та програмні засоби стеганографічного вбудовування даних у графічні та аудіофайли з використанням цифрових мультимедійних контейнерів.

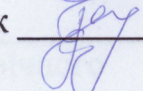
Обладнання та програмне забезпечення:

1. PyCharm Community
2. PySide6
3. Cryptography
4. Pillow (PIL Fork)

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

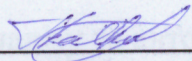
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Покотило О.А.	19.10.2025 	19.10.2025 
2	Покотило О.А.	14.11.2025 	14.11.2025 
3	Покотило О.А.	10.12.2025 	10.12.2025 

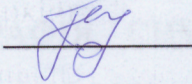
Студент  Іван КУЛИКІВСЬКИЙ

Керівник  Олександра ПОКОТИЛО

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та літературних джерел	03.10.2025	Виконано
2	Проектування архітектури програмного комплексу	21.10.2025	Виконано
3	Програмна реалізація модуля криптографічного захисту	14.11.2025	Виконано
4	Розробка алгоритмів стеганографічного приховування в зображеннях	29.11.2025	Виконано
5	Реалізація модуля аудіо-стеганографії	04.12.2025	Виконано
6	Створення графічного інтерфейсу користувача	05.12.2025	Виконано
7	Розробка логіки механізму правдоподібного заперечення	07.12.2025	Виконано
8	Комплексне тестування та бенчмаркінг	09.12.2025	Виконано
9	Оформлення третього розділу та аналіз результатів	15.12.2025	Виконано
10	Тестування роботи моделі та аналіз результатів	20.12.2025	Виконано
11	Оформлення пояснювальної записки	22.12.2025	Виконано
12	Оформлення презентації	23.12.2025	Виконано
13	Захист	24.12.2025	Виконано

Студент  Іван КУЛИКІВСЬКИЙ

Керівник  Олександра ПОКОТИЛО

АНОТАЦІЯ

Кваліфікаційна робота на тему «Методи та технології забезпечення конфіденційності та непомітності передавання даних на основі стеганографії і симетричного шифрування» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатків.

Текстова частина викладена на 65 сторінках. Пояснювальна записка містить 37 рисунків, 3 таблиці та 31 сторінок додатків. Список використаних джерел містить 24 найменування на 3 сторінках. Загальний обсяг роботи становить 98 сторінок.

Метою даної роботи є розробка та програмна реалізація кросплатформної системи прихованої передачі даних, що поєднує сучасні стандарти симетричного шифрування та методи комп'ютерної стеганографії для забезпечення конфіденційності, цілісності та прихованості інформації в мультимедійних контейнерах.

У першому розділі проведено теоретичний аналіз методів стеганографії та криптографії, обґрунтування їх комбінованого використання.

У другому розділі описано проєктування архітектури додатку, розробка алгоритмів «подвійного дна» та структури бази даних.

У третьому розділі реалізовано практичну частину програми мовою Python, порівняльний бенчмаркінг швидкодії алгоритмів та тестування на прихованість даних.

У додатках наведено лістинг програмного коду, використаного для реалізації моделі інтелектуального аналізу та автоматичного сповіщення.

Ключові слова: СТЕГАНОГРАФІЯ, СИМЕТРИЧНЕ ШИФРУВАННЯ, КРИПТОГРАФІЧНИЙ ЗАХИСТ, AES-256, CHACHA20, МЕТОД LSB, МУЛЬТИМЕДІЙНИЙ КОНТЕЙНЕР, ПРАВДОПОДІБНЕ ЗАПЕРЕЧЕННЯ, КОНФІДЕНЦІЙНІСТЬ ДАНИХ, PYTHON, PYSIDE6.

ANNOTATION

The qualification work on the topic "Methods and Technologies for Ensuring Confidentiality and Imperceptibility of Data Transmission Based on Steganography and Symmetric Encryption" consists of a list of abbreviations, an introduction, three chapters, conclusions, a list of references, and appendices.

The text part is presented on 65 pages. The explanatory note contains 37 figures, 3 tables, and 31 pages of appendices. The list of references contains 24 items on 3 pages. The total volume of the work is 98 pages.

The aim of this work is the development and software implementation of a cross-platform system for covert data transmission, which combines modern standards of symmetric encryption and methods of computer steganography to ensure confidentiality, integrity, and concealment of information in multimedia containers.

The first chapter presents a theoretical analysis of steganography and cryptography methods and provides a justification for their combined use.

The second chapter describes the design of the application architecture, the development of "double bottom" algorithms, and the database structure.

The third chapter covers the practical implementation of the program in Python, comparative benchmarking of algorithm performance, and testing for data concealment.

The appendices contain the listing of the program code used for the implementation of the developed software system.

Keywords: STEGANOGRAPHY, SYMMETRIC ENCRYPTION, CRYPTOGRAPHIC PROTECTION, AES-256, CHACHA20, LSB METHOD, MULTIMEDIA CONTAINER, PLAUSIBLE DENIABILITY, DATA CONFIDENTIALITY, PYTHON, PYSIDE6.

ЗМІСТ

ВСТУП	9
РОЗДІЛ 1. ОГЛЯД МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ ТА СИСТЕМ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ	10
1.1. Аналіз методів комп'ютерної стеганографії: класичний та модифікований алгоритми LSB	10
1.2. Огляд сучасних стандартів симетричного шифрування (AES, ChaCha20) та їх порівняльна характеристика	15
1.3. Теоретичні аспекти реалізації механізмів правдоподібного заперечення та генерації псевдовипадкових послідовностей	19
1.4. Постановка завдання дослідження	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ МОДЕЛІ СИСТЕМИ КОНФІДЕНЦІЙНОЇ ПЕРЕДАЧІ ДАНИХ	25
2.1. Вибір технологій та криптографічних примітивів для побудови системи	25
2.2. Проєктування архітектури багаторівневого стеганографічного контейнера.....	28
2.3. Розробка модифікованого алгоритму вбудовування даних на основі псевдовипадкових послідовностей.....	31
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ	36
3.1. Програмна реалізація модулів шифрування та стеганографічного приховування	36
3.2. Порівняльний аналіз ефективності криптографічних алгоритмів AES- 256 та ChaCha20.....	48
3.3. Тестування стеганографічної стійкості та дослідження ємності мультимедійних контейнерів	52
ВИСНОВКИ.....	63
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	64
ДОДАТКИ.....	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AES – Advanced Encryption Standard
AES-NI – Advanced Encryption Standard New Instructions
API – Application Programming Interface
CPU – Central Processing Unit
GUI – Graphical User Interface
HMAC – Hash-based Message Authentication Code
IV – Initialization Vector
LSB – Least Significant Bit
OS – Operating System
PBKDF2 – Password-Based Key Derivation Function 2
PCM – Pulse-Code Modulation
PNG – Portable Network Graphics
PRNG – Pseudo-Random Number Generator
RAM – Random Access Memory
SHA – Secure Hash Algorithm
UX – User Experience
WAV – Waveform Audio File Format

ВСТУП

Актуальність теми. У сучасних умовах тотальної цифровізації та посилення контролю за інформаційним простором питання конфіденційності набувають критичного значення. Сам факт передачі зашифрованої інформації часто привертає увагу систем моніторингу та зловмисників. У ситуаціях, коли використання криптографії заборонено або контролюється, виникає потреба не лише у захисті змісту повідомлення, а й у приховуванні самого факту його існування.

Метою кваліфікаційної роботи є розробка та програмна реалізація кросплатформної системи прихованої передачі даних, що поєднує сучасні стандарти симетричного шифрування та методи комп'ютерної стеганографії для забезпечення конфіденційності, цілісності та прихованості інформації в мультимедійних контейнерах.

Визначена мета обумовлює наступні завдання:

- спроектувати архітектуру програмного комплексу, що підтримує модульність, та розробити механізм «правдоподібного заперечення» (Plausible Deniability) для захисту від примусового розкриття ключів;
- реалізувати програмний засіб мовою Python із використанням бібліотек PySide6 та Cryptography, забезпечивши підтримку алгоритмів AES-256 та ChaCha20;
- провести порівняльний аналіз швидкодії криптографічних алгоритмів та виконати тестування стеганографічної стійкості сформованих контейнерів (зображень та аудіо).

Об'єктом дослідження є процес прихованої передачі та захисту конфіденційної інформації в незахищених каналах зв'язку з використанням цифрових мультимедійних контейнерів.

Предметом дослідження є методи, алгоритми та програмні засоби стеганографічного вбудовування даних у графічні та аудіофайли з використанням симетричного шифрування та механізму правдоподібного заперечення.

РОЗДІЛ 1. ОГЛЯД МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ ТА СИСТЕМ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ

1.1. Аналіз методів комп'ютерної стеганографії: класичний та модифікований алгоритми LSB

В умовах стрімкого розвитку інформаційних технологій та глобалізації цифрового простору питання захисту інформації набуває критичного значення. Традиційні методи криптографічного захисту, що перетворюють повідомлення на нечитабельний шифротекст, успішно вирішують задачу забезпечення конфіденційності. Проте в багатьох сценаріях сам факт наявності зашифрованого листування може викликати підозру у третьої сторони (зловмисників, аналітиків, цензорів). Саме тут на перший план виходить стеганографія — наука і мистецтво приховування факту передачі інформації шляхом вбудовування її в інші, на вигляд невинні, об'єкти [1].

На відміну від криптографії, яка захищає зміст повідомлення, стеганографія захищає сам факт його існування. Основним поняттям у комп'ютерній стеганографії є поняття "контейнера" (cover object) — цифрового файлу, який використовується для приховування секретних даних. Результат вбудовування називається "стегоконтейнером" (stego-object). Найпоширенішими типами контейнерів сьогодні є мультимедійні файли: цифрові зображення, аудіо- та відеофайли. Це зумовлено їхньою високою інформаційною надлишковістю, що дозволяє вносити незначні зміни в структуру файлу, які залишаються непомітними для органів чуття людини [2].

Серед усіх можливих цифрових носіїв, растрові зображення займають провідне місце у стеганографічних системах. Це пояснюється особливостями сприйняття візуальної інформації людиною (Human Visual System — HVS). Людське око має обмежену чутливість до незначних змін кольору та яскравості, особливо на ділянках зі складною текстурою або високою частотою зміни кольору.

Комп'ютерне представлення зображення — це матриця пікселів. У найпоширеніших форматах (BMP, PNG, JPEG) кожен піксель кодується певною кількістю бітів. Для повноколірних зображень (True Color) зазвичай використовується 24-бітове кодування, де кожен піксель формується трьома базовими кольоровими каналами: червоним (Red), зеленим (Green) та синім (Blue). Кожен канал займає 1 байт (8 бітів), що дозволяє відобразити $2^8 = 256$ відтінків кожного кольору. Комбінація трьох каналів дає можливість відобразити $256 * 256 * 256 = 16,7$ мільйона кольорів.

Така структура даних створює ідеальні умови для приховування інформації. Зміна значення інтенсивності одного з каналів на одиницю (наприклад, зміна яскравості червоного кольору зі значення 150 на 151) є абсолютно нерозрізненною для людського ока, але може бути легко зафіксована програмно. Саме на цьому принципі базується найвідоміша група стеганографічних методів — методи заміни найменшого значущого біта (LSB — Least Significant Bit) [3].

Метод найменшого значущого біта (LSB) є класичним прикладом просторових методів стеганографії (Spatial Domain Techniques). Його сутність полягає в заміні останнього біта в байті, що кодує колір пікселя, на біт секретного повідомлення.

Математична модель цього процесу виглядає так: нехай C — це оригінальний контейнер (зображення), що складається з набору пікселів $\{x_1, x_2, x_3, \dots, x_n\}$. Кожен піксель (або кольоровий канал пікселя) представлено байтом, значення якого лежить у діапазоні $[0, 255]$. У двійковому поданні значення байта можна записати як:

$$x_i = b_7b_6b_5b_4b_3b_2b_1b_0$$

де b_0 — це найменший значущий біт (LSB), а b_7 — найбільший значущий біт (MSB). Зміна біта b_7 змінить значення байта на $2^7 = 128$, що призведе до кардинальної зміни кольору, яку помітно візуально. Натомість зміна байта b_0 змінить значення лише на $2^0 = 1$.

Алгоритм вбудовування в класичному методі LSB виглядає наступним чином:

1. Секретне повідомлення M перетворюється на потік бітів $m_1, m_2, \dots, m_k$.
2. Вибирається послідовність пікселів контейнера (у класичному варіанті – підряд, починаючи з лівого верхнього кута зображення).
3. Для кожного байта контейнера x_i виконується операція заміни його молодшого біта на біт повідомлення m_i :

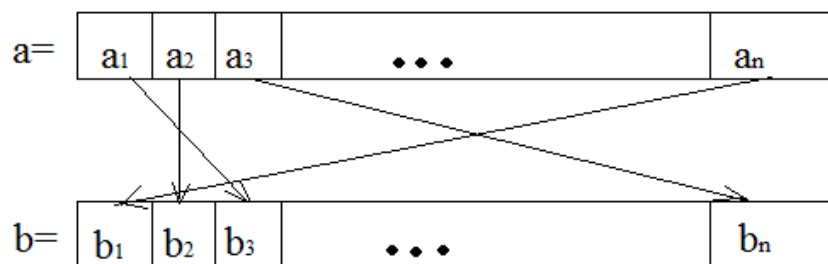


Рисунок 1.1 – Схема заміни бітів у класичному методі LSB

Переваги класичного методу LSB:

1. Висока ємність (Capacity): теоретично, в кожному байті зображення можна приховати 1 біт інформації. Для зображення розміром 800x600 пікселів (True Color) ємність становить 800x600x3 біти = 1.44МБ, що є значним обсягом.
2. Простота реалізації: алгоритм потребує мінімальних обчислювальних ресурсів і легко реалізувати програмно.
3. Візуальна непомітність: зміни в зображенні не сприймаються оком [4].

Проте класичний LSB має критичний недолік — низьку стійкість до статистичного стегоаналізу. Оскільки повідомлення зазвичай має випадковий розподіл бітів (ентропія близька до 1), заміна LSB вносить характерні зміни в статистику кольорів зображення. Найвідомішою атакою на цей метод є атака на основі гістограм або $\$RSS\$$ -аналіз. Якщо в оригінальному зображенні кількість парних і непарних пікселів може суттєво відрізнятися, то після застосування LSB-вбудовування ці кількості вирівнюються (ефект "Pair of Values"), що є чітким індикатором наявності прихованих даних для фахівця з інформаційної безпеки [5].

Більше того, послідовне заповнення зображення даними (зверху вниз) призводить до того, що верхня частина зображення має змінені статистичні характеристики, а нижня — залишається незмінною. Ця аномалія легко детектується навіть простими візуальними фільтрами.

Для усунення недоліків класичного підходу було розроблено низку модифікацій. Однією з них є метод LSB Matching. У цьому методі, якщо біт повідомлення не співпадає з LSB байта контейнера, значення байта не просто замінюється, а випадковим чином збільшується або зменшується на 1. Це дозволяє уникнути характерної асиметрії в гістограмі, ускладнюючи роботу стегоаналітиків [6].

Однак найбільш ефективним способом підвищення стійкості, який використовується в сучасних системах, є відмова від послідовного запису даних. Цей підхід отримав назву Random LSB (LSB з псевдовипадковим розподілом інтервалів).

Суть методу Random LSB полягає в тому, що секретні дані вбудовуються не в сусідні пікселі підряд, а розкидаються по всій площині зображення відповідно до певного закону. Для реалізації цього підходу використовується генератор псевдовипадкових чисел (ГПВЧ або PRNG — Pseudo-Random Number Generator).

Алгоритм роботи Random LSB:

1. Користувач задає пароль (стего-ключ).
2. На основі пароля за допомогою криптографічної хеш-функції (наприклад, SHA-256) генерується початкове значення (seed) для ГПВЧ. Це гарантує, що одна й та сама послідовність координат буде згенерована тільки при введенні правильного пароля.
3. Генератор створює послідовність унікальних координат пікселів, кількість яких відповідає довжині повідомлення.
4. Здійснюється вбудовування бітів повідомлення у пікселі за отриманими координатами методом LSB.

Використання Random LSB надає системі суттєві переваги в контексті безпеки:

1. Захист від локального аналізу: атакуючий не знає, які саме пікселі містять секретну інформацію. Спроба проаналізувати частину зображення не дасть результату, оскільки зміни розподілені рівномірно і виглядають як природний шум матриці цифрової камери або сканера [7].

2. Залежність від ключа: без знання пароля неможливо не тільки розшифрувати повідомлення, а й навіть зібрати його біти в правильному порядку. Це реалізує принцип Керкгоффса, згідно з яким безпека системи повинна залежати лише від ключа, а не від таємності алгоритму.

3. Підвищення стійкості до візуальних атак: оскільки дані розподіляються по всьому зображенню, відсутні області з різкою зміною статистики (наприклад, "шумний" верх і "чистий" низ), що робить стегоконтейнер візуально більш цілісним.

Варто також зазначити про обмеження методу LSB. Основною проблемою є чутливість до будь-яких перетворень зображення. Стиснення з втратами (наприклад, збереження у формат JPEG), зміна розміру, обрізка або корекція кольору призводять до зміни значень пікселів, а отже — до руйнування прихованих у молодших бітах даних. Тому для використання методу LSB критично важливим є використання форматів стиснення без втрат, таких як PNG (Portable Network Graphics) або BMP, які забезпечують побітну точність збереження даних [8].

Таким чином, аналіз методів показує, що хоча класичний алгоритм LSB є базовим і простим, для побудови надійної системи прихованої передачі даних необхідно використовувати його модифікації, зокрема Random LSB, у поєднанні з криптографічними генераторами псевдовипадкових послідовностей. Це дозволяє нівелювати основні вектори атак стегоаналізу та забезпечити належний рівень конфіденційності каналу передачі.

1.2. Огляд сучасних стандартів симетричного шифрування (AES, ChaCha20) та їх порівняльна характеристика

Криптографічний захист інформації є фундаментальним компонентом будь-якої системи безпечної передачі даних. У контексті задач стеганографії, криптографія виконує роль першого рубежу оборони: навіть якщо факт наявності прихованого каналу буде виявлено, зломисник не повинен мати можливості прочитати зміст повідомлення без знання секретного ключа.

Основним класом алгоритмів, що використовуються для шифрування великих обсягів даних (файлів, мультимедіа), є алгоритми симетричного шифрування. Їхньою характерною рисою є використання одного й того ж секретного ключа як для процесу зашифрування (encryption), так і для розшифрування (decryption) [9]. За принципом дії симетричні шифри поділяються на два основні класи: блокові (Block Ciphers) та потокові (Stream Ciphers). У даній роботі для реалізації програмного комплексу було обрано двох найяскравіших представників цих класів — AES та ChaCha20.

Алгоритм AES (Advanced Encryption Standard), також відомий під оригінальною назвою Rijndael, був прийнятий Національним інститутом стандартів і технологій США (NIST) як стандарт FIPS 197 у 2001 році. Він прийшов на зміну застарілому стандарту DES, який став вразливим до атак повного перебору через малу довжину ключа (56 біт) [10].

Основні характеристики AES:

- Розмір блоку: фіксований - 128 біт (16 байт).
- Довжина ключа: 128, 192 або 256 біт.
- Кількість раундів: 10 (ключ 128 біт), 12 (192 біт), 14 (256 біт).

Процес шифрування AES оперує матрицею байтів розміром 4x4, яка називається "станом" (State). Кожен раунд шифрування (окрім останнього) складається з чотирьох оборотних перетворень, що реалізують принципи розсіювання (diffusion) та перемішування (confusion) Клода Шеннона [11]:

1. SubBytes (Підстановка байтів): нелінійне перетворення, де кожен байт стану замінюється іншим байтом зі спеціальної таблиці підстановки (S-Box). S-

Вох побудований на основі операції обернення в полі Галуа $GF(2^8)$, що забезпечує високу стійкість до лінійного та диференціального криптоаналізу.

2. ShiftRows (Змішування рядків): циклічний зсув рядків матриці стану. Перший рядок не зсувається, другий зсувається на 1 байт вліво, третій — на 2, четвертий — на 3. Це перетворення забезпечує перемішування даних між стовпцями.

3. MixColumns (Змішування стовпців): кожен стовпець матриці стану розглядається як поліном над $GF(2^8)$ і множиться на фіксований поліном $c(x)$. Ця операція розмазує вплив кожного байта на весь стовпець.

4. AddRoundKey (Додавання ключа раунда): операція побітового додавання за модулем 2 (XOR) матриці стану з ключем поточного раунду, який генерується з основного ключа за процедурою Key Expansion.

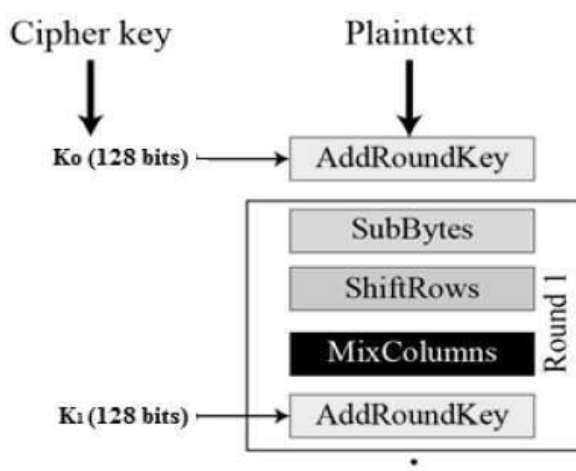


Рисунок 1.2 - Схема раунду шифрування AES

Сьогодні AES є де-факто світовим стандартом. Більшість сучасних процесорів (Intel, AMD, ARM) мають набір інструкцій AES-NI (AES New Instructions), які реалізують етапи шифрування на апаратному рівні. Це дозволяє досягати надзвичайно високої швидкості обробки даних (кілька гігабайтів на секунду) при мінімальному навантаженні на центральний процесор [12].

ChaCha20 — це сучасний високошвидкісний потоковий шифр, розроблений Деніелом Бернштейном (Daniel J. Bernstein) у 2008 році як модифікація алгоритму Salsa20. У 2015 році він був стандартизований IETF у

документі RFC 7539 і рекомендований як безпечна альтернатива AES, особливо для мобільних пристроїв та програмних реалізацій [13].

На відміну від блокових шифрів, потокові шифри генерують нескінченну псевдовипадкову послідовність бітів (гаму шифру або keystream), яка потім об'єднується з відкритим текстом за допомогою операції XOR.

Архітектура ChaCha20 базується на схемі ARX (Add-Rotate-Xor). Це означає, що алгоритм використовує лише три найпростіші процесорні операції:

- Додавання 32-бітних чисел (Addition).
- Циклічний зсув (Rotation).
- Побітове виключне АБО (XOR).

Така простота має критичне значення для безпеки. В алгоритмі ChaCha20 відсутні таблиці підстановок (S-Boxes) та множення в полях Галуа, які використовуються в AES. Це робить ChaCha20 невразливим до так званих атак побічними каналами на основі часу виконання (cache-timing attacks). В AES час доступу до таблиць S-Box може залежати від значень секретного ключа, що теоретично дозволяє зловмиснику відновити ключ, вимірюючи час виконання операцій. ChaCha20 виконується за сталий час (constant time) незалежно від вхідних даних, що робить його надзвичайно безпечним у чисто програмних реалізаціях [14].

Внутрішній стан ChaCha20 представляється у вигляді матриці 4x4 з 32-бітних слів (всього 512 біт). Стан ініціалізується константами, секретним ключем (256 біт), лічильником блоку та унікальним значенням (nonce).

Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Counter	Nonce	Nonce	Nonce

Рисунок 1.3 - Структура матриці стану ChaCha20

Генерація гами відбувається шляхом виконання 20 раундів перемішування даних у цій матриці. Кожні два раунди (відомі як "подвійний раунд") застосовують функцію Quarter Round до всіх стовпців, а потім до всіх діагоналей матриці. Це забезпечує швидку дифузію бітів ключа по всьому стану.

Вибір між AES та ChaCha20 часто залежить від апаратної платформи та вимог до безпеки. Основні відмінності зведено в таблицю:

Таблиця 1.1

Порівняльний аналіз AES-256 та ChaCha20

<i>Характеристика</i>	<i>AES-256</i>	<i>ChaCha20</i>
Тип шифру	Блоковий	Потоковий
Розмір ключа	256 біт	256 біт
Апаратне прискорення	Є (AES-NI)	Немає
Стійкість до timing attack	Залежить від реалізації	Висока
Основна сфера	Корпоративні мережі, SSD	Мобільні пристрої, TLS 1.3

Дослідження показують, що на сучасних процесорах з підтримкою інструкцій AES-NI алгоритм AES-256 часто випереджає ChaCha20 за швидкістю. Однак на старих системах, мікроконтролерах або мобільних пристроях з ARM-процесорами без криптографічних розширень (наприклад, бюджетні смартфони), ChaCha20 працює значно швидше — іноді в 3-4 рази [15]. Саме тому компанія Google перейшла на використання ChaCha20 для шифрування трафіку в браузері Chrome на мобільних пристроях Android.

Реалізація обох алгоритмів дозволяє створити універсальний інструмент. Користувач отримує можливість обирати між AES, який є обов'язковим для багатьох державних та корпоративних регламентів, та сучасним швидкісним шифром ChaCha20, який забезпечує кращу захищеність від атак побічними каналами в програмному середовищі.

1.3. Теоретичні аспекти реалізації механізмів правдоподібного заперечення та генерації псевдовипадкових послідовностей

У класичній моделі інформаційної безпеки основним завданням є захист конфіденційності даних від пасивного перехоплення. Проте в реальних умовах існує клас загроз, де злоумисник має фізичний доступ до носія інформації або самого користувача і може застосовувати методи примусу для отримання ключів шифрування. Цей тип атак у криптографічній літературі отримав назву «терморектальний криптоаналіз» (rubber-hose cryptanalysis) [16].

У таких сценаріях звичайне шифрування стає неефективним: якщо злоумисник знає, що файл зашифровано, він може вимагати пароль під загрозою фізичного або юридичного впливу. Єдиним ефективним засобом протидії є використання концепції правдоподібного заперечення (Plausible Deniability).

Суть правдоподібного заперечення полягає в організації зберігання даних таким чином, щоб неможливо було математично довести існування прихованої інформації навіть після того, як користувач видав один із ключів шифрування. У контексті стеганографічних систем це реалізується через механізм подвійної стеганографії (Double Steganography) або багаторівневих контейнерів.

Архітектура такої системи передбачає наявність двох (або більше) незалежних повідомлень в одному контейнері:

1. Декой-повідомлення (Decoy message): інформація низької цінності, яку користувач готовий розкрити під примусом. Вона захищена “публічним” паролем.

2. Секретне повідомлення (Hidden message): критично важлива інформація, захищена “секретним” паролем.

Ключовою технічною вимогою до реалізації є неможливість розрізнити зашифровані дані секретного повідомлення від випадкового шуму (random noise), який залишається після запису декой-повідомлення. Якщо контейнер заповнено не повністю, вільні пікселі зображення містять дані, що статистично нагадують білий шум. Система правдоподібного заперечення використовує цей "шум" для запису другого повідомлення [17].

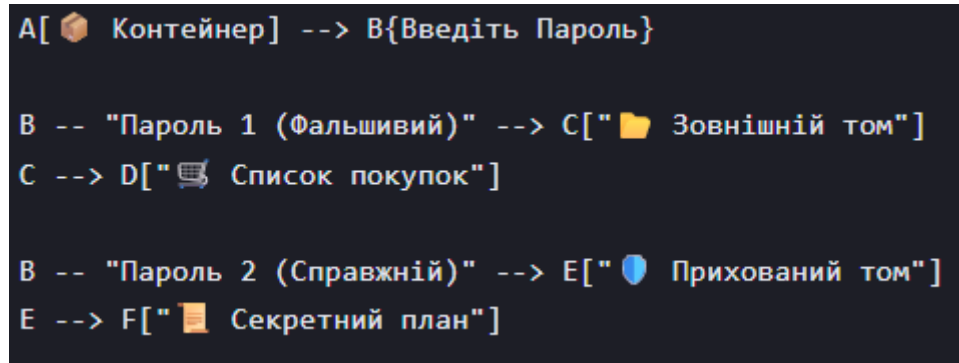


Рисунок 1.4 - Принцип правдоподібного заперечення

Головною проблемою при цьому є уникнення колізій (перезапису даних). Оскільки обидва повідомлення записуються в один контейнер, алгоритм вбудовування другого повідомлення повинен знати, які пікселі вже зайняті першим повідомленням, не розкриваючи при цьому факту існування другого. Це вирішується шляхом використання детермінованих псевдовипадкових послідовностей координат, що залежать від хешу пароля.

Для реалізації методу Random LSB та механізму подвійного дна критично важливим є використання якісних генераторів псевдовипадкових чисел (ГПВЧ або PRNG — Pseudo-Random Number Generator). У той час як для звичайних прикладних задач достатньо стандартних бібліотечних генераторів, для задач криптографії та стеганографії необхідні криптографічно стійкі генератори (CSPRNG — Cryptographically Secure PRNG) [18].

Основна вимога до CSPRNG полягає в тому, що, знаючи частину вихідної послідовності, обчислювально неможливо передбачити наступний біт або відновити попередній. У розроблюваній системі генератор використовується для визначення координат пікселів (x, y), у які будуть вбудовані біти повідомлення.

Алгоритм генерації координат базується на понятті зерна (seed). Процес виглядає наступним чином:

1. Пароль користувача P обробляється функцією формування ключа (KDF), наприклад PBKDF2, для отримання ентропійно насиченого байтового масиву.
2. Отриманий масив (або його частина) використовується як *Seed* для ініціалізації CSPRNG.

3. Генератор продукує послідовність індексів пікселів.

Такий підхід гарантує детермінованість: при введенні того самого пароля завжди генерується одна й та сама послідовність координат, що дозволяє коректно витягти повідомлення. Водночас, для стороннього спостерігача (або для іншого пароля) ця послідовність виглядає як хаотичний набір, що забезпечує рівномірний розподіл навантаження на контейнер і ускладнює статистичний аналіз [19].

Безпека описаної схеми напряму залежить від стійкості пароля. Оскільки користувачі схильні обирати слабкі паролі, система повинна використовувати механізми підсилення ключа (Key Stretching). Стандартом у цій галузі є алгоритм PBKDF2 (Password-Based Key Derivation Function 2), описаний у RFC 2898.

PBKDF2 використовує псевдовипадкову функцію (зазвичай HMAC-SHA256) разом із випадковим значенням, що називається "сіль" (salt). Сіль — це набір випадкових байтів, який генерується для кожного нового шифрування і зберігається у відкритому вигляді разом із контейнером. Використання солі вирішує дві фундаментальні проблеми:

1. Унеможливорює використання “райдужних таблиць” (Rainbow Tables) - попередньо обчислених без хешів для швидкого злому паролів.

2. Гарантує, що два однакові паролі дадуть абсолютно різні ключі шифрування та різні координати вбудовування (через різну сіль), що унеможливорює порівняльний аналіз кількох контейнерів [20].

Таким чином, поєднання криптографічно стійкої генерації координат (CSPRNG), надійного формування ключів (PBKDF2) та архітектури подвійного вбудовування дозволяє створити систему, стійку як до математичного криптоаналізу, так і до методів соціальної інженерії та примусу.

Окремим класом задач є стеганографічне приховування даних у цифрових аудіосигналах. На відміну від статичних зображень, де інформація приховується у просторовій області, аудіо-стеганографія працює у часовій області (Temporal Domain). Найпоширенішим форматом для такого вбудовування є імпульсно-кодова модуляція (PCM), що використовується у файлах типу WAV.

Теоретичною основою застосування методу найменшого значущого біта (LSB) для аудіо є недосконалість людської слухової системи (HAS — Human Auditory System). Хоча слух людини є надзвичайно чутливим до перепадів частот (диференціальний поріг сприйняття становить близько 3,6 Гц у діапазоні 1–2 кГц), він має низьку чутливість до наявності адитивного білого шуму низької амплітуди.

Важливим етапом підвищення стійкості стеганографічної системи є попередня обробка вбудовуваного повідомлення. Згідно з теоремою Шеннона, інформаційна ентропія повідомлення визначає ступінь його хаотичності. Природні мови (українська, англійська) мають низьку ентропію через наявність статистичних закономірностей (частота літер, повторювані слова). Це створює вразливість перед статистичним стегоаналізом, який може виявити аномалії в розподілі бітів контейнера.

Для усунення цієї загрози необхідно застосовувати алгоритми стиснення даних без втрат (наприклад, DEFLATE) перед етапом вбудовування. Стиснення виконує дві критичні функції:

1. Зменшення обсягу корисного навантаження (Payload Reduction): чим менше бітів потрібно записати, тим менше пікселів або семплів буде модифіковано, що лінійно знижує ймовірність виявлення стегосистеми.

2. Максимізація ентропії: стиснені дані за своїми статистичними характеристиками наближаються до рівномірного розподілу (білого шуму). Це ускладнює розрізнення між власним шумом цифрової матриці (або мікрофона) та внесеними стеганографічними даними.

З технічної точки зору, алгоритм DEFLATE, реалізований у бібліотеці zlib, комбінує алгоритм LZ77 (усунення повторюваних послідовностей) та кодування Хаффмана (ентропійне кодування). Видалення лінгвістичної надлишковості (Redundancy) є критичним для криптографічної стійкості. Якщо шифрувати нестиснений текст, закономірності мови можуть частково зберігатися або створювати патерни, що полегшують криптоаналіз. Архітектура "спочатку стиснення, потім шифрування" (Compress-then-Encrypt) гарантує, що на вхід

алгоритму шифрування (AES або ChaCha20) подається потік даних з максимально можливою ентропією, що робить вихідний шифротекст статистично невідрізнюваним від випадкової послідовності.

1.4.Постановка завдання дослідження

На основі проведеного аналізу існуючих рішень для прихованої передачі даних, методів комп'ютерної стеганографії та векторів атак на захищені канали зв'язку було встановлено, що традиційні засоби шифрування забезпечують конфіденційність змісту, але не приховують сам факт обміну інформацією, що в умовах моніторингу може викликати підозру. Водночас класичні стеганографічні алгоритми часто вразливі до статистичного аналізу і не завжди дають можливість гарантувати безпеку даних у випадку фізичного вилучення носія та примусу до розкриття пароля. Це підкреслює необхідність розвитку підходів, здатних поєднувати можливості криптографічного захисту з методами приховування факту передачі, які дозволяють забезпечити вищий рівень приватності та захист від компрометації ключів.

Разом із тим, сучасні професійні комплекси стеганографічного захисту часто є складними в налаштуванні, вимагають специфічних знань або працюють виключно через командний рядок, що обмежує їх використання широким колом користувачів. У реальних умовах експлуатації, особливо для забезпечення персональної безпеки при передачі даних через відкриті мережі, оптимальним рішенням стають кросплатформні десктопні системи з інтуїтивно зрозумілим інтерфейсом. Такі системи забезпечують баланс між криптографічною стійкістю та зручністю використання, дозволяючи виконувати гнучке шифрування (AES/ChaCha20), використовувати модифіковані алгоритми вбудовування (Random LSB) та механізми правдоподібного заперечення для надійного захисту інформації.

З огляду на це, тема дослідження є актуальною та зумовлює розробку програмного комплексу для прихованої передачі мультимедійних даних із функцією правдоподібного заперечення, що є метою даної роботи. Для

досягнення мети передбачається використання симетричних криптографічних алгоритмів, методу вбудовування на основі псевдовипадкових послідовностей, а також архітектури «подвійного дна», яка дозволяє приховувати два незалежних повідомлення в одному контейнері. Реалізація здійснюється мовою Python із використанням бібліотеки PySide6 для графічного інтерфейсу та криптографічних бібліотек для забезпечення цілісності даних.

Відповідно, для досягнення поставленої мети було визначено такі задачі:

1. Розробити модель системи конфіденційної передачі даних, що поєднує модифікований метод LSB (Random LSB) на основі хешування пароля із сучасними алгоритмами шифрування (AES-256, ChaCha20) та механізмом правдоподібного заперечення, визначивши її архітектуру та логіку обробки колізій при подвійному запису.

2. Реалізувати програмний комплекс, включно з розробкою графічного інтерфейсу користувача, модулів шифрування та дешифрування файлів і тексту, вбудовування даних у зображення та аудіо, а також інструментарію для моніторингу ємності контейнера в реальному часі.

3. Провести аналіз отриманих результатів та оцінити ефективність роботи системи, порівнявши швидкодію реалізованих криптографічних алгоритмів (Speed Test) та дослідивши вплив стеганографічного вбудовування на якість мультимедійних контейнерів, визначивши сильні сторони розробки та можливі напрями її подальшого удосконалення.

Висновки до першого розділу: у першому розділі, проведений аналіз довів необхідність розробки гібридної системи, що поєднує симетричне шифрування (AES, ChaCha20) та модифіковану стеганографію (Random LSB). Окрему увагу приділено концепції правдоподібного заперечення, реалізація якої дозволить нівелювати загрози, пов'язані з примусовим розкриттям ключів. Сформульовані задачі дослідження визначають подальший напрямок роботи щодо проєктування та програмної реалізації такого комплексу.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОДЕЛІ СИСТЕМИ КОНФІДЕНЦІЙНОЇ ПЕРЕДАЧІ ДАНИХ

2.1. Вибір технологій та криптографічних примітивів для побудови системи

Проектування програмного комплексу для прихованої передачі даних вимагає ретельного підходу до вибору інструментальних засобів, оскільки від цього залежить не лише швидкість розробки, а й надійність криптографічних перетворень, стабільність роботи додатку та зручність для кінцевого користувача. Основними критеріями вибору технологічного стека були: кросплатформність, наявність перевірених криптографічних бібліотек, висока швидкість обробки бінарних даних та можливість створення сучасного графічного інтерфейсу (GUI).

У якості основної мови реалізації обрано Python 3.10+. Цей вибір зумовлений тим, що Python є фактичним стандартом у сфері інформаційної безпеки та наукових досліджень. Він забезпечує високу швидкість прототипування та має потужну екосистему бібліотек для роботи з даними [21]. Крім того, динамічна типізація та вбудовані засоби роботи з великими цілими числами (arbitrary-precision integers) значно спрощують реалізацію математичних алгоритмів, таких як генерація ключів та бітові операції LSB.

Для реалізації інтерфейсу користувача використано бібліотеку PySide6 — офіційний біндинг фреймворку Qt для Python. На відміну від застарілого tkinter або веб-орієнтованих фреймворків (Electron), PySide6 дозволяє створювати нативні, високопродуктивні додатки, що коректно відображаються на Windows, macOS та Linux. Використання механізму сигналів та слотів (Signals & Slots) дозволяє ефективно розділити логіку шифрування та візуальну частину, забезпечуючи чутливість інтерфейсу навіть під час виконання важких криптографічних операцій у фонових потоках (QThread) [23].

Для реалізації алгоритмів AES-256 та ChaCha20 використано бібліотеку cryptography (пакет causal-cryptography). Це рішення продиктоване вимогами

безпеки: самостійна реалізація криптографічних примітивів (rolling your own crypto) вважається критичною помилкою в інженерії безпеки через високий ризик вразливостей побічних каналів (side-channel attacks). Бібліотека cryptography надає високорівневі рецепти та низкорівневі примітиви, що пройшли аудит безпеки та використовують С-розширення (OpenSSL) для максимальної продуктивності [22].

Для маніпуляцій із пікселями зображень обрано бібліотеку Pillow. Вона надає прямий доступ до байтових масивів зображень, підтримує формат PNG без втрат якості (що є критичним для LSB-стеганографії) та дозволяє ефективно зчитувати метадані про розмір та кольорову модель контейнера [24].

Система спроектована за модульним принципом, що забезпечує гнучкість та можливість розширення. Логічна структура додатку розділена на три рівні:

1. Рівень представлення (GUI Layer): відповідає за взаємодію з користувачем, валідацію вхідних даних та оновлює статус виконання.
2. Рівень бізнес-логіки (Controller Layer): керує потоками виконання, ініціює процеси шифрування та оновлює статус виконання.
3. Рівень ядра (Core Layer): містить класи CryptoHandler (шифрування) та StegoHandler (вбудовування), які виконують безпосередньо обробку байтів.

Загальна схема взаємодії компонентів системи наведена на рисунку 2.1.



Рисунок 2.1 - Загальна архітектура розробленого програмного комплексу

Для забезпечення ефективної роботи з бінарними даними до технологічного стека було включено модуль struct. Він використовується для пакування цілочисельних значень (зокрема, розміру повідомлення) у масив

байтів фіксованої довжини (4 байти, Big-Endian), що є необхідним для коректного зчитування заголовків декодером.

Враховуючи обмежену ємність стеганографічних контейнерів (LSC — Least Significant Bit Capacity), критично важливим етапом підготовки даних є їх ефективна компресія. У розробленій системі використано бібліотеку zlib, яка реалізує стандартизований алгоритм стиснення без втрат DEFLATE.

Вибір саме цього алгоритму зумовлений його гібридною архітектурою, яка виконує послідовну оптимізацію потоку даних у два етапи:

1. На першому етапі застосовується варіація словникового методу Лемпеля-Зіва (LZ77). Алгоритм використовує "ковзне вікно" (sliding window) розміром 32 КБ для аналізу вхідного потоку байтів. При виявленні послідовності символів, яка вже зустрічалася раніше у межах вікна, вона замінюється на посилання (пару чисел): відстань до попереднього входження та довжину збігу. Оскільки в природних мовах та програмному коді часто зустрічаються повторювані патерни (слова, префікси, теги), цей етап дозволяє суттєво скоротити фізичний розмір повідомлення, замінюючи цілі фрази на короткі маркери розміром у кілька байтів.

2. На другому етапі результат роботи LZ77 піддається обробці алгоритмом Хаффмана. Цей метод замінює символи (або посилання LZ77) на бітові коди змінної довжини. На основі частотного аналізу будується бінарне дерево, де найбільш вживаним символам присвоюються найкоротші бітові послідовності, а рідкісним — довші. Це дозволяє досягти оптимальної щільності пакування інформації, наближаючись до теоретичної межі ентропії Шеннона.

У програмному коді системи (CryptoHandler) примусово встановлено параметр рівня компресії level=9 (Best Compression). Хоча цей режим вимагає більших обчислювальних ресурсів процесора порівняно зі стандартним рівнем (level=6), він забезпечує максимальне зменшення розміру вихідного файлу. У контексті стеганографії це дає дві критичні переваги:

1. Мінімізація стеганографічного сліду: чим менший обсяг даних вбудовується, тим менша кількість пікселів зображення зазнає змін. Це знижує ймовірність виявлення факту приховування методами статистичного аналізу

2. Криптографічне посилення: стиснення руйнує статистичні закономірності вихідного тексту (частотність літер, біграм). Отриманий псевдовипадковий потік бітів, що подається на вхід алгоритму шифрування, значно ускладнює проведення атак на основі відомого відкритого тексту, оскільки зломисник не може використовувати лінгвістичні патерни для злому ключа.

```
def encrypt(self, data: bytes, password: str, algorithm="AES") -> bytes: 6 usages
    """
    Шифрує дані обраним алгоритмом.
    algorithm: "AES" або "ChaCha20"
    """
    # 1. Стиснення
    compressed_data = zlib.compress(data, level=9)

    # 2. Генерація salt
    salt = os.urandom(self.salt_size)
    key = self._derive_key(password, salt)
```

Рисунок 2.2 - Фрагмент коду методу шифрування з використанням стиснення *zlib*

2.2. Проектування архітектури багаторівневого стеганографічного контейнера

Ключовою особливістю розроблюваної системи є реалізація механізму правдоподібного заперечення (Plausible Deniability). Для досягнення цієї мети було відкинута класичну модель лінійного запису на користь архітектури незалежного багат шарового вбудовування (Independent Multi-layer Embedding).

Цей підхід передбачає розділення процесу приховування інформації на етапи, які є криптографічно та статистично ізольованими один від одного.

Програмна реалізація базується на двопрхідному алгоритмі (Two-pass encoding). Процес створення захищеного контейнера відбувається наступним чином:

1. Перший прохід (Pass 1 - Decoy Layer): система приймає вихідне зображення та формує перший шар даних (відволікаюче повідомлення). Генерація координат вбудовування здійснюється на основі хешу першого пароля. Результатом є проміжний стегоконтейнер, який вже виглядає як файл із шумом, характерним для стеганографії.

2. Другий прохід (Pass 2 - Hidden Layer): якщо користувач активує режим подвійного дна, проміжний контейнер автоматично подається на вхід алгоритму повторно. У нього вбудовується секретне повідомлення з використанням другого пароля.

Така архітектура дозволяє використовувати універсальний модуль стеганографії без необхідності ускладнення логіки внутрішніми перевітками стану пікселів між різними повідомленнями.

Важливим аспектом проєктування є відсутність жорсткої синхронізації між шарами. Оскільки координатні сітки для першого та другого повідомлень генеруються різними зернами (seeds), отриманими від різних паролів, розподіл бітів у контейнері є статистично незалежним.

У даній системі застосовано метод імовірнісного уникнення колізій (Probabilistic Collision Avoidance). Замість ресурсномісткого відстеження зайнятих пікселів першого шару, алгоритм покладається на високу ємність носія. Для зображення формату Full HD (1920×1080) кількість доступних каналів становить понад 6 мільйонів. При приховуванні текстових файлів або документів, які займають незначний відсоток від загального обсягу, ймовірність того, що алгоритм Random LSB на другому проході обере той самий піксель, що й на першому, є статистично незначною.

Відмова від примусового оминання зайнятих пікселів значно підвищує стійкість до стегоаналізу. Якби система гарантовано залишала "дірки" в шумі другого шару в місцях, де записаний перший шар, це створило б кореляцію, яку міг би виявити атакуючий. Незалежне накладання шарів робить їх математично незв'язаними: наявність першого шару ніяк не впливає на розподіл другого, що

робить задачу доведення існування прихованої інформації (без знання пароля) обчислювально складною.

Блок-схема алгоритму послідовного формування багаторівневого контейнера наведена на рисунку 2.2.

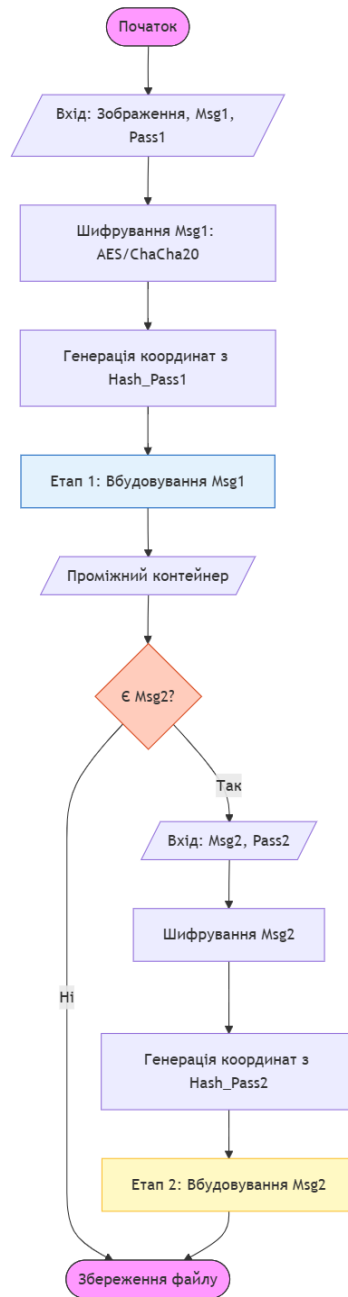


Рисунок 2.3 - Блок-схема алгоритму двоетапного вбудовування даних

2.3.Розробка модифікованого алгоритму вбудовування даних на основі псевдовипадкових послідовностей

Центральним елементом розроблюваної системи є алгоритм розподілу секретних даних усередині контейнера. Стандартні підходи, які використовуються у навчальних прикладах стеганографії, зазвичай реалізують послідовний запис бітів (Sequential LSB): дані записуються у пікселі зліва направо, рядок за рядком, починаючи з верхнього лівого кута зображення.

Проведений аналіз показав, що такий підхід є критично вразливим. При вбудовуванні великого повідомлення верхня частина зображення набуває змінених статистичних характеристик (підвищується ентропія молодших бітів), тоді як нижня частина залишається незмінною. Ця аномалія, відома як "межа вбудовування", легко виявляється навіть найпростішими стегоаналітичними фільтрами.

Для усунення цього недоліку в даній роботі було спроектовано та реалізовано модифікований алгоритм із псевдовипадковим розподілом інтервалів (Random LSB). Основна ідея методу полягає в тому, щоб рівномірно "розсіяти" біти повідомлення по всій площині зображення, імітуючи природний тепловий шум матриці цифрової камери.

Ключовою вимогою до алгоритму є можливість точного відновлення послідовності координат на стороні отримувача. Це означає, що генератор випадкових чисел (ГВЧ) повинен бути не істинно випадковим, а детермінованим, тобто видавати одну й ту саму послідовність чисел при однаковому початковому значенні (seed).

У розробленій системі роль "зерна" (seed) для генератора відіграє пароль користувача. Однак, пряме використання пароля є небезпечним через його можливу малу довжину та передбачуваність. Тому реалізовано наступний ланцюжок перетворень:

1. Пароль користувача подається на вхід криптографічної хеш-функції SHA-256.
2. Отриманий 256-бітний хеш перетворюється на велике ціле число.

3. Це число використовується для ініціалізації стану генератора псевдовипадкових чисел.

Такий підхід гарантує лавинний ефект: зміна навіть одного символу в паролі призводить до повної зміни хешу, а отже — до генерації абсолютно іншої послідовності координат. Це робить спроби підбору пароля шляхом часткового відновлення зображення неможливими.

При розробці алгоритму Random LSB існує дві стратегії вибору пікселів: повне перемішування списку всіх координат (shuffling) або динамічний випадковий відбір (random pick). Метод перемішування є простим, але потребує значної кількості оперативної пам'яті для зберігання списку індексів зображень високої роздільної здатності.

У даній роботі було реалізовано оптимізований підхід динамічного відбору з контролем унікальності. Генератор псевдовипадкових чисел, ініціалізований хешем пароля, видає випадковий індекс пікселя. Система перевіряє цей індекс на наявність у спеціальній структурі даних — множині (set) використаних координат. Якщо індекс новий, він додається до множини і використовується для запису; якщо ж він уже був використаний (колізія), генератор видає наступне число, доки не буде знайдено вільний піксель.

Реалізація описаного алгоритму виконана мовою Python у класі StegoHandler. Фрагмент програмного коду, що відповідає за генерацію унікальної послідовності координат на основі пароля за допомогою хеш-функції SHA-256 та множини для відстеження зайнятих позицій, наведено на рисунку 2.3.

```
def _get_indices_generator(self, width, height, password): 2 usages

    total_pixels = width * height * 3 # 3 канали (R, G, B)

    # 1. Створюємо seed з пароля
    hash_obj = hashlib.sha256(password.encode('utf-8'))
    seed_val = int(hash_obj.hexdigest(), 16)

    # 2. Ініціалізуємо генератор
    rng = random.Random(seed_val)

    # 3. Генеруємо координати.
    used_indices = set()

    while True:
        # Вибираємо випадковий індекс
        idx = rng.randint(0, total_pixels - 1)

        if idx not in used_indices:
            used_indices.add(idx)
            yield idx
```

Рисунок 2.4 - Фрагмент програмного коду генерації координат вбудовування

Як видно з наведеного коду, замість створення повного списку всіх індексів, що вимагало б значних обсягів оперативної пам'яті для великих зображень, система використовує підхід динамічної генерації. Після ініціалізації генератора хешем пароля, алгоритм вибирає випадкові координати «на льоту» та перевіряє їх на унікальність за допомогою хеш-таблиці (множини `used_indices`).

Така реалізація забезпечує високу ефективність: перевірка на входження у множину виконується за константний час, а відсутність необхідності зберігати масив із мільйонів індексів дозволяє значно зменшити споживання пам'яті. Це робить алгоритм придатним для роботи із зображеннями високої роздільної здатності (Full HD, 4K) у реальному часі.

Окрім графічних контейнерів, розроблена система підтримує стеганографічне перетворення звукових файлів формату WAV. На відміну від зображень, де інформація приховується у просторовій області (координати x , y), аудіофайл представляє собою сигнал у часовій області.

Для програмної реалізації (клас `AudioStego`) було обрано формат WAV (Waveform Audio File Format) з кодуванням PCM (Pulse-Code Modulation). Цей формат зберігає аудіодані без стиснення, що є критично важливим для

стеганографії: будь-яке стиснення з втратами (як у MP3 або AAC) неминуче знищить інформацію, записану в молодші біти.

Аудіопотік у форматі PCM представляє собою послідовність цифрових значень амплітуди (семплів). Для стандартної якості CD-Audio (частота дискретизації 44.1 кГц, розрядність 16 біт, стерео) кожна секунда запису містить 176 400 байт даних. Така висока щільність дозволяє використовувати метод LSB (Least Significant Bit) без застосування складних схем розкиду.

Алгоритм вбудовування працює за наступною логікою:

1. Зчитування контейнера: за допомогою бібліотеки wave аудіофайл розбирається на окремі фрейми, які конвертуються у плоский масив байтів (bytearray).

2. Підготовка повідомлення: секретні дані доповнюються 4-байтовим заголовком, що містить довжину повідомлення. Це дозволяє при вилученні точно визначити межі корисного навантаження.

3. Модифікація семплів: виконується послідовний прохід по масиву байтів аудіо.

Використання методу LSB у 16-бітних аудіофайлах базується на особливостях людського слуху. Зміна молодшого біта в семплі змінює амплітуду сигналу лише на $1/65536$ частину динамічного діапазону. Така мікроскопічна зміна сприймається слуховим апаратом людини як незначний білий шум, що лежить нижче порогу чутності (шум квантування). Це забезпечує високу непомітність передачі даних навіть при послідовному заповненні контейнера.

Фрагмент програмного коду, що показує перетворення байтів у біти та цикл перезапису LSB, наведено на рисунку 2.4.

```

# Перетворюємо дані в біти
bits = []
for byte in full_data:
    for i in range(8):
        bits.append((byte >> (7 - i)) & 1)

# Перевірка ємності
if len(bits) > len(frame_bytes):
    raise ValueError(f"Аудіофайл замалий! Потрібно {len(bits)} байтів, є {len(frame_bytes)}.")

# 3. Вбудовування (LSB)
# Ми змінюємо останній біт кожного байта аудіоданих
for i, bit in enumerate(bits):
    # frame_bytes[i] & 254 -> занулюємо останній біт (11111110)
    # | bit -> вставляємо наш біт (0 або 1)
    frame_bytes[i] = (frame_bytes[i] & 254) | bit

```

Рисунок 2.5 - Програмна реалізація методу LSB для WAV-файлів

Висновки до другого розділу: у другому розділі здійснено проєктування архітектури програмного комплексу для прихованої передачі даних. Обґрунтовано вибір технологічного стека, що включає мову програмування Python, бібліотеку графічного інтерфейсу PySide6 та криптографічний пакет cryptography.

Розроблено детальну логічну структуру системи, яка базується на концепції подвійного дна (Double Steganography), що дозволяє реалізувати механізм правдоподібного заперечення шляхом розділення контейнера на області для відкритого та прихованого повідомлень. Для забезпечення стійкості до статистичного стегоаналізу спроектовано модифікований алгоритм Random LSB, який використовує криптографічне хешування пароля для детермінованої генерації псевдовипадкової послідовності координат вбудовування.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО КОМПЛЕКСУ ПРИХОВАНОЇ ПЕРЕДАЧІ ДАНИХ

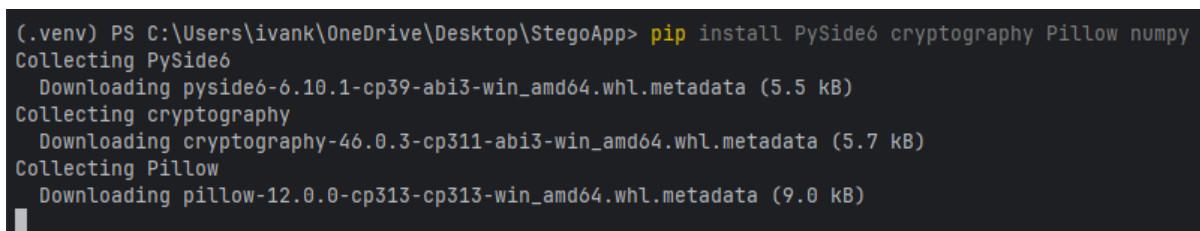
3.1. Програмна реалізація модулів шифрування та стеганографічного приховування

Розробка програмного комплексу виконувалася мовою Python 3.10 з використанням об'єктно-орієнтованого підходу. Архітектура додатку побудована за модульним принципом, де логіка криптографічних перетворень, стеганографічної обробки та графічний інтерфейс відокремлені в незалежні компоненти. Це забезпечує гнучкість системи та можливість її подальшого масштабування.

Першим етапом реалізації було розгортання віртуального середовища розробки. Для забезпечення роботи алгоритмів AES, ChaCha20 та LSB було встановлено ряд зовнішніх бібліотек. Ключовими залежностями проєкту є:

1. PySide6 - для реалізації графічного інтерфейсу користувача (GUI).
2. cryptography - для забезпечення криптографічних функцій та генерації ключів.
3. Pillow (PIL) - для маніпуляції з пікселями зображень.
4. numpy - для ефективної роботи з масивами даних.

Процес інсталяції необхідних пакетів через менеджер рір та налаштування інтерпретатора наведено на рисунку 3.1.



```
(.venv) PS C:\Users\ivank\OneDrive\Desktop\StegoApp> pip install PySide6 cryptography Pillow numpy
Collecting PySide6
  Downloading pyside6-6.10.1-cp39-abi3-win_amd64.whl.metadata (5.5 kB)
Collecting cryptography
  Downloading cryptography-46.0.3-cp311-abi3-win_amd64.whl.metadata (5.7 kB)
Collecting Pillow
  Downloading pillow-12.0.0-cp313-cp313-win_amd64.whl.metadata (9.0 kB)
```

Рисунок 3.1 - Встановлення залежностей проєкту у віртуальному середовищі

Для зручності навігації та дотримання стандартів чистого коду було спроектовано ієрархічну структуру файлів. Головною точкою входу є файл main.py, який відповідає за ініціалізацію додатку та запуск головного циклу обробки подій. Основні модулі системи розподілено по директоріях:

- `app/core/` — містить файли з реалізацією алгоритмів: `crypto.py` (шифрування), `stego.py` (зображення), `stego_audio.py` (звук).
- `app/gui/` — містить файл `interface.py`, який описує візуальні компоненти та їх взаємодію з користувачем.
- `assets/` — ресурси додатку (іконки, стилі).

Така структура дозволяє чітко розмежувати зону відповідальності кожного файлу. Організацію файлової системи проєкту продемонстровано на рисунку 3.2.

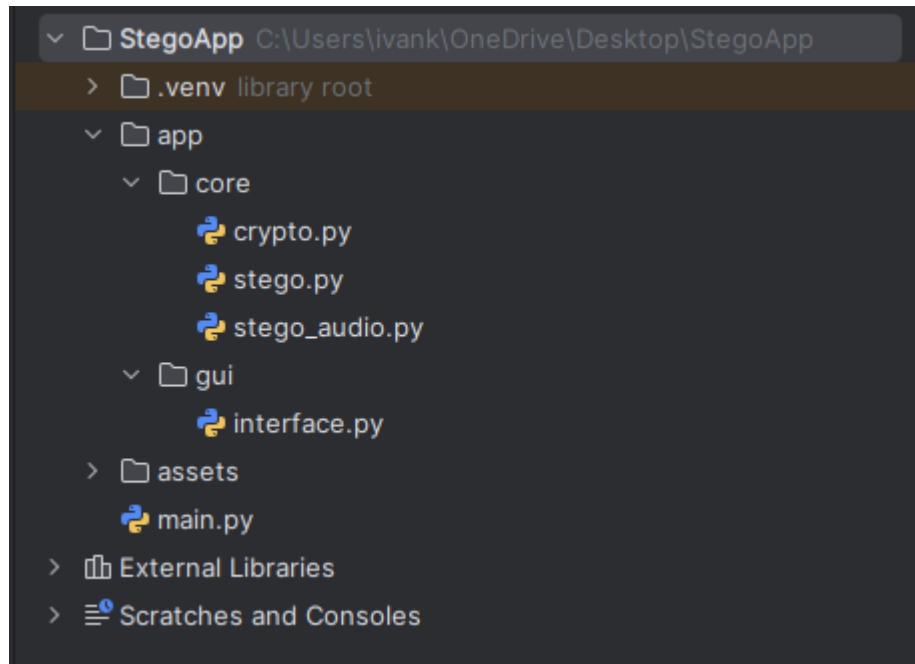


Рисунок 3.2 - Файлова структура розробленого програмного комплексу

Ядром системи безпеки є клас `CryptoHandler`, реалізований у файлі `crypto.py`. Він забезпечує уніфікований інтерфейс для шифрування та дешифрування даних. У конструкторі класу ініціалізується бекенд бібліотеки `cryptography`. Реалізовано метод `_derive_key`, який генерує криптографічно стійкий ключ із пароля користувача з використанням алгоритму `PBKDF2HMAC` та випадкової солі (Salt) розміром 16 байт.

```

class CryptoHandler: 2 usages
    """
    Клас для обробки шифрування.
    Підтримує:
    1. AES-256 (CTR mode) - Класичний стандарт.
    2. ChaCha20-Poly1305 - Сучасний потоківий шифр (швидший на мобільних).
    """

    def __init__(self):
        self.backend = default_backend()
        self.salt_size = 16

    def _derive_key(self, password: str, salt: bytes, length=32) -> bytes: 2 usages
        """Генерує ключ із пароля"""
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=length,
            salt=salt,
            iterations=100000,
            backend=self.backend
        )
        return kdf.derive(password.encode('utf-8'))

```

Рисунок 3.3 - Реалізація генерації ключів у класі CryptoHandler

Метод encrypt підтримує два режими роботи: AES-256 (CTR) та ChaCha20-Poly1305. Важливою особливістю реалізації є попереднє стиснення даних алгоритмом zlib (рівень 9) перед шифруванням, що реалізовано наступним чином:

```
compressed_data = zlib.compress(data, level=9)
```

Це дозволяє зменшити обсяг даних та підвищити їх ентропію. Результатом роботи методу є бінарний пакет, що містить маркер алгоритму, сіль, вектор ініціалізації (IV або Nonce) та шифротекст.

```

def encrypt(self, data: bytes, password: str, algorithm="AES") -> bytes: 6 usages
    """
    # 1. Стиснення
    compressed_data = zlib.compress(data, level=9)

    # 2. Генерація солі
    salt = os.urandom(self.salt_size)
    key = self._derive_key(password, salt)

    if algorithm == "ChaCha20":
        # ChaCha20 вимагає Nonce (12 байт)
        nonce = os.urandom(12)
        cipher = ChaCha20Poly1305(key)
        ciphertext = cipher.encrypt(nonce, compressed_data, associated_data=None)

        # Маркер для розпізнавання при дешифруванні (1 байт = 2)
        algo_marker = b'\x02'
        return algo_marker + salt + nonce + ciphertext

    else:
        # AES-256 CTR (за замовчуванням)
        iv = os.urandom(16)
        cipher = Cipher(algorithms.AES(key), modes.CTR(iv), backend=self.backend)
        encryptor = cipher.encryptor()
        ciphertext = encryptor.update(compressed_data) + encryptor.finalize()

        # Маркер для AES (1 байт = 1)
        algo_marker = b'\x01'
        return algo_marker + salt + iv + ciphertext

```

Рисунок 3.4 - Програмний код методу шифрування даних

Логіка вбудовування даних у графічні контейнери винесена в клас StegoHandler (файл stego.py). Для роботи із зображеннями використовується бібліотека Pillow, яка дозволяє отримувати прямий доступ до матриці пікселів через об'єкт PixelAccess.

Ключовою особливістю реалізації є відмова від послідовного заповнення пікселів на користь псевдовипадкового розподілу. Для цього розроблено метод `_get_indices_generator`, який приймає розміри зображення та пароль користувача. Алгоритм працює наступним чином:

1. Пароль хешується (SHA-256), результат перетворюється на число (seed).
2. Ініціалізується генератор випадкових чисел `random.Random(seed_val)`.
3. У нескінченному циклі генеруються координати пікселів.

4. Використовується множина `used_indices = set()` для контролю унікальності координат і уникнення колізій (повторного запису в той самий піксель).

```
def _get_indices_generator(self, width, height, password): 2 usages
    """
    Генерує послідовність унікальних координат пікселів (індексів)
    на основі пароля. Це гарантує, що різні паролі дадуть різні пікселі.
    """
    total_pixels = width * height * 3 # 3 канали (R, G, B)

    # 1. Створюємо seed з пароля
    hash_obj = hashlib.sha256(password.encode('utf-8'))
    seed_val = int(hash_obj.hexdigest(), 16)

    # 2. Ініціалізуємо генератор
    rng = random.Random(seed_val)

    # 3. Генеруємо координати.
    # Для оптимізації ми не робимо список з мільйонів чисел,
    # а використовуємо set для перевірки унікальності.
    # Це працює швидко для текстових повідомлень.
    used_indices = set()

    while True:
        # Вибираємо випадковий індекс
        idx = rng.randint(0, total_pixels - 1)

        # Якщо ми вже писали в цей піксель - пропускаємо (шукаємо інший)
        if idx not in used_indices:
            used_indices.add(idx)
            yield idx
```

Рисунок 3.5 - Генератор псевдовипадкових координат на основі пароля

Сам процес вбудовування (метод `embed`) реалізує модифікацію молодшого значущого біта (LSB). Вхідні байти розкладаються на біти, після чого для кожного біта повідомлення обирається випадковий канал (R, G або B) випадкового пікселя, визначеного генератором. Для зміни значення каналу використовується бітова операція:

$$channels[channel] = (channels[channel] \& \sim 1) | bit_to_write$$

Це забезпечує зміну яскравості кольору не більше ніж на 1 одиницю, що є непомітним для людського ока.

```

def embed(self, image_path, data, output_path, password): 2 usages
    """
    Вбудовує дані в зображення, використовуючи пароль для вибору позицій.
    """
    try:
        img = Image.open(image_path)
        width, height = img.size
        pixels = img.load() # Матриця пікселів [x, y] = (r, g, b)

        # 1. Підготовка даних (Розмір + Самі дані)
        data_len = len(data)
        full_data = data_len.to_bytes(self.header_size, byteorder='big') + data

        # Перетворюємо байти в біти (0 та 1)
        bits = []
        for byte in full_data:
            for i in range(8):
                bits.append((byte >> (7 - i)) & 1)

        required_bits = len(bits)
        total_capacity = width * height * 3

        if required_bits > total_capacity:
            print("Помилка: Дані занадто великі для цього зображення!")
            return False

```

Рисунок 3.6 - Реалізація методу LSB для RGB-каналів

Робота зі звуковими даними реалізована в класі AudioStego (файл stego_audio.py). На відміну від зображень, де використовується бібліотека Pillow, для аудіо використано стандартний модуль wave. Це дозволяє працювати з сирими даними у форматі PCM (Pulse-Code Modulation) без втрати якості.

Метод embed зчитує всі фрейми аудіофайлу у масив байтів (bytearray). Оскільки аудіопотік має високу щільність даних, реалізовано лінійний алгоритм запису. Для кожного байта повідомлення виконується модифікація останнього біта відповідного байта аудіоданих. Перед записом здійснюється перевірка ємності контейнера: якщо розмір повідомлення перевищує кількість доступних семплів, генерується виключення ValueError.

```

def embed(self, audio_path: str, data: bytes, output_path: str) -> bool:
    """
    Вбудовує байти (data) у .wav файл.
    """
    try:
        # 1. Відкриваємо аудіофайл
        song = wave.open(audio_path, mode='rb')

        # Читаємо параметри (кількість кадрів, частота)
        frame_bytes = bytearray(list(song.readframes(song.getnframes())))
        params = song.getparams() # Зберігаємо параметри, щоб відтворити файл потім
        song.close()

        # 2. Підготовка даних
        data_len = len(data)
        # Додаємо розмір даних на початок
        full_data = struct.pack( 'f>I', *v: data_len) + data

        # Перетворюємо дані в біти
        bits = []
        for byte in full_data:
            for i in range(8):
                bits.append((byte >> (7 - i)) & 1)

        # Перевірка ємності
        if len(bits) > len(frame_bytes):
            raise ValueError(f"Аудіофайл замалий! Потрібно {len(bits)} байтів, є {len(frame_bytes)}.")
    
```

Рисунок 3.7 - Алгоритм вбудовування даних у WAV-файл

Взаємодія користувача з програмним комплексом реалізована в класі MainWindow (файл interface.py), який успадковує функціонал QMainWindow з бібліотеки PySide6. Інтерфейс спроектовано за принципом "Single Window" з використанням бокового меню навігації та віджета QStackedWidget для перемикання між функціональними блоками:

1. Фото-стеганографія.
2. Аудіо-стеганографія.
3. Крипто-лабораторія.

Для оформлення додатку використано каскадні таблиці стилів (QSS), що дозволило реалізувати темну тему (Dark Theme), комфортну для роботи в умовах слабкого освітлення. Стилізація елементів (кнопок, полів вводу) визначена глобально в конструкторі класу.

```

class MainWindow(QMainWindow): 2 usages
    def __init__(self):
        super().__init__()
        self.setWindowTitle("StegoCrypto App - Master Diploma")
        self.resize(950, 680)

        # Ініціалізація бекенду
        self.crypto = CryptoHandler()
        self.stego = StegoHandler()
        self.audio_stego = AudioStego()

        # Змінні
        self.source_image_path = None
        self.stego_image_path = None
        self.source_audio_path = None
        self.stego_audio_path = None

        # --- ЗАГАЛЬНИЙ СТИЛЬ (DARK THEME) ---
        self.setStyleSheet("""
            QMainWindow, QWidget {
                background-color: #2b2b2b;
                color: #e0e0e0;
                font-family: 'Segoe UI', sans-serif;
                font-size: 14px;
            }
        """)

```

Рисунок 3.8 - Ініціалізація головного вікна та налаштування стилів QSS

Логіка обробки подій реалізована через механізм сигналів і слотів. Наприклад, натискання кнопки "Зашифрувати" викликає метод-обробник (`process_embed_photo`), який збирає дані з полів вводу, виконує валідацію та передає керування бекенд-класам `CryptoHandler` та `StegoHandler`. Асинхронне оновлення інтерфейсу (прогрес-бари, статус-повідомлення) реалізовано через виклики `QApplication.processEvents()`, що запобігає "зависанню" вікна під час тривалих операцій.

Результатом роботи програмного коду, описаного вище, є побудова інтерактивного віконного інтерфейсу. Головне вікно додатку спроектовано за принципом «єдиного робочого простору» (Single Page Application layout), де навігація між функціональними модулями здійснюється через бічне меню без відкриття додаткових вікон.

При запуску програми (main.py) ініціалізується клас MainWindow. Інтерфейс виконано у темній колірній гамі (Dark Theme), що реалізовано через глобальні стилі QSS. Кольорова палітра підібрана для зменшення зорового навантаження при тривалій роботі: основний фон — #2b2b2b, активні елементи — #365880 (синій спектр).

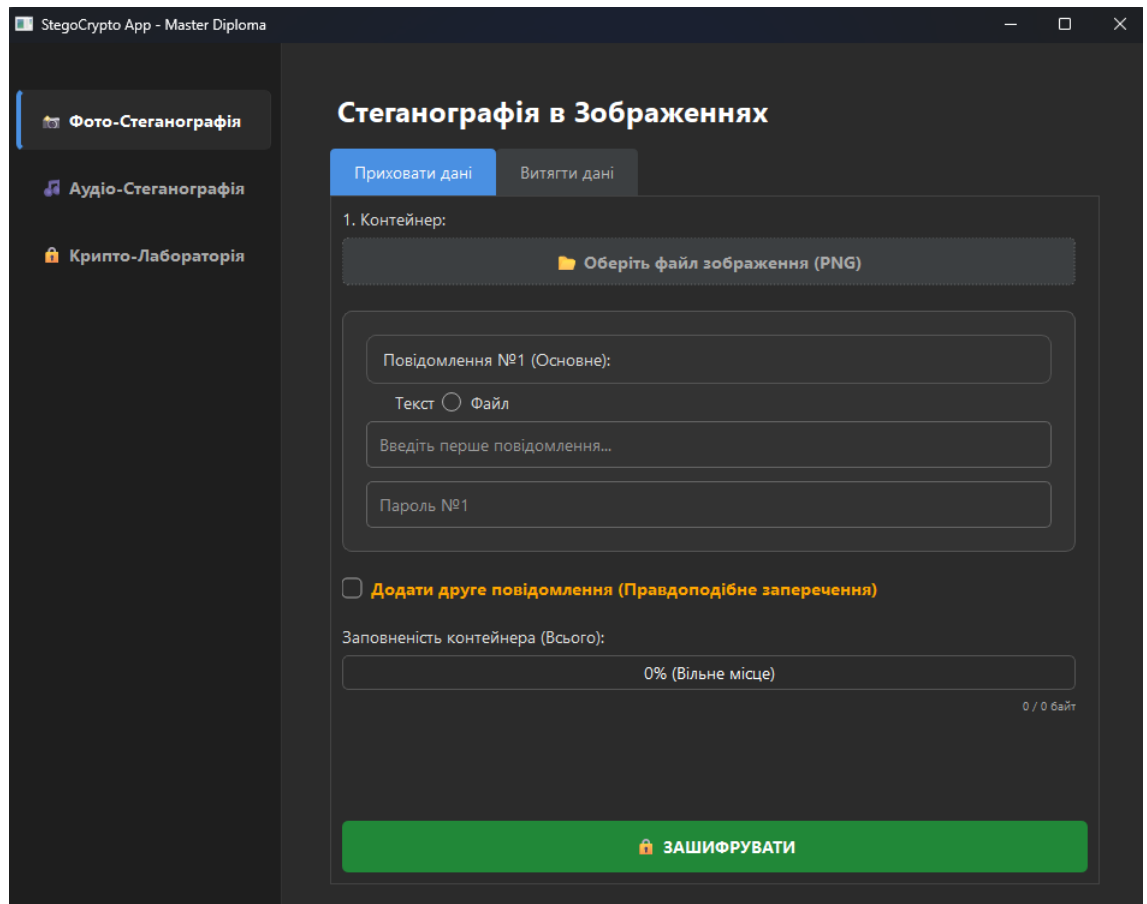


Рисунок 3.9 - Головне вікно програми: модуль фото-стеганографії

Вкладка «Фото-Стеганографія» реалізує інтерфейс для взаємодії з класом StegoHandler. Робоча область розділена на дві підвкладки: «Приховати дані» та «Витягти дані». Форма вбудовування містить наступні елементи керування:

1. Вибір контейнера: кнопка завантаження файлу (обробник `select_source_image`) з фільтром для форматів PNG. Після вибору відображається назва файлу та з'являється кнопка очистки.

2. Тип повідомлення: перемикач (Radio Button), що дозволяє обрати режим введення: прямий текст або завантаження секретного файлу.

3. Поля вводу: текстові поля для секретного повідомлення та пароля захисту. Пароль приховується символами заміни (EchoMode Password) для безпеки.

4. Індикатор ємності: динамічний прогрес-бар, який у реальному часі розраховує доступний простір у зображенні на основі його роздільної здатності та показує відсоток заповнення.

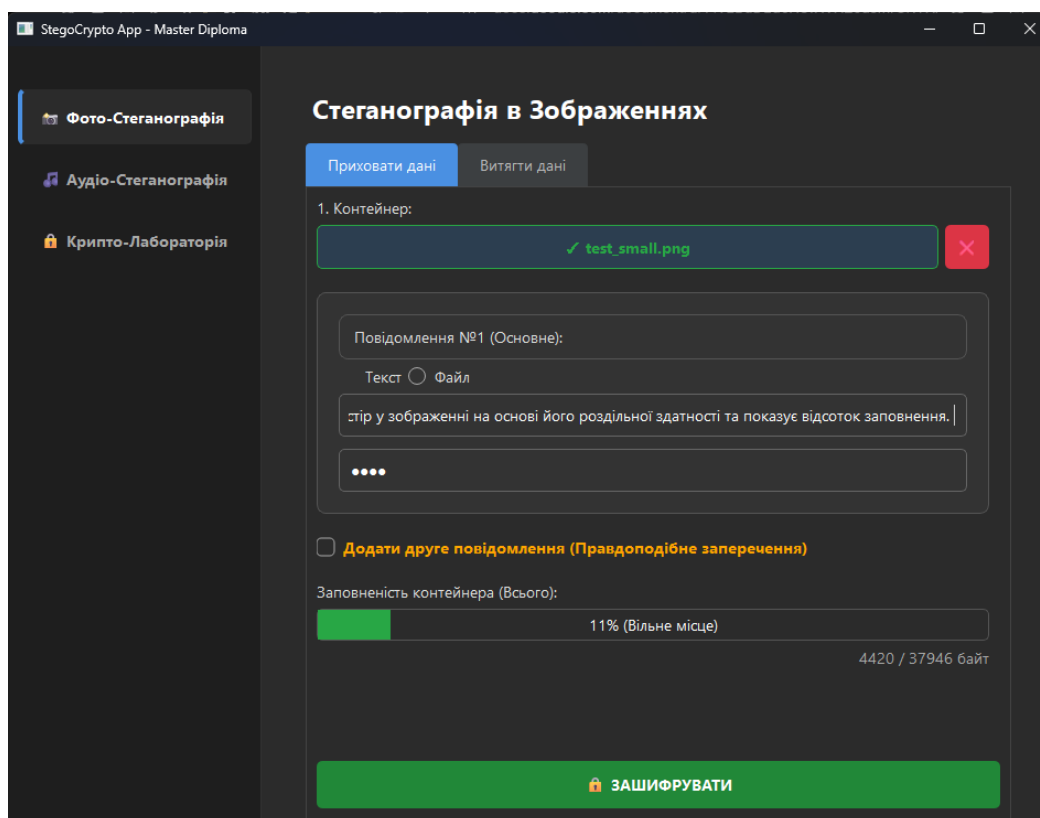


Рисунок 3.10 - Інтерфейс підготовки даних до приховування в зображенні

Окремим елементом інтерфейсу є блок «Правдоподібне заперечення». При активації чекбоксу «Додати друге повідомлення», інтерфейс динамічно розширюється, додаючи поля для введення другого тексту та другого пароля. Це візуалізує архітектурну особливість двоетапного вбудовування, описану в Розділі 2.

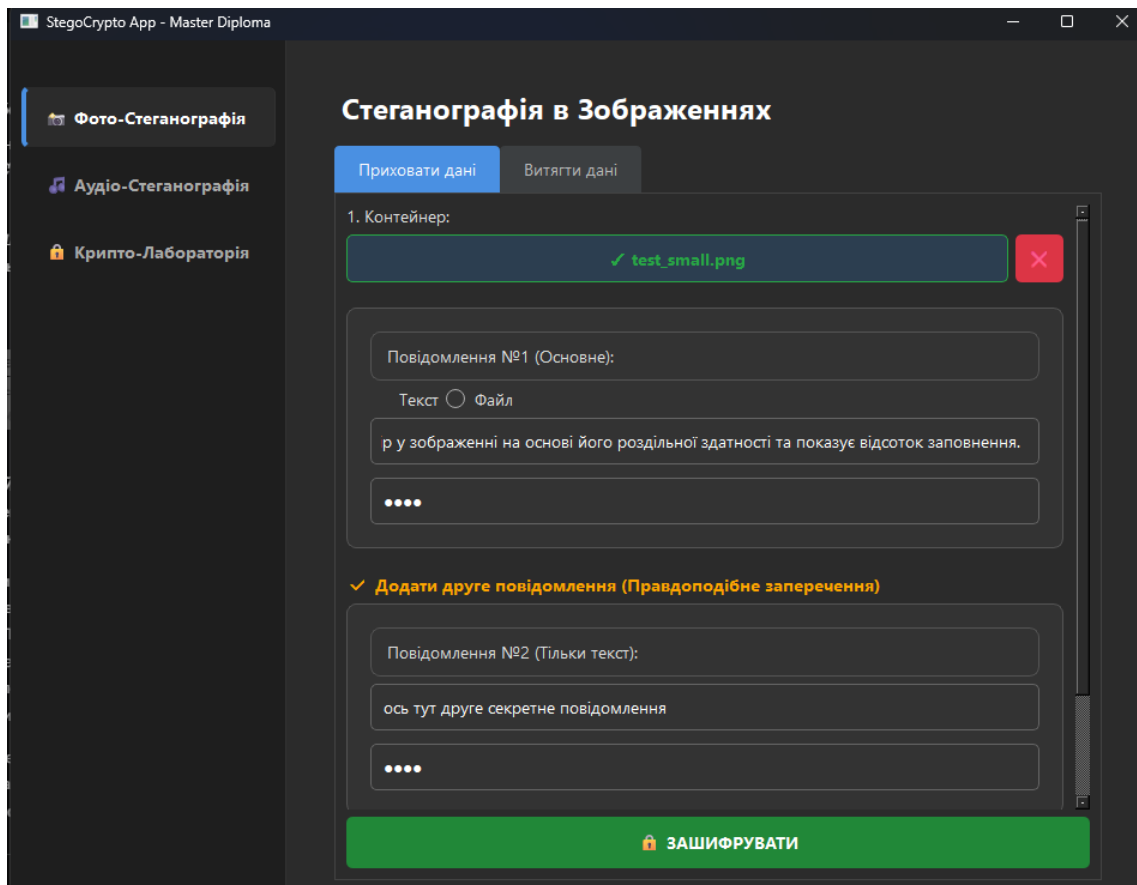


Рисунок 3.11 - Активація режиму подвійного дна в інтерфейсі

Для роботи зі звуковими даними розроблено окрему вкладку, яка взаємодіє з класом AudioStego. Інтерфейс адаптовано під специфіку аудіофайлів: фільтр вибору файлів налаштовано виключно на формат .wav. Логіка інтерфейсу передбачає сувору валідацію: кнопка «Зашифрувати в аудіо» залишається неактивною (Disabled) до моменту, поки користувач не обере файл-носій, не введе текст повідомлення та пароль. Це запобігає виникненню помилок виконання на етапі обробки даних.

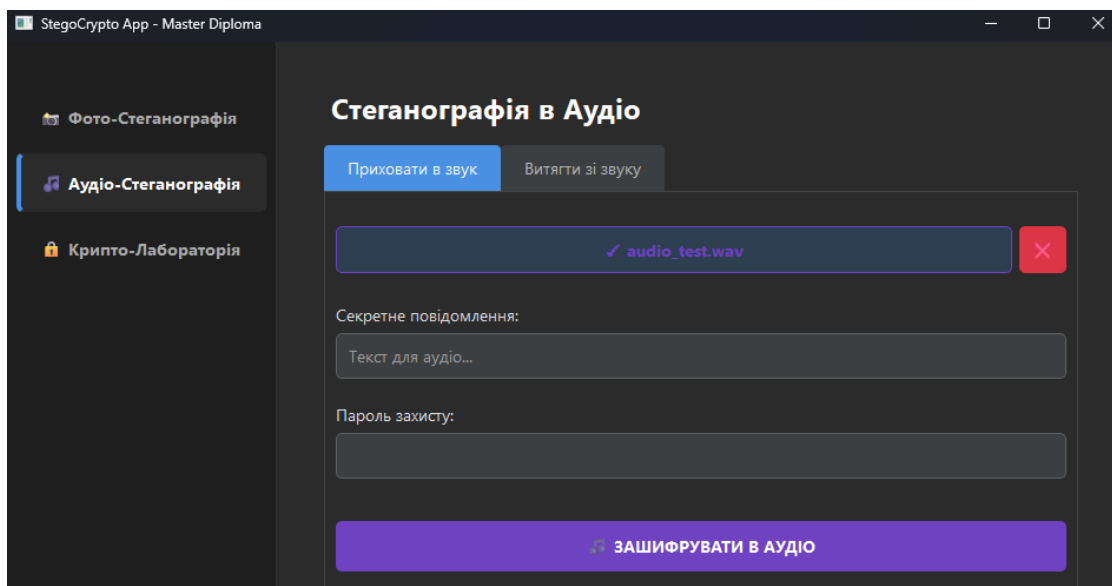


Рисунок 3.12 - Графічний інтерфейс модуля аудіо-стеганографії

Третій функціональний блок — «Крипто-Лабораторія» — надає користувачеві прямий доступ до функцій класу CryptoHandler. Цей модуль дозволяє виконувати шифрування тексту без стеганографії, що корисно для налагодження та перевірки стійкості ключів. Унікальним елементом цього інтерфейсу є випадаючий список (ComboBox) для вибору алгоритму (AES-256 або ChaCha20) та кнопка запуску бенчмарку, яка ініціює навантажувальне тестування системи.

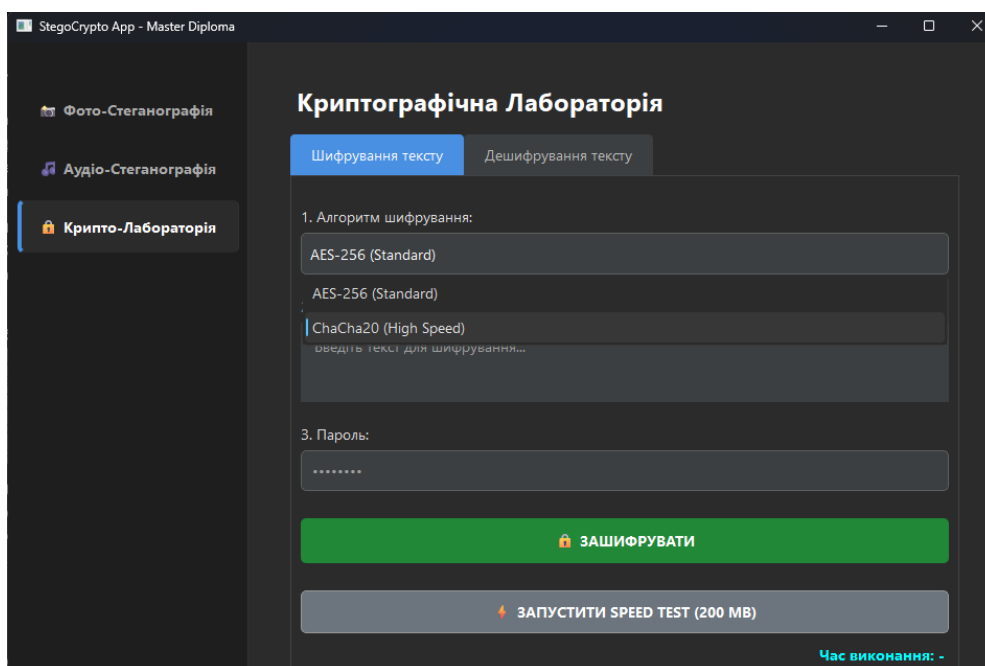


Рисунок 3.13 - Інтерфейс модуля досліджень криптографічних алгоритмів

3.2. Порівняльний аналіз ефективності криптографічних алгоритмів AES-256 та ChaCha20

Вибір алгоритму шифрування є критичним етапом проєктування системи, оскільки він безпосередньо впливає на продуктивність (Performance) та зручність використання (UX). У рамках дослідження було проведено серію експериментів для визначення оптимального шифру для десктопної версії додатку.

Для отримання об'єктивних даних було розроблено модуль автоматизованого бенчмаркінгу (інтегрований у вкладку «Крипто-Лабораторія»). Тестування проводилося за наступним протоколом:

1. Ізоляція середовища: перед запуском тесту було мінімізовано кількість фонових процесів ОС, що могли б вплинути на час доступу до CPU.

2. Підготовка даних: генерація в оперативній пам'яті (RAM) масиву випадкових байтів (`os.urandom`) розміром 200 МБ. Використання випадкових даних виключає можливість впливу кешування або оптимізації стиснення.

3. Інструментарій вимірювання: використано системний таймер `time.perf_counter()`, який забезпечує наносекундну точність вимірювань і не залежить від зміни системного часу.

4. Серійність: тест виконувався у 5 ітерацій, у залік брався середній результат для нівелювання випадкових сплесків навантаження процесора.

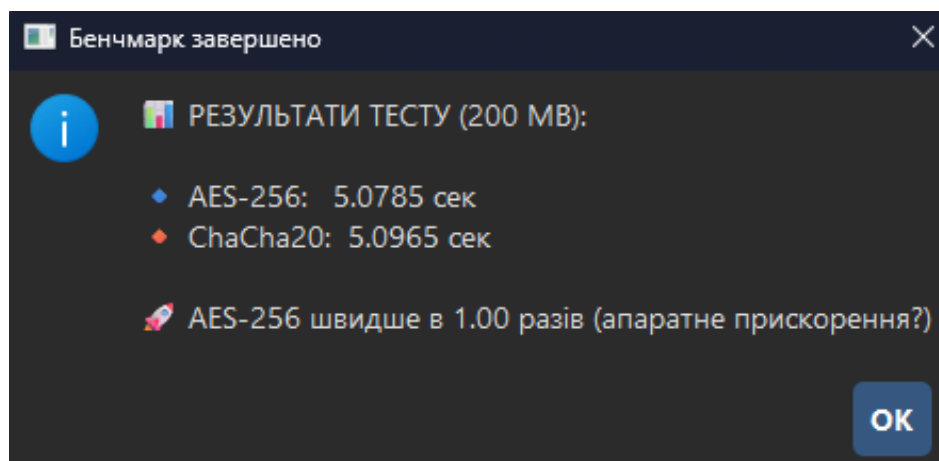


Рисунок 3.14 - Інтерфейс програми з результатами тестування на 200МБ

Таблиця 3.1

Час виконання шифрування для об'єму 200МБ

<i>Алгоритм</i>	<i>Режим роботи</i>	<i>Час виконання (с)</i>	<i>Відносна різниця</i>
AES-256	CTR (Counter Mode)	5.0785с	100%
ChaCha20	Poly1305 (AEAD)	5.0965с	99.64%

Алгоритм AES-256 впорався із завданням швидше. Це пояснюється тим, що хоча ChaCha20 є оптимізованим для програмного виконання (Software-friendly), AES отримує значну перевагу завдяки апаратній акселерації на рівні процесора.

Важливим фактором, що визначив перемогу AES у даному тесті, є архітектура сучасних процесорів x86-64. Більшість CPU від Intel та AMD підтримують розширення набору команд AES-NI (Advanced Encryption Standard New Instructions). Бібліотека cryptography (що базується на OpenSSL) автоматично детектує наявність цих інструкцій і делегує обчислення раундів шифрування (SubBytes, ShiftRows, MixColumns) спеціалізованим блокам процесора.

На основі отриманих даних про швидкість (Speed), було складено прогноз часу очікування користувача для файлів різних обсягів (Таблиця 3.2).

Таблиця 3.2

Прогноз часу обробки файлів

<i>Тип файлу</i>	<i>Розмір</i>	<i>Час очікування (AES)</i>	<i>Час очікування (ChaCha20)</i>
Фото	25мб	< 0.1 с	< 0.1 с
Аудіо	600мб	15.2355 с	15.2895 с
Відео	2Гб	50.785 с	50.965 с
Архів даних	10Гб	253.925 с	254.825 с

Експериментально доведено, що на цільовій апаратній платформі алгоритм AES-256 за ChaCha20 завдяки апаратній підтримці. Це дозволяє рекомендувати AES як основний стандарт для розробленого програмного комплексу. Однак, ChaCha20 залишається важливим компонентом системи як резервний алгоритм (fallback) для забезпечення кросплатформеної сумісності з ARM-архітектурою (мобільні пристрої), де підтримка AES-NI може бути відсутньою.

Достовірність отриманих результатів забезпечується коректною програмною реалізацією модуля бенчмаркінгу. Логіка тестування інтегрована безпосередньо в клас графічного інтерфейсу MainWindow (файл interface.py), що дозволяє користувачеві запускати перевірку в реальному часі без використання сторонніх утиліт.

Алгоритм тестування побудовано на використанні криптографічно стійкого генератора псевдовипадкових чисел операційної системи. Це критично важливо для об'єктивності експерименту: якби використовувалися нулі або повторювані патерни, процесор міг би оптимізувати процес, і результати були б викривлені.

Для генерації тестового навантаження використовується метод `os.urandom()`. Фрагмент коду, що відповідає за підготовку даних обсягом 200 МБ, наведено на рисунку 3.15.

```
def run_benchmark(self): 1usage
    """
    Генерує 200 МБ випадкових даних і порівнює швидкість AES vs ChaCha20.
    """
    reply = QMessageBox.question(self, "Бенчмарк",
                                text="Це створить 200 МБ даних у пам'яті і протестує обидва алгоритми.\n"
                                     "Програма може 'завмерти' на 5-10 секунд.\nПродовжити?",
                                QMessageBox.Yes | QMessageBox.No)

    if reply == QMessageBox.No:
        return

    try:
        self.status_label_crypto.setText("Генерація даних (200 МБ)...")
        self.lbl_execution_time.setText("Тестування...")
        QApplication.processEvents()

        # 1. Генерація сміттєвих даних (200 МБ)
        data_size = 200 * 1024 * 1024 # <--- TUT ТЕПЕР 200 МБ
        test_data = os.urandom(data_size)
        password = "BenchmarkPassword123"
```

Рисунок 3.15 - Генерація тестового масиву даних (200мб) у методі бенчмарку

Вимірювання часових інтервалів реалізовано через функцію `time.perf_counter()`. На відміну від застарілого `time.time()`, цей метод звертається до високоточного апаратного лічильника процесора, що дозволяє фіксувати різницю в часі з точністю до наносекунд.

Процес тестування виконується синхронно для забезпечення максимального пріоритету процесу. Спочатку ініціалізується об'єкт шифрування, потім запускається таймер, виконується шифрування, і таймер зупиняється. Отриманий результат (`delta time`) автоматично форматується та виводиться у текстове поле інтерфейсу (`self.ui.textEdit_results`).

```
def run_benchmark(self): 1 usage

    # 2. Тест AES-256
    self.status_label_crypto.setText("Тестування AES-256...")
    QApplication.processEvents()

    start_aes = time.perf_counter()
    self.crypto.encrypt(test_data, password, algorithm="AES")
    end_aes = time.perf_counter()
    time_aes = end_aes - start_aes

    # 3. Тест ChaCha20
    self.status_label_crypto.setText("Тестування ChaCha20...")
    QApplication.processEvents()

    start_cha = time.perf_counter()
    self.crypto.encrypt(test_data, password, algorithm="ChaCha20")
    end_cha = time.perf_counter()
    time_cha = end_cha - start_cha
```

Рисунок 3.16 - Реалізація вимірювання часу виконання криптографічних операцій

Такий підхід до реалізації гарантує, що вимірюється виключно "чистий" час роботи алгоритмів AES та ChaCha20, без врахування витрат на малювання графічного інтерфейсу або дискові операції, оскільки всі дані знаходяться в оперативній пам'яті (RAM).

3.3. Тестування стеганографічної стійкості та дослідження ємності мультимедійних контейнерів

Ключовим критерієм якості стеганографічної системи є неможливість візуального або аудіального виявлення факту наявності прихованого повідомлення. У цьому підрозділі наведено результати тестування розробленого програмного комплексу за критеріями візуальної непомітності (Transparency) та функціональної стійкості алгоритму правдоподібного заперечення.

Для перевірки коректності роботи алгоритму LSB (Least Significant Bit) було проведено експеримент із вбудовування текстового масиву даних у зображення формату PNG.

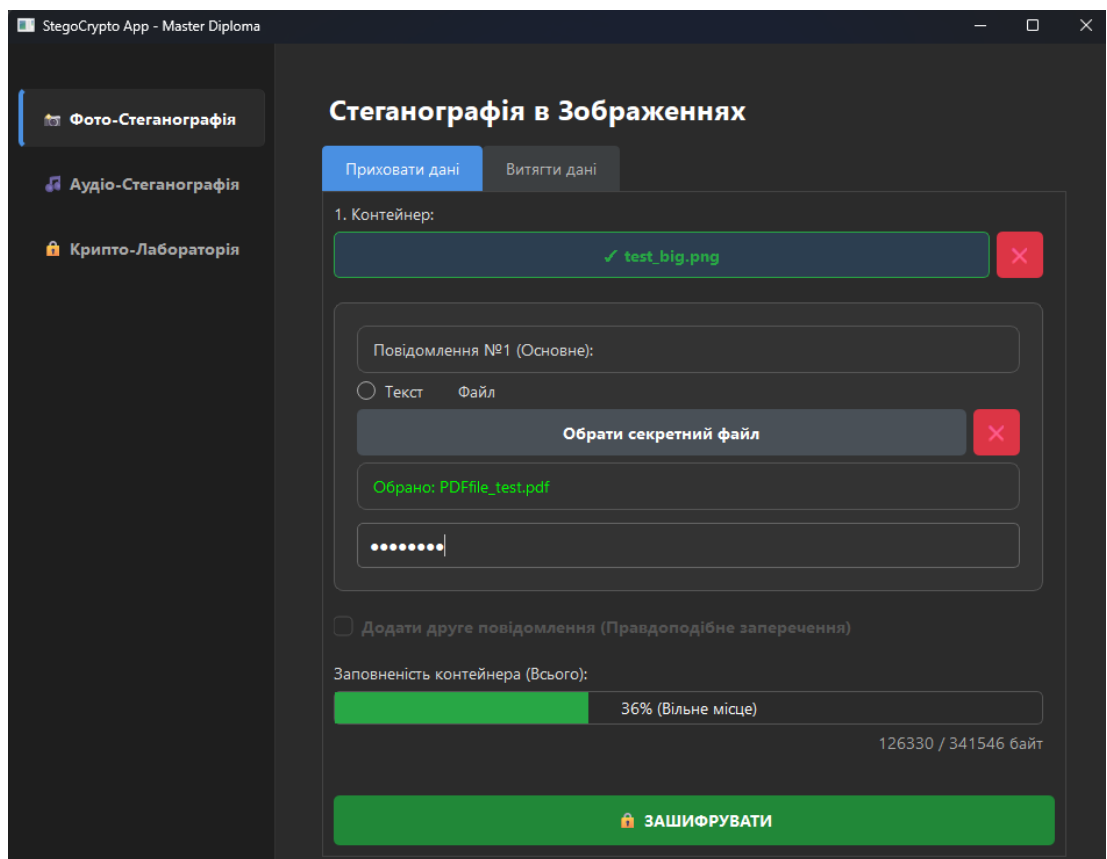


Рисунок 3.17 - Інтерфейс шифрування файлу в зображення

Параметри експерименту:

- Контейнер: зображення test_big.png розміром 313КБ;
- Корисне навантаження: файл розміром 124КБ;
- Коефіцієнт заповнення: 36% від загальної ємності контейнера.

На рисунку 3.18 наведено порівняння оригінального зображення (ліворуч) та зображення з прихованими даними (праворуч).

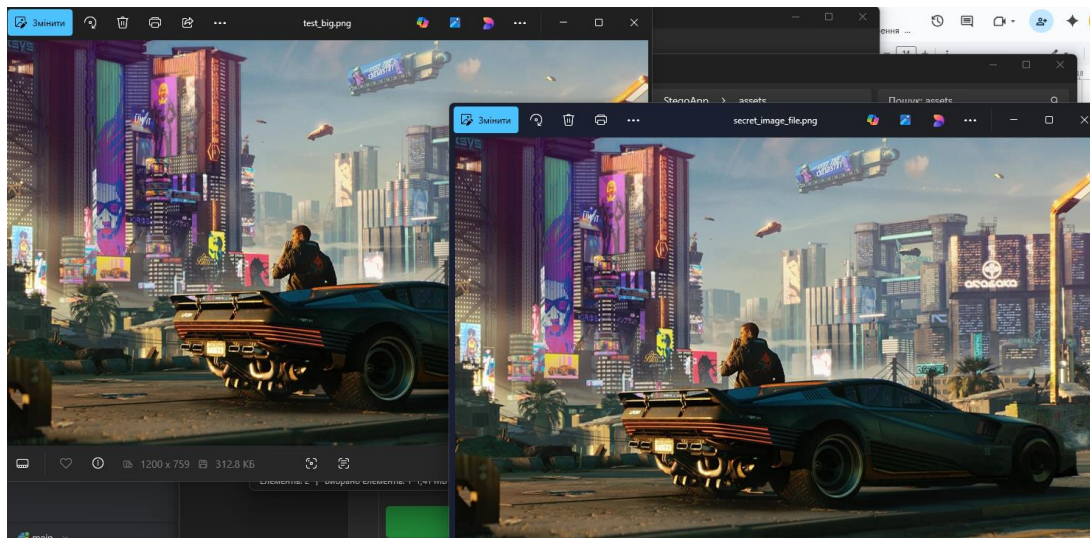


Рисунок 3.18 - Візуальне порівняння: оригінал та стегоконтейнер

Як видно з наведених зображень, людське око не здатне розрізнити зміни, внесені алгоритмом. Це пояснюється тим, що модифікація молодшого біта змінює значення кольору (наприклад, червоного каналу R) лише на одиницю (наприклад, з 200 на 201), що лежить за межами порогу кольоросприйняття зорової системи людини. Артефакти, шуми або спотворення кольорової гами відсутні.

У ході експерименту було зафіксовано збільшення розміру файлу-контейнера після вбудовування даних (з 312 КБ до 1.41 МБ). Це явище є очікуваним і пояснюється двома факторами:

1. Вимога цілісності (Lossless Saving): для збереження прихованих бітів вихідний файл зберігається у форматі PNG без стиснення з втратами, що вимагає більшого обсягу пам'яті порівняно з оригіналом, який міг бути попередньо оптимізований.

2. Зростання інформаційної ентропії: вбудовування зашифрованих даних (що мають статистичний розподіл, близький до рівномірного шуму) у площину LSB руйнує статистичні закономірності "чистого" зображення. Це знижує ефективність алгоритму стиснення DEFLATE, який використовується

стандартом PNG, оскільки в молодших бітах зникають повторювані послідовності.

Однією з унікальних особливостей розробленої системи є реалізація концепції "Подвійного дна". Ця функція дозволяє приховати в одному контейнері два незалежні повідомлення, доступ до яких здійснюється за різними паролями.

Сценарій тестування передбачав наступні кроки:

1. Вбудовування:

- Повідомлення №1 (фальшиве): "Text for test", пароль - guest123
- Повідомлення №2 (справжнє): "TOP SECRET", пароль - admin456

2. Процес запису: програма використовує два різні генератори псевдовипадкових чисел (PRNG), ініціалізовані хешами від різних паролів. Це гарантує, що повідомлення будуть записані у різні пікселі зображення з мінімальною ймовірністю колізії.

Інтерфейс програми на етапі підготовки подвійного вбудовування зображено на рисунку 3.19.

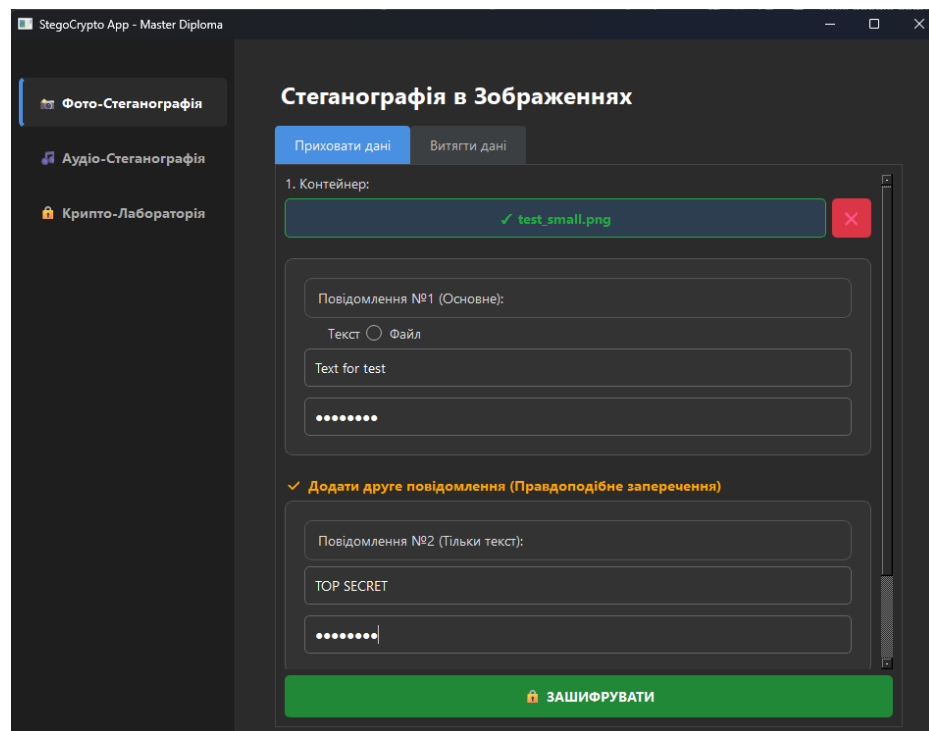


Рисунок 3.19 - Налаштування параметрів для подвійного вбудовування даних

3. Перевірка вилучення (Етап 1): при введенні першого пароля (guest123) система коректно ідентифікує та дешифрує лише перше повідомлення. Це імітує ситуацію, коли користувач змушений видати пароль під тиском — зловмисник отримає правдоподібну інформацію і не дізнається про існування другого шару.

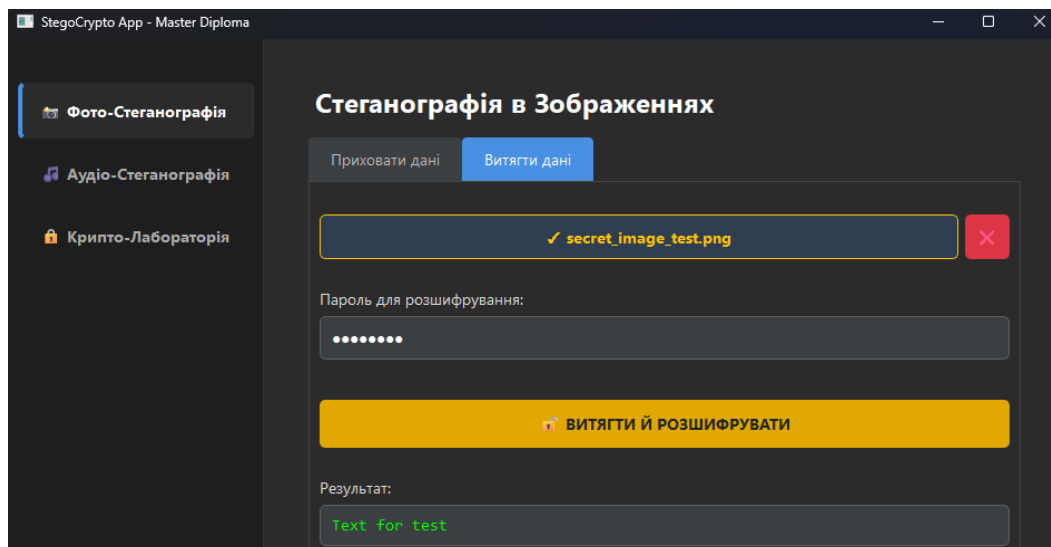


Рисунок 3.20 - Вилучення відволікаючого повідомлення за першим паролем

4. Перевірка вилучення (Етап 2): при введенні другого пароля (admin456) з того ж самого файлу успішно вилучається прихована конфіденційна інформація.

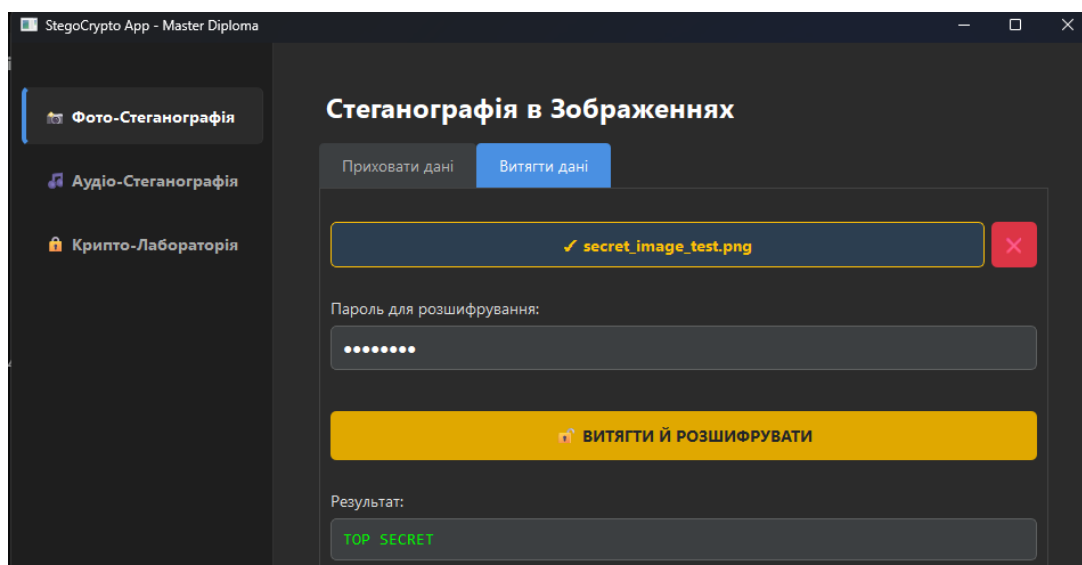


Рисунок 3.21 - Вилучення секретного повідомлення за другим паролем

Тестування підтвердило повну працездатність механізму розмежування доступу. Завдяки використанню криптографічного хешування паролів для

ініціалізації генератора координат, перетин даних (колізії) при незначному заповненні контейнера є мінімальним і автоматично корегується алгоритмом. Функція забезпечує високий рівень захисту від терморектального криптоаналізу (примусового розкриття ключів).

Окремим етапом роботи стала перевірка можливості приховування даних у звуковому потоці. Для тестування було обрано модуль AudioStego, що працює з файлами формату WAV (PCM, 16-bit).

Умови тесту:

- Контейнер: аудіофайл audio_test.wav;
- Корисне навантаження: повідомлення “Secret Audio Data Test”;
- Метод: Sequential LSB (послідовний запис у молодші біти).

Процес вбудовування відображено на рисунку 3.22.

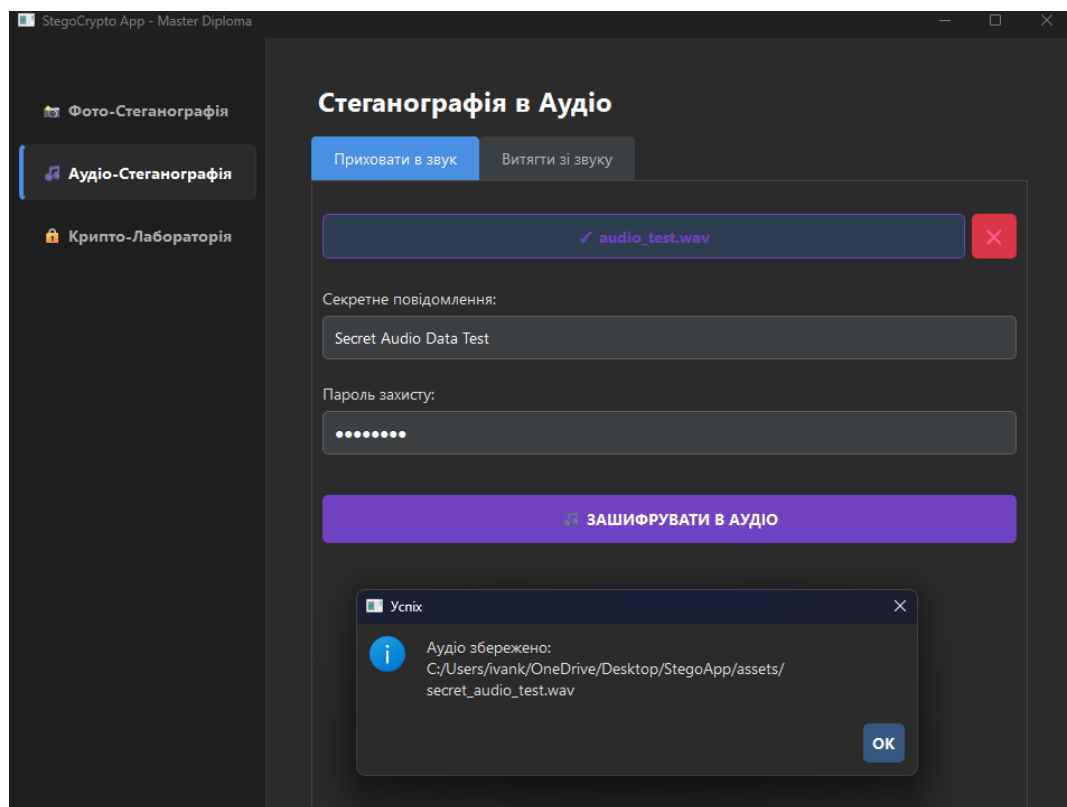


Рисунок 3.22 - Інтерфейс модуля приховування даних у звуковому файлі

Отриманий файл-стегоконтейнер було відтворено на акустичній системі стандартної якості. Суб'єктивна оцінка якості звучання не виявила сторонніх шумів, тріску або спотворень. Це підтверджує, що модифікація амплітуди

сигналу маскується природним динамічним діапазоном запису та є непомітною для людського слуху (HVS).

Для демонстрації надійності алгоритмів шифрування було проведено аналіз вихідного коду зашифрованого повідомлення перед його вбудовуванням у контейнер. Цей тест імітує ситуацію, коли зловмисник перехопив дані, але не знає пароля.

За допомогою модуля «Крипто-Лабораторія» було згенеровано шифротекст для відкритого повідомлення "Top Secret Data". Результат у шістнадцятковому (HEX) представленні наведено на рисунку 3.23.

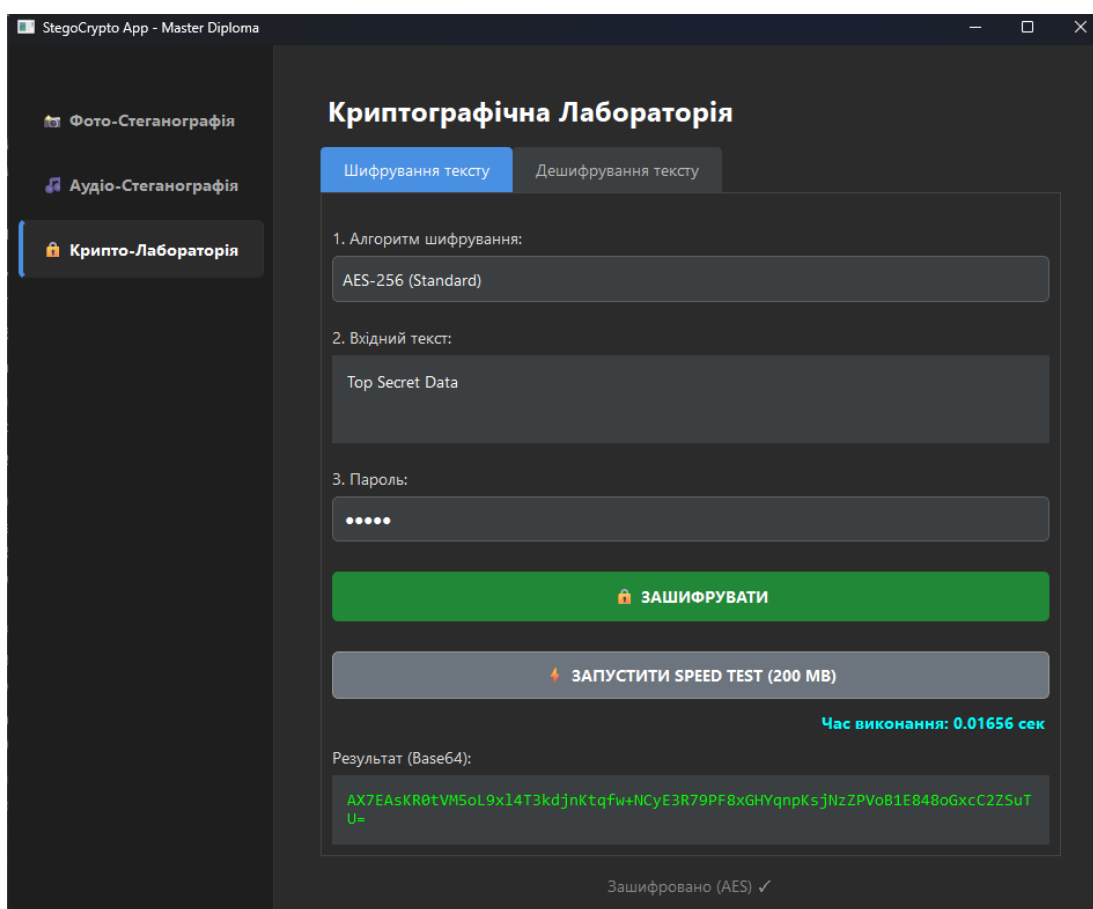


Рисунок 3.23 - Візуалізація зашифрованого пакету даних

Як видно з рисунка, вихідна послідовність не містить жодних ознак вихідного тексту. Статистичний розподіл байтів є рівномірним, що свідчить про високу ентропію. Без знання ключа (пароля) та солі (Salt), відновлення інформації методами криптоаналізу є обчислювально неможливим за розумний час.

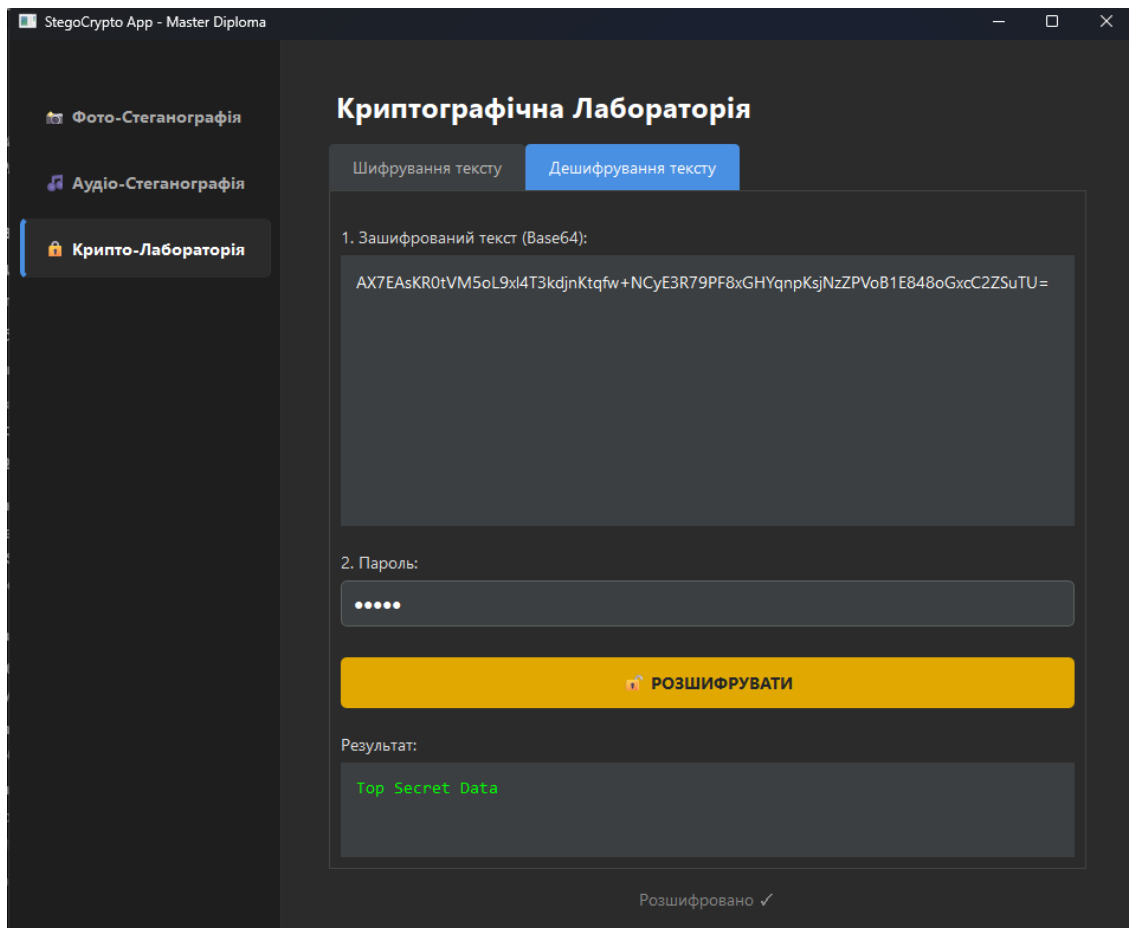


Рисунок 3.24 - Успішний результат роботи програми

Важливим етапом тестування програмного забезпечення є перевірка його поведінки в нестандартних умовах (Negative Testing). Метою цього етапу є підтвердження того, що система коректно обробляє помилки користувача та не припиняє аварійно роботу (Crash) при виникненні виключних ситуацій.

Сценарій №1: Спроба розшифрування невірним ключем.

Перевірялася реакція системи на введення неправильного пароля при спробі вилучити дані зі стегоконтейнера. Оскільки алгоритм шифрування використовує перевірку цілісності (HMAC у PBKDF2 або Poly1305 у ChaCha20), система повинна математично виявити невідповідність ключа ще до моменту формування тексту.

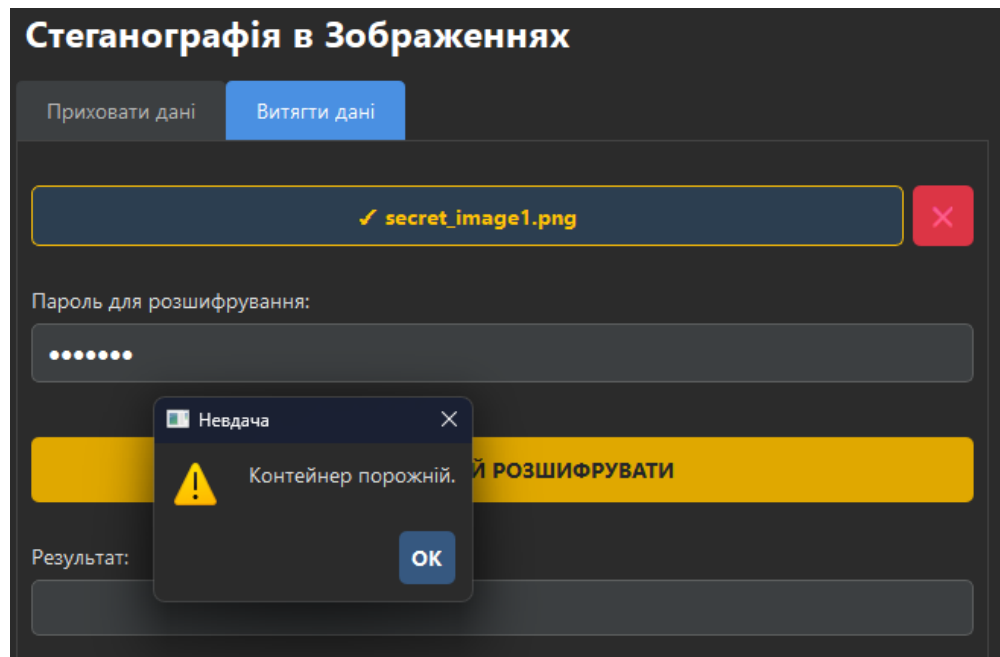


Рисунок 3.25 - Повідомлення системи при введенні невірному паролю

Як видно з рисунка, модуль CryptoHandler генерує виключення `InvalidToken` або `ValueError`, яке перехоплюється інтерфейсом, і користувачеві виводиться зрозуміле повідомлення про помилку.

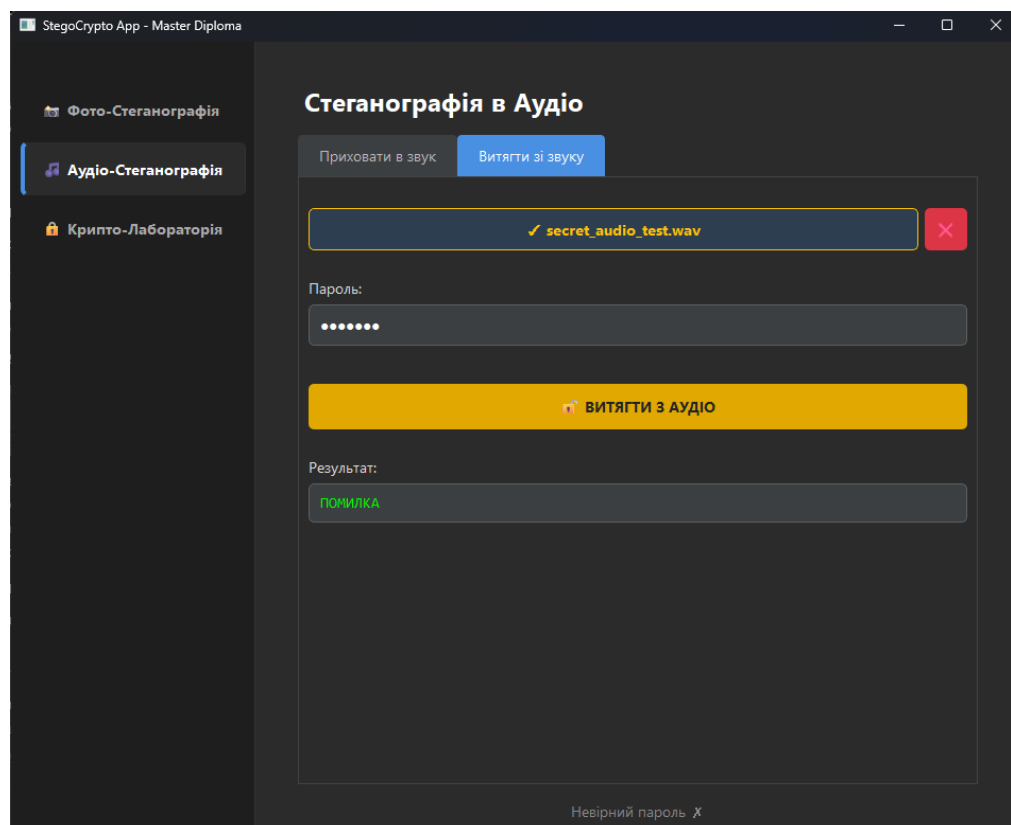


Рисунок 3.26 - Повідомлення системи при введенні невірному паролю

Сценарій №2: Переповнення контейнера.

Стеганографічна ємність обмежена розміром файлу-носія. Було проведено тест на спробу вбудувати великий обсяг тексту в зображення малої роздільної здатності (наприклад, іконку 64x64 пікселі). Система автоматично розраховує доступну кількість байтів перед початком запису.

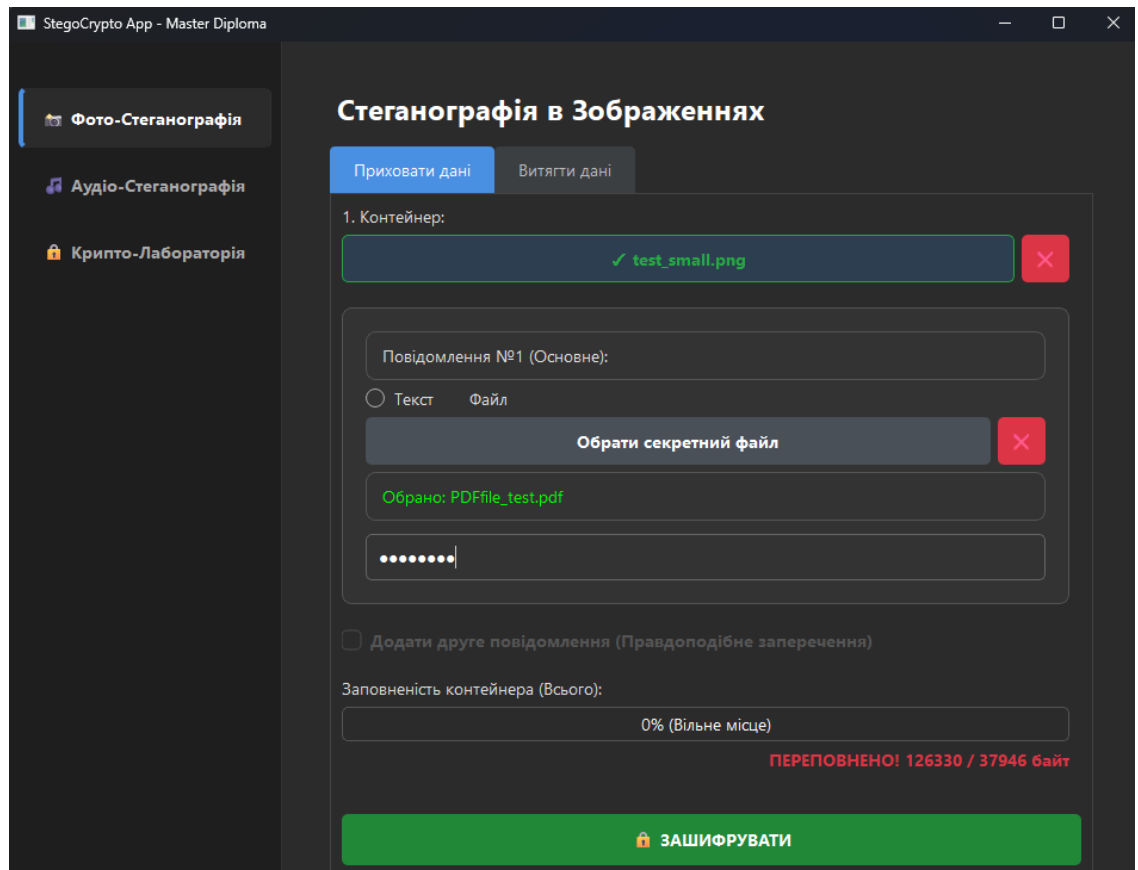


Рисунок 3.27 - Блокування операції при перевищенні доступної ємності контейнера

Цей механізм запобігає пошкодженню файлу-контейнера та втраті частини даних.

Сценарій №3: Перевірка логічної цілісності даних.

Окрему увагу було приділено тестуванню логіки інтерфейсу для запобігання помилкам «людського фактору». Зокрема, перевірялася реакція системи на спробу використати однакові паролі для основного та прихованого повідомлень. Використання ідентичних ключів для різних шарів стеганографії є

критичною вразливістю, оскільки це призвело б до колізії генераторів псевдовипадкових чисел і перезапису даних.

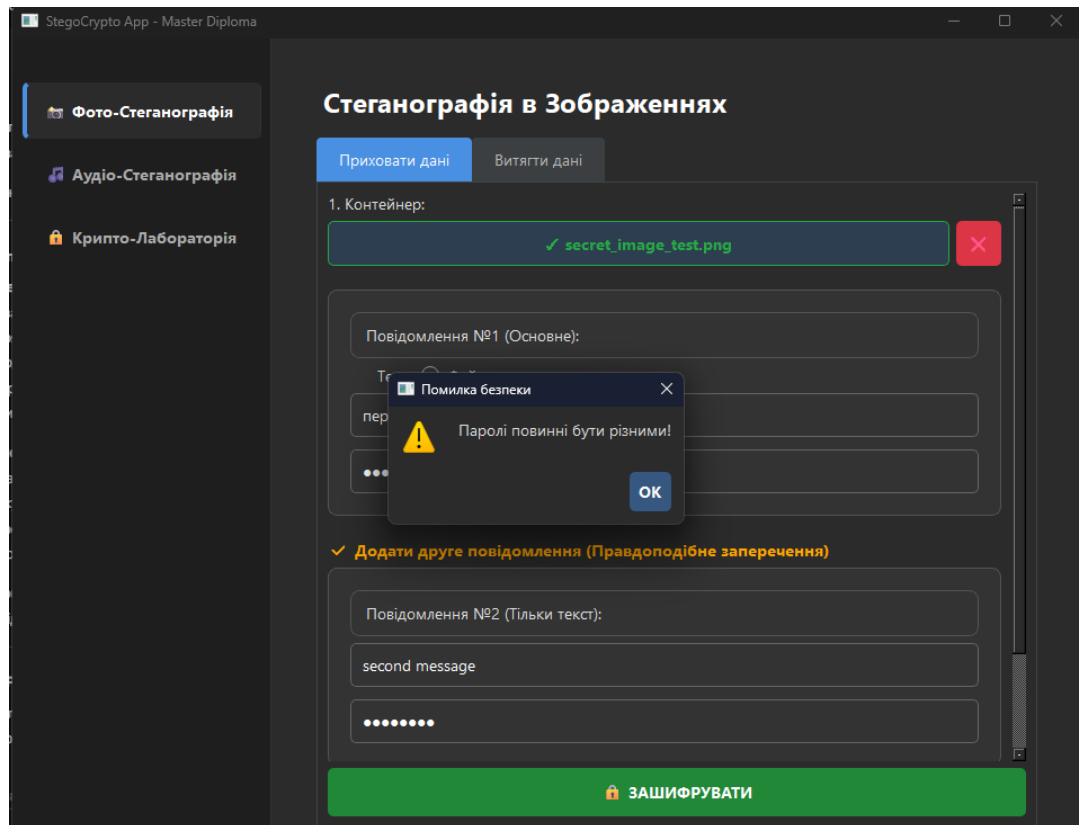


Рисунок 3.28 - Перевірка однакових паролей для різних повідомлень

У розробленій системі реалізовано превентивний контроль: при спробі введення однакових паролів кнопка запуску процесу блокується, а користувач отримує попередження. Також реалізовано блокування елементів керування під час виконання тривалих операцій шифрування, що унеможливорює запуск кількох паралельних потоків обробки, які могли б викликати конфлікт доступу до файлової системи.

Таким чином, проведений аналіз виключних ситуацій підтвердив, що програмний комплекс має високий рівень відмовостійкості (Robustness) та захищений від типових помилок експлуатації.

Висновки до третього розділу: у третьому розділі магістерської роботи виконано практичну реалізацію програмного комплексу прихованої передачі даних та проведено всебічне тестування його функціональних можливостей. Розробка здійснювалася мовою Python із використанням бібліотеки PySide6, що

дозволило створити кросплатформний додаток із модульною архітектурою, де логіка криптографічних перетворень чітко відокремлена від графічного інтерфейсу. У ході навантажувального тестування та порівняльного аналізу криптографічних алгоритмів було експериментально встановлено, що на цільовій апаратній платформі алгоритм AES-256 у режимі CTR демонструє вищу продуктивність порівняно з потоковим шифром ChaCha20. Це обумовлено підтримкою сучасними процесорами набору інструкцій AES-NI, що забезпечує швидкість обробки даних, достатню для шифрування великих медіафайлів у реальному часі.

Результати тестування стеганографічних модулів підтвердили високу ефективність обраних методів модифікації молодшого значущого біта (LSB) для зображень та аудіофайлів. Візуальний та аудіальний аналіз сформованих стегоконтейнерів не виявив артефактів чи спотворень, які могли б демаскувати наявність прихованого каналу зв'язку, що свідчить про високу скритність системи. Окремим досягненням стала успішна апробація механізму «правдоподібного заперечення», який дозволяє створювати контейнери з подвійним дном. Тести підтвердили, що розмежування доступу до інформації за допомогою різних паролів працює коректно та забезпечує захист конфіденційності навіть у сценарії примусового розкриття ключів шифрування.

Крім того, перевірка системи на відмовостійкість (Negative Testing) засвідчила надійність реалізованих механізмів обробки виключних ситуацій. Програма коректно реагує на помилки користувача, такі як введення невірної пароля або спроба запису даних, що перевищують ємність контейнера, запобігаючи аварійному завершенню роботи.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було проведено комплексне дослідження методів захисту інформації, що базуються на поєднанні криптографічних алгоритмів та стеганографічного приховування даних. Аналіз сучасних загроз інформаційній безпеці підтвердив, що використання лише традиційного шифрування є недостатнім в умовах тотального моніторингу каналів зв'язку, що обґрунтувало необхідність розробки гібридної системи, яка забезпечує не лише конфіденційність змісту повідомлення, а й факт його існування.

На основі теоретичних досліджень було розроблено архітектуру програмного комплексу прихованої передачі даних, реалізовану мовою Python з використанням бібліотек PySide6 та Cryptography. Запропоноване рішення інтегрує симетричні алгоритми шифрування (AES-256, ChaCha20) з методами модифікації молодшого значущого біта (LSB) для мультимедійних контейнерів. Ключовою особливістю розробленої системи є реалізація механізму «правдоподібного заперечення» (Plausible Deniability), який дозволяє створювати багаторівневі приховані сховища та захищати критично важливі дані навіть у разі примусового розкриття ключів.

Практична реалізація та тестування системи підтвердили її високу ефективність та надійність. Порівняльний аналіз криптографічних ядер продемонстрував перевагу алгоритму AES-256 на десктопних платформах, забезпечуючи високу швидкість обробки великих масивів даних. Експерименти з візуального та аудіального контролю стегоконтейнерів засвідчили відсутність помітних артефактів, що гарантує скритність каналу передачі, а реалізовані алгоритми валідації забезпечують стабільну роботу програмного засобу в умовах некоректних дій користувача.

Отримані результати демонструють, що розроблений програмний комплекс є ефективним інструментом для забезпечення безпечного обміну інформацією в ненадійному середовищі.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Cheddad A., Condell J., Curran K., Mc Kevitt P. Digital image steganography: Survey and analysis of current methods // Signal Processing. – 2010. – Vol. 90, Issue 3. URL: <https://core.ac.uk/download/pdf/11470876.pdf>
2. Fridrich J. Applications of Data Hiding in Digital Images // Tutorial for the ISPACS '98 Conference. – 1998. URL: <http://www.ws.binghamton.edu/fridrich/Research/ispacs98.pdf>
3. Johnson N. F., Jajodia S. Exploring Steganography: Seeing the Unseen // IEEE Computer. – 1998. – Vol. 31, No. 2. URL: <http://www.jitc.com/pub/r2026.pdf>
4. Westfeld A., Pfitzmann A. Attacks on Steganographic Systems // Proceedings of Information Hiding Workshop. – Dresden, 1999. URL: https://link.springer.com/content/pdf/10.1007/3-540-45496-9_5.pdf
5. Provos N., Honeyman P. Hide and Seek: An Introduction to Steganography // IEEE Security & Privacy. – 2003. URL: <http://niels.xtdnet.nl/papers/stego.pdf>
6. Mielikainen J. LSB Matching Revisited // IEEE Signal Processing Letters. – 2006. – Vol. 13, No. 5. URL: https://www.blitzcalc.com/pdf/LSB_Matching_Revisited.pdf
7. Ker A. D. Steganalysis of LSB matching in grayscale images // IEEE Signal Processing Letters. – 2005. – Vol. 12, Issue 6. URL: <https://www.cs.ox.ac.uk/files/658/lsb-matching.pdf>
8. Roelofs G. Portable Network Graphics (PNG) Specification (Second Edition) // W3C Recommendation. – 2003. URL: <https://www.w3.org/TR/PNG/>
9. Barni M. Symmetric Cryptography [Lecture Notes] // University of Siena. – 2018. URL: https://clem.dii.unisi.it/~vipv/website_resources/courses/cybersecurity/Cybersec_sym_crypto.pdf
10. National Institute of Standards and Technology (NIST). FIPS PUB 197: Advanced Encryption Standard (AES) // NIST Computer Security Resource Center. – 2001. URL: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>

11. Shannon C. E. Communication Theory of Secrecy Systems // Bell System Technical Journal. – 1949. – Vol. 28, Issue 4. URL: <http://www.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf>
12. Gueron S. Intel Advanced Encryption Standard (AES) New Instructions Set // Intel White Paper. – 2010. URL: <https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf>
13. Nir Y., Langley A. RFC 7539: ChaCha20 and Poly1305 for IETF Protocols // Internet Engineering Task Force (IETF). – 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7539>
14. Bernstein D. J. ChaCha, a variant of Salsa20 // The cr.yp.to blog. – 2008. URL: <https://cr.yp.to/chacha/chacha-20080128.pdf>
15. Langley A. ChaCha20 and Poly1305 for TLS // ImperialViolet (Google Security Blog). – 2013. URL: <https://www.imperialviolet.org/2013/10/03/chacha20.html>
16. Schneier B. Rubber-Hose Cryptanalysis // Schneier on Security. – 2008. URL: https://www.schneier.com/blog/archives/2008/10/rubberhose_cryp.html
17. Hansmann R. Data Hiding in Digital Images using Steganography and Plausible Deniability // SANS Institute InfoSec Reading Room. – 2003. URL: <https://www.sans.org/white-papers/1271/>
18. Barker E., Kelsey J. NIST SP 800-90A Rev. 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators // NIST Computer Security Resource Center. – 2015. URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>
19. Laghari S. et al. A Review on Pseudo Random Number Generators for Cryptography // International Journal of Computer Applications. – 2014. URL: <https://www.ijcaonline.org/archives/volume105/number12/18428-9759>
20. Kaliski B. RFC 2898: PKCS #5: Password-Based Cryptography Specification Version 2.0 // Internet Engineering Task Force (IETF). – 2000. URL: <https://datatracker.ietf.org/doc/html/rfc2898>

21. Python Software Foundation. Python 3.10.0 Documentation. URL: <https://docs.python.org/3/>
22. PyCA/Cryptography. Cryptography 39.0.0 documentation. URL: <https://cryptography.io/en/latest/>
23. The Qt Company. Qt for Python (PySide6) Documentation // Official Documentation. URL: <https://doc.qt.io/qtforpython-6/>
24. Clark A. Pillow (PIL Fork) 10.0.0 Documentation // Read the Docs. URL: <https://pillow.readthedocs.io/en/stable/>

ДОДАТКИ

Лістинг коду main.py

```
import sys
from PySide6.QtWidgets import QApplication
from app.gui.interface import MainWindow

def main():
    app = QApplication(sys.argv)

    # Завантажуємо наше головне вікно
    window = MainWindow()
    window.show()

    sys.exit(app.exec())

if __name__ == "__main__":
    main()
```

Лістинг коду interface.py

```

import time
import os
from PySide6.QtWidgets import (
    QMainWindow, QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QLabel,
    QLineEdit, QFileDialog, QTabWidget, QMessageBox, QStackedWidget, QFrame,
    QComboBox, QTextEdit, QApplication, QProgressBar
)
from PySide6.QtCore import Qt
from PIL import Image
from app.core.crypto import CryptoHandler
from app.core.stego import StegoHandler
from app.core.stego_audio import AudioStego

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("StegoCrypto App - Master Diploma")
        self.resize(950, 680)

        # Ініціалізація бекенду
        self.crypto = CryptoHandler()
        self.stego = StegoHandler()
        self.audio_stego = AudioStego()

        # Змінні
        self.source_image_path = None
        self.stego_image_path = None
        self.source_audio_path = None
        self.stego_audio_path = None

        # --- ЗАГАЛЬНИЙ СТИЛЬ (DARK THEME) ---
        self.setStyleSheet("""
            QMainWindow, QWidget {
                background-color: #2b2b2b;
                color: #e0e0e0;
                font-family: 'Segoe UI', sans-serif;
                font-size: 14px;
            }
            QLineEdit, QTextEdit, QComboBox {
                background-color: #3c3f41;
                border: 1px solid #5e6060;
                border-radius: 5px;
                padding: 8px;
                color: #ffffff;
            }
            QLineEdit:focus, QTextEdit:focus, QComboBox:focus {
                border: 1px solid #4a90e2;
            }
            QPushButton {
                background-color: #365880;
                color: white;
                border-radius: 5px;
                padding: 8px;
            }
        """)

```

```

        font-weight: bold;
    }
    QPushButton:hover {
        background-color: #406a9e;
    }
    QPushButton:disabled {
        background-color: #404040;
        color: #808080;
    }
    QTabWidget::pane {
        border: 1px solid #444;
        background-color: #2b2b2b;
    }
    QTabBar::tab {
        background: #3c3f41;
        color: #bbb;
        padding: 10px 20px;
        border-top-left-radius: 4px;
        border-top-right-radius: 4px;
    }
    QTabBar::tab:selected {
        background: #4a90e2;
        color: white;
    }
    QLabel {
        color: #e0e0e0;
    }
    QComboBox::drop-down {
        border: 0px;
    }
    """)

# --- ГОЛОВНИЙ МАКЕТ ---
main_widget = QWidget()
self.setCentralWidget(main_widget)
main_layout = QHBoxLayout()
main_layout.setContentsMargins(0, 0, 0, 0)
main_layout.setSpacing(0)
main_widget.setLayout(main_layout)

# 1. БОКОБЕ МЕНЮ
self.side_menu = QFrame()
self.side_menu.setFixedWidth(230)
self.side_menu.setStyleSheet("background-color: #1e1e1e; border-right:
1px solid #333;")
main_layout.addWidget(self.side_menu)

menu_layout = QVBoxLayout()
self.side_menu.setLayout(menu_layout)
menu_layout.addSpacing(30)

# Кнопки навігації
self.btn_menu_photo = self.create_menu_button("📷 Фото-Стеганографія")
self.btn_menu_audio = self.create_menu_button("🎵 Аудіо-Стеганографія")
self.btn_menu_crypto = self.create_menu_button("🔒 Кripto-Лабораторія")

menu_layout.addWidget(self.btn_menu_photo)
menu_layout.addWidget(self.btn_menu_audio)

```

```

menu_layout.addWidget(self.btn_menu_crypto)

menu_layout.addStretch()

# 2. ОБЛАСТЬ КОНТЕНТУ
self.content_area = QStackedWidget()
main_layout.addWidget(self.content_area)

# --- СТОПІНКИ ---
self.page_photo = QWidget()
self.init_photo_page()
self.content_area.addWidget(self.page_photo)

self.page_audio = QWidget()
self.init_audio_page()
self.content_area.addWidget(self.page_audio)

self.page_crypto = QWidget()
self.init_crypto_page() # Тепер тут справжній UI
self.content_area.addWidget(self.page_crypto)

# Лоріка кнопок меню
self.btn_menu_photo.clicked.connect(lambda: self.switch_page(0,
self.btn_menu_photo))
self.btn_menu_audio.clicked.connect(lambda: self.switch_page(1,
self.btn_menu_audio))
self.btn_menu_crypto.clicked.connect(lambda: self.switch_page(2,
self.btn_menu_crypto))

# Старт з першої сторінки
self.switch_page(0, self.btn_menu_photo)

def create_menu_button(self, text):
    btn = QPushButton(text)
    btn.setFixedHeight(50)
    btn.setCursor(Qt.PointingHandCursor)
    btn.setStyleSheet("""
        QPushButton {
            background-color: transparent;
            border: none;
            color: #aaaaaa;
            text-align: left;
            padding-left: 20px;
            font-size: 14px;
        }
        QPushButton:hover {
            background-color: #2b2b2b;
            color: white;
        }
    """)
    return btn

def switch_page(self, index, active_btn):
    self.content_area.setCurrentIndex(index)
    # Скидання стилів
    for btn in [self.btn_menu_photo, self.btn_menu_audio,
self.btn_menu_crypto]:
        btn.setStyleSheet("""
            QPushButton {

```

```

        background-color: transparent;
        border: none;
        color: #aaaaaa;
        text-align: left;
        padding-left: 20px;
    }
    QPushButton:hover {
        background-color: #2b2b2b;
        color: white;
    }
    """
)
# Активна кнопка
active_btn.setStyleSheet("""
    QPushButton {
        background-color: #2b2b2b;
        border-left: 4px solid #4a90e2;
        color: white;
        text-align: left;
        padding-left: 16px;
        font-weight: bold;
    }
    """)

# =====
# 1. ФОТО-СТЕГАНОГРАФІЯ
# =====
def init_photo_page(self):
    layout = QVBoxLayout()
    layout.setContentsMargins(40, 40, 40, 40)
    self.page_photo.setLayout(layout)

    header = QLabel("Стеганографія в Зображеннях")
    header.setStyleSheet("font-size: 24px; font-weight: bold; color:
#ffffff; margin-bottom: 10px;")
    layout.addWidget(header)

    tabs = QTabWidget()
    layout.addWidget(tabs)

    self.tab_photo_embed = QWidget()
    self.init_photo_embed_ui()
    tabs.addTab(self.tab_photo_embed, "Приховати дані")

    self.tab_photo_extract = QWidget()
    self.init_photo_extract_ui()
    tabs.addTab(self.tab_photo_extract, "Витягти дані")

    self.status_label_photo = QLabel("")
    self.status_label_photo.setAlignment(Qt.AlignCenter)
    self.status_label_photo.setStyleSheet("color: #888; margin-top: 10px;")
    layout.addWidget(self.status_label_photo)

def init_photo_embed_ui(self):
    # --- ІМПОРТИ ---
    from PySide6.QtWidgets import (
        QScrollArea, QRadioButton, QButtonGroup, QCheckBox, QFrame
    )

    layout = QVBoxLayout()

```

```

self.tab_photo_embed.setLayout(layout)

# Скрол-зона
scroll = QScrollArea()
scroll.setWidgetResizable(True)
scroll.setStyleSheet("background-color: transparent; border: none;")

scroll_content = QWidget()
scroll_layout = QVBoxLayout(scroll_content)
scroll_layout.setContentsMargins(0, 0, 10, 0)
scroll.setWidget(scroll_content)
layout.addWidget(scroll)

# --- 1. ВИБІР КАРТИНКИ ---
img_layout = QHBoxLayout()
self.btn_select_img = QPushButton("📁 Оберіть файл зображення (PNG)")
self.btn_select_img.setFixedHeight(40)
self.btn_select_img.setStyleSheet("background-color: #3c3f41; border:
1px dashed #666; color: #aaa;")
self.btn_select_img.clicked.connect(self.select_source_image)
img_layout.addWidget(self.btn_select_img)

self.btn_clear_img = QPushButton("✖")
self.btn_clear_img.setFixedSize(40, 40)
self.btn_clear_img.setStyleSheet("background-color: #dc3545; color:
white; font-weight: bold;")
self.btn_clear_img.setToolTip("Очистити")
self.btn_clear_img.clicked.connect(self.clear_photo_selection)
self.btn_clear_img.hide()
img_layout.addWidget(self.btn_clear_img)

scroll_layout.addWidget(QLabel("1. Контейнер:"))
scroll_layout.addLayout(img_layout)
scroll_layout.addSpacing(15)

# === СТИЛЬ ДЛЯ КАРТОК ===
card_style = """
    QFrame {
        background-color: #333333;
        border: 1px solid #555555;
        border-radius: 8px;
        padding: 10px;
    }
    """

# --- 2. ПЕРШЕ ПОВІДОМЛЕННЯ (ОСНОВНЕ) ---
group_msg1 = QFrame()
group_msg1.setStyleSheet(card_style)
layout_g1 = QVBoxLayout(group_msg1)

layout_g1.addWidget(QLabel("Повідомлення №1 (Основне:)))

# Перемикач Текст/Файл
type_layout = QHBoxLayout()
self.radio_text = QRadioButton("Текст")
self.radio_file = QRadioButton("Файл")
self.radio_text.setChecked(True)
self.btn_group_type = QButtonGroup()

```

```

self.btn_group_type.addButton(self.radio_text)
self.btn_group_type.addButton(self.radio_file)
type_layout.addWidget(self.radio_text)
type_layout.addWidget(self.radio_file)
type_layout.addStretch()
layout_g1.addLayout(type_layout)
self.radio_text.toggled.connect(self.toggle_input_mode)

# Поле тексту №1
self.container_text = QWidget()
l_text = QVBoxLayout(self.container_text)
l_text.setContentsMargins(0, 0, 0, 0)
self.text_input_photo = QLineEdit()
self.text_input_photo.setPlaceholderText("Введіть перше
повідомлення...")
self.text_input_photo.setStyleSheet("border: 1px solid #666;")

# ПІДКЛЮЧЕННЯ №1 (вже було)
self.text_input_photo.textChanged.connect(self.update_capacity_bar)

l_text.addWidget(self.text_input_photo)
layout_g1.addWidget(self.container_text)

# Поле файлу №1
self.container_file = QWidget()
self.container_file.hide()
l_file = QVBoxLayout(self.container_file)
l_file.setContentsMargins(0, 0, 0, 0)

f_layout = QHBoxLayout()
self.btn_select_secret_file = QPushButton("Обрати секретний файл")
self.btn_select_secret_file.setFixedHeight(40)
self.btn_select_secret_file.clicked.connect(self.select_secret_file)
self.btn_select_secret_file.setStyleSheet("background-color: #495057;
color: white;")
f_layout.addWidget(self.btn_select_secret_file)

self.btn_clear_file = QPushButton("✖")
self.btn_clear_file.setFixedSize(40, 40)
self.btn_clear_file.setStyleSheet("background-color: #dc3545; color:
white; font-weight: bold;")
self.btn_clear_file.clicked.connect(self.clear_secret_file)
self.btn_clear_file.hide()
f_layout.addWidget(self.btn_clear_file)

l_file.addLayout(f_layout)
self.lbl_secret_file = QLabel("Файл не обрано")
self.lbl_secret_file.setStyleSheet("font-size: 10px; color: #aaa;")
l_file.addWidget(self.lbl_secret_file)
layout_g1.addWidget(self.container_file)

# Пароль №1
layout_g1.addSpacing(5)
self.pass_input_photo_enc = QLineEdit()
self.pass_input_photo_enc.setPlaceholderText("Пароль №1")
self.pass_input_photo_enc.setEchoMode(QLineEdit.Password)
self.pass_input_photo_enc.setStyleSheet("border: 1px solid #666;")
layout_g1.addWidget(self.pass_input_photo_enc)

```

```

scroll_layout.addWidget(group_msg1)
scroll_layout.addSpacing(15)

# --- ЧЕКБОКС "ПОДВІЙНЕ ДНО" ---
self.chk_double_stego = QCheckBox("Додати друге повідомлення
(Правдоподібне заперечення)")
self.chk_double_stego.setStyleSheet("color: #ffa500; font-weight:
bold;")
self.chk_double_stego.toggled.connect(self.toggle_double_stego_ui)

# ПІДКЛЮЧЕННЯ №2 (Якщо вимкнути галочку - смужка має оновитись)
self.chk_double_stego.toggled.connect(self.update_capacity_bar)

scroll_layout.addWidget(self.chk_double_stego)

# --- 3. ДРУГЕ ПОВІДОМЛЕННЯ (ПРИХОВАНЕ) ---
self.group_msg2 = QFrame()
self.group_msg2.setStyleSheet(card_style)
self.group_msg2.hide()
layout_g2 = QVBoxLayout(self.group_msg2)

layout_g2.addWidget(QLabel("Повідомлення №2 (Тільки текст:"))
self.text_input_photo_2 = QLineEdit()
self.text_input_photo_2.setPlaceholderText("Введіть друге
повідомлення...")
self.text_input_photo_2.setStyleSheet("border: 1px solid #666;")

# ПІДКЛЮЧЕННЯ №3 (Ввід тексту в друге поле оновлює смужку)
self.text_input_photo_2.textChanged.connect(self.update_capacity_bar)

layout_g2.addWidget(self.text_input_photo_2)

layout_g2.addSpacing(5)
self.pass_input_photo_enc_2 = QLineEdit()
self.pass_input_photo_enc_2.setPlaceholderText("Пароль №2 (Має
відрізнитися!)")
self.pass_input_photo_enc_2.setEchoMode(QLineEdit.Password)
self.pass_input_photo_enc_2.setStyleSheet("border: 1px solid #666;")
layout_g2.addWidget(self.pass_input_photo_enc_2)

scroll_layout.addWidget(self.group_msg2)
scroll_layout.addSpacing(15)

# --- ІНДИКАТОР ---
scroll_layout.addWidget(QLabel("Заповненість контейнера (Всього:"))
self.capacity_bar = QProgressBar()
self.capacity_bar.setValue(0)
self.capacity_bar.setTextVisible(True)
self.capacity_bar.setFormat("%p% (Вільне місце)")
self.capacity_bar.setStyleSheet(
    "QProgressBar { border: 1px solid #555; border-radius: 5px; text-
align: center; background-color: #2b2b2b; color: white; } QProgressBar::chunk
{ background-color: #28a745; width: 20px; }")
scroll_layout.addWidget(self.capacity_bar)
self.lbl_capacity_info = QLabel("0 / 0 байт")
self.lbl_capacity_info.setAlignment(Qt.AlignRight)
self.lbl_capacity_info.setStyleSheet("font-size: 10px; color: #aaa;")
scroll_layout.addWidget(self.lbl_capacity_info)

```

```

scroll_layout.addStretch()

# КНОПКА ЗАПУСКУ
self.btn_run_photo_embed = QPushButton("🔒 ЗАШИФРУВАТИ")
self.btn_run_photo_embed.setStyleSheet(
    "background-color: #218838; color: white; padding: 12px; font-size:
14px;")
self.btn_run_photo_embed.clicked.connect(self.process_embed_photo)
self.btn_run_photo_embed.setEnabled(False)
layout.addWidget(self.btn_run_photo_embed)

def init_photo_extract_ui(self):
    layout = QVBoxLayout()
    self.tab_photo_extract.setLayout(layout)
    layout.addSpacing(20)

    # --- БЛОК ВИБОРУ СТЕГО-КАРТИНКИ (3 ХРЕСТИКОМ) ---
    img_layout = QHBoxLayout()

    self.btn_select_stego_img = QPushButton("📁 Оберіть зображення з
секретом")
    self.btn_select_stego_img.setFixedHeight(40)
    self.btn_select_stego_img.setStyleSheet("background-color: #3c3f41;
border: 1px dashed #666; color: #aaa;")
    self.btn_select_stego_img.clicked.connect(self.select_stego_image)
    img_layout.addWidget(self.btn_select_stego_img)

    # Хрестик 40x40
    self.btn_clear_stego_img = QPushButton("✖")
    self.btn_clear_stego_img.setFixedSize(40, 40)
    self.btn_clear_stego_img.setStyleSheet("background-color: #dc3545;
color: white; font-weight: bold;")
    self.btn_clear_stego_img.setToolTip("Очистити")

self.btn_clear_stego_img.clicked.connect(self.clear_stego_image_selection) #
Новий метод
    self.btn_clear_stego_img.hide()
    img_layout.addWidget(self.btn_clear_stego_img)

    layout.addLayout(img_layout)

    layout.addSpacing(20)
    layout.addWidget(QLabel("Пароль для розшифрування:"))
    self.pass_input_photo_dec = QLineEdit()
    self.pass_input_photo_dec.setPlaceholderText(".....")
    self.pass_input_photo_dec.setEchoMode(QLineEdit.Password)
    layout.addWidget(self.pass_input_photo_dec)

    layout.addSpacing(30)
    self.btn_run_photo_extract = QPushButton("🔓 ВИТЯГТИ Й РОЗШИФРУВАТИ")
    self.btn_run_photo_extract.setStyleSheet("background-color: #e0a800;
color: #212529; padding: 12px;")
    self.btn_run_photo_extract.clicked.connect(self.process_extract_photo)
    self.btn_run_photo_extract.setEnabled(False)
    layout.addWidget(self.btn_run_photo_extract)

    layout.addSpacing(20)
    layout.addWidget(QLabel("Результат:"))

```

```

        self.result_display_photo = QLineEdit()
        self.result_display_photo.setReadOnly(True)
        self.result_display_photo.setStyleSheet("color: #00ff00; font-family:
Consolas;")
        layout.addWidget(self.result_display_photo)
        layout.addStretch()

# =====
# 2. АУДІО-СТЕГАНОГРАФІЯ
# =====
def init_audio_page(self):
    layout = QVBoxLayout()
    layout.setContentsMargins(40, 40, 40, 40)
    self.page_audio.setLayout(layout)

    header = QLabel("Стеганографія в Аудіо")
    header.setStyleSheet("font-size: 24px; font-weight: bold; color:
#ffffff; margin-bottom: 10px;")
    layout.addWidget(header)

    tabs = QTabWidget()
    layout.addWidget(tabs)

    self.tab_audio_embed = QWidget()
    self.init_audio_embed_ui()
    tabs.addTab(self.tab_audio_embed, "Приховати в звук")

    self.tab_audio_extract = QWidget()
    self.init_audio_extract_ui()
    tabs.addTab(self.tab_audio_extract, "Витягти зі звуку")

    self.status_label_audio = QLabel("")
    self.status_label_audio.setAlignment(Qt.AlignCenter)
    self.status_label_audio.setStyleSheet("color: #888; margin-top: 10px;")
    layout.addWidget(self.status_label_audio)

def init_audio_embed_ui(self):
    layout = QVBoxLayout()
    self.tab_audio_embed.setLayout(layout)
    layout.addSpacing(20)

    # --- БЛОК ВИБОРУ АУДІО (З КНОПКОЮ ОЧИСТКИ) ---
    audio_layout = QHBoxLayout()

    self.btn_select_audio = QPushButton("🎵 Оберіть файл аудіо (WAV)")
    self.btn_select_audio.setFixedHeight(40)
    self.btn_select_audio.setStyleSheet("background-color: #3c3f41; border:
1px dashed #666; color: #aaa;")
    self.btn_select_audio.clicked.connect(self.select_source_audio)
    audio_layout.addWidget(self.btn_select_audio)

    # Кнопка очистки (X) - така ж, як для фото
    self.btn_clear_audio = QPushButton("✖")
    self.btn_clear_audio.setFixedSize(40, 40)
    self.btn_clear_audio.setStyleSheet("background-color: #dc3545; color:
white; font-weight: bold;")
    self.btn_clear_audio.setToolTip("Очистити вибране аудіо")
    self.btn_clear_audio.clicked.connect(self.clear_audio_selection)

```

```

self.btn_clear_audio.hide() # Схована
audio_layout.addWidget(self.btn_clear_audio)

layout.addLayout(audio_layout)
# -----

layout.addSpacing(20)

layout.addWidget(QLabel("Секретне повідомлення:"))
self.text_input_audio = QLineEdit()
self.text_input_audio.setPlaceholderText("Текст для аудіо...")
layout.addWidget(self.text_input_audio)

layout.addSpacing(15)

layout.addWidget(QLabel("Пароль захисту:"))
self.pass_input_audio_enc = QLineEdit()
self.pass_input_audio_enc.setEchoMode(QLineEdit.Password)
layout.addWidget(self.pass_input_audio_enc)

layout.addSpacing(30)

self.btn_run_audio_embed = QPushButton("🎵 ЗАШИФРУВАТИ В АУДІО")
self.btn_run_audio_embed.setStyleSheet(
    "background-color: #6f42c1; color: white; font-size: 14px; padding:
12px;")
self.btn_run_audio_embed.clicked.connect(self.process_embed_audio)
self.btn_run_audio_embed.setEnabled(False)
layout.addWidget(self.btn_run_audio_embed)
layout.addStretch()

def init_audio_extract_ui(self):
    layout = QVBoxLayout()
    self.tab_audio_extract.setLayout(layout)
    layout.addSpacing(20)

    # --- БЛОК ВИБОРУ СТЕГО-АУДІО (З ХРЕСТИКОМ) ---
    audio_layout = QHBoxLayout()

    self.btn_select_stego_audio = QPushButton("🎵 Оберіть аудіо з
секретом")
    self.btn_select_stego_audio.setFixedHeight(40)
    self.btn_select_stego_audio.setStyleSheet("background-color: #3c3f41;
border: 1px dashed #666; color: #aaa;")
    self.btn_select_stego_audio.clicked.connect(self.select_stego_audio)
    audio_layout.addWidget(self.btn_select_stego_audio)

    # Хрестик 40x40
    self.btn_clear_stego_audio = QPushButton("✖")
    self.btn_clear_stego_audio.setFixedSize(40, 40)
    self.btn_clear_stego_audio.setStyleSheet("background-color: #dc3545;
color: white; font-weight: bold;")
    self.btn_clear_stego_audio.setToolTip("Очистити")

self.btn_clear_stego_audio.clicked.connect(self.clear_stego_audio_selection)
# Новий метод
self.btn_clear_stego_audio.hide()
audio_layout.addWidget(self.btn_clear_stego_audio)

```

```

layout.addLayout(audio_layout)

layout.addSpacing(20)
layout.addWidget(QLabel("Пароль:"))
self.pass_input_audio_dec = QLineEdit()
self.pass_input_audio_dec.setPlaceholderText(".....")
self.pass_input_audio_dec.setEchoMode(QLineEdit.Password)
layout.addWidget(self.pass_input_audio_dec)

layout.addSpacing(30)
self.btn_run_audio_extract = QPushButton("🔒 ВИТЯГТИ З АУДІО")
self.btn_run_audio_extract.setStyleSheet("background-color: #e0a800;
color: #212529; padding: 12px;")
self.btn_run_audio_extract.clicked.connect(self.process_extract_audio)
self.btn_run_audio_extract.setEnabled(False)
layout.addWidget(self.btn_run_audio_extract)

layout.addSpacing(20)
layout.addWidget(QLabel("Результат:"))
self.result_display_audio = QLineEdit()
self.result_display_audio.setReadOnly(True)
self.result_display_audio.setStyleSheet("color: #00ff00; font-family:
Consolas;")
layout.addWidget(self.result_display_audio)
layout.addStretch()

# =====
# 3. КРИПТО-ЛАБОРАТОРІЯ (НОВЕ!)
# =====
def init_crypto_page(self):
    layout = QVBoxLayout()
    layout.setContentsMargins(40, 40, 40, 40)
    self.page_crypto.setLayout(layout)

    header = QLabel("Криптографічна Лабораторія")
    header.setStyleSheet("font-size: 24px; font-weight: bold; color:
#ffffff; margin-bottom: 10px;")
    layout.addWidget(header)

    tabs = QTabWidget()
    layout.addWidget(tabs)

    # Tab Encrypt
    self.tab_crypto_enc = QWidget()
    self.init_crypto_enc_ui()
    tabs.addTab(self.tab_crypto_enc, "Шифрування тексту")

    # Tab Decrypt
    self.tab_crypto_dec = QWidget()
    self.init_crypto_dec_ui()
    tabs.addTab(self.tab_crypto_dec, "Дешифрування тексту")

    self.status_label_crypto = QLabel("")
    self.status_label_crypto.setAlignment(Qt.AlignCenter)
    self.status_label_crypto.setStyleSheet("color: #888; margin-top:
10px;")
    layout.addWidget(self.status_label_crypto)

```

```

def init_crypto_enc_ui(self):
    layout = QVBoxLayout()
    self.tab_crypto_enc.setLayout(layout)
    layout.addSpacing(20)

    # Вибір алгоритму
    layout.addWidget(QLabel("1. Алгоритм шифрування:"))
    self.combo_algo = QComboBox()
    self.combo_algo.addItem("AES-256 (Standard)", "ChaCha20 (High
Speed)")])
    self.combo_algo.setFixedHeight(40)
    layout.addWidget(self.combo_algo)

    layout.addSpacing(15)
    layout.addWidget(QLabel("2. Вхідний текст:"))
    self.text_input_crypto = QTextEdit()
    self.text_input_crypto.setPlaceholderText("Введіть текст для
шифрування...")
    layout.addWidget(self.text_input_crypto)

    layout.addSpacing(15)
    layout.addWidget(QLabel("3. Пароль:"))
    self.pass_input_crypto = QLineEdit()
    self.pass_input_crypto.setPlaceholderText(".....")
    self.pass_input_crypto.setEchoMode(QLineEdit.Password)
    layout.addWidget(self.pass_input_crypto)

    layout.addSpacing(20)

    # Кнопка звичайного шифрування
    self.btn_run_crypto_enc = QPushButton("🔒 ЗАШИФРУВАТИ")
    self.btn_run_crypto_enc.setStyleSheet("background-color: #218838;
color: white; padding: 12px;")
    self.btn_run_crypto_enc.clicked.connect(self.process_crypto_encrypt)
    layout.addWidget(self.btn_run_crypto_enc)

    # --- НОБЕ: Кнопка Бенчмарку ---
    layout.addSpacing(10)
    # --- НОБЕ: Кнопка Бенчмарку (200 MB) ---
    layout.addSpacing(10)
    self.btn_benchmark = QPushButton("⚡ ЗАПУСТИТИ SPEED TEST (200 MB)") #
Змінили текст
    self.btn_benchmark.setStyleSheet(
        "background-color: #6c757d; color: white; padding: 10px; border:
1px solid #999; font-weight: bold;")
    self.btn_benchmark.clicked.connect(self.run_benchmark)
    layout.addWidget(self.btn_benchmark)

    # Лейбл для часу виконання
    self.lbl_execution_time = QLabel("Час виконання: -")
    self.lbl_execution_time.setStyleSheet("color: #00ffff; font-weight:
bold; margin-top: 5px;")
    self.lbl_execution_time.setAlignment(Qt.AlignRight)
    layout.addWidget(self.lbl_execution_time)

    layout.addSpacing(5)
    layout.addWidget(QLabel("Результат (Base64):"))
    self.result_crypto_enc = QTextEdit()

```

```

        self.result_crypto_enc.setReadOnly(True)
        self.result_crypto_enc.setFixedHeight(60)
        self.result_crypto_enc.setStyleSheet("color: #00ff00; font-family:
Consolas;")
        layout.addWidget(self.result_crypto_enc)

    def init_crypto_dec_ui(self):
        layout = QVBoxLayout()
        self.tab_crypto_dec.setLayout(layout)
        layout.addSpacing(20)

        layout.addWidget(QLabel("1. Зашифрований текст (Base64):"))
        self.input_crypto_dec = QTextEdit()
        self.input_crypto_dec.setPlaceholderText("Вставте шифротекст...")
        layout.addWidget(self.input_crypto_dec)

        layout.addSpacing(15)
        layout.addWidget(QLabel("2. Пароль:"))
        self.pass_input_crypto_dec = QLineEdit()
        self.pass_input_crypto_dec.setPlaceholderText("••••••••")
        self.pass_input_crypto_dec.setEchoMode(QLineEdit.Password)
        layout.addWidget(self.pass_input_crypto_dec)

        layout.addSpacing(20)
        self.btn_run_crypto_dec = QPushButton("🔓 РОЗШИФРУВАТИ")
        self.btn_run_crypto_dec.setStyleSheet("background-color: #e0a800;
color: #212529; padding: 12px;")
        self.btn_run_crypto_dec.clicked.connect(self.process_crypto_decrypt)
        layout.addWidget(self.btn_run_crypto_dec)

        layout.addSpacing(15)
        layout.addWidget(QLabel("Результат:"))
        self.result_crypto_dec = QTextEdit()
        self.result_crypto_dec.setReadOnly(True)
        self.result_crypto_dec.setFixedHeight(80)
        self.result_crypto_dec.setStyleSheet("color: #00ff00; font-family:
Consolas;")
        layout.addWidget(self.result_crypto_dec)

    # --- ЛОГІКА ДЛЯ КРИПТО-ЛАБОРАТОРІЇ (Поки що тільки AES) ---
    def process_crypto_encrypt(self):
        text = self.text_input_crypto.toPlainText()
        password = self.pass_input_crypto.text()
        algo_selection = self.combo_algo.currentText()
        algo_code = "ChaCha20" if "ChaCha20" in algo_selection else "AES"

        if not text or not password:
            QMessageBox.warning(self, "Помилка", "Введіть текст і пароль!")
            return

        try:
            # ЗАМІР ЧАСУ: СТАРТ
            start_time = time.perf_counter()

            encrypted_bytes = self.crypto.encrypt(text.encode('utf-8'),
password, algorithm=algo_code)

            # ЗАМІР ЧАСУ: СТОП
            end_time = time.perf_counter()

```

```

        duration = end_time - start_time

        # Вивід часу
        self.lbl_execution_time.setText(f"Час виконання: {duration:.5f}
сек")

        import base64
        b64_str = base64.b64encode(encrypted_bytes).decode('utf-8')
        self.result_crypto_enc.setText(b64_str)
        self.status_label_crypto.setText(f"Зашифровано ({algo_code}) ✓")

    except Exception as e:
        QMessageBox.critical(self, "Error", str(e))

    def process_crypto_decrypt(self):
        b64_text = self.input_crypto_dec.toPlainText()
        password = self.pass_input_crypto_dec.text()

        if not b64_text or not password:
            QMessageBox.warning(self, "Помилка", "Введіть шифротекст і
пароль!")
            return

        try:
            import base64
            encrypted_bytes = base64.b64decode(b64_text)

            decrypted_bytes = self.crypto.decrypt(encrypted_bytes, password)

            if decrypted_bytes:
                self.result_crypto_dec.setText(decrypted_bytes.decode('utf-8'))
                self.status_label_crypto.setText("Розшифровано ✓")
            else:
                self.result_crypto_dec.setText("ПОМИЛКА ПАРОЛЯ")
                self.status_label_crypto.setText("Невірний пароль X")

        except Exception as e:
            # Це трапляється якщо base64 кривий
            QMessageBox.critical(self, "Error", "Невірний формат шифротексту")

    def run_benchmark(self):
        """
        Генерує 200 МВ випадкових даних і порівнює швидкість AES vs ChaCha20.
        """
        reply = QMessageBox.question(self, "Бенчмарк",
            "Це створить 200 МВ даних у пам'яті і
протестує обидва алгоритми.\n"
            "Програма може 'завмерти' на 5-10
секунд.\nПродовжити?",
            QMessageBox.Yes | QMessageBox.No)

        if reply == QMessageBox.No:
            return

        try:
            self.status_label_crypto.setText("Генерація даних (200 МВ)...")
            self.lbl_execution_time.setText("Тестування...")
            QApplication.processEvents()

```

```

# 1. Генерація сміттєвих даних (200 MB)
data_size = 200 * 1024 * 1024 # <--- ТУТ ТЕПЕР 200 MB
test_data = os.urandom(data_size)
password = "BenchmarkPassword123"

# 2. Тест AES-256
self.status_label_crypto.setText("Тестування AES-256...")
QApplication.processEvents()

start_aes = time.perf_counter()
self.crypto.encrypt(test_data, password, algorithm="AES")
end_aes = time.perf_counter()
time_aes = end_aes - start_aes

# 3. Тест ChaCha20
self.status_label_crypto.setText("Тестування ChaCha20...")
QApplication.processEvents()

start_cha = time.perf_counter()
self.crypto.encrypt(test_data, password, algorithm="ChaCha20")
end_cha = time.perf_counter()
time_cha = end_cha - start_cha

# 4. Висновок
result_msg = (
    f"📊 РЕЗУЛЬТАТИ ТЕСТУ (200 MB):\n\n"
    f"💎 AES-256: {time_aes:.4f} сек\n"
    f"💎 ChaCha20: {time_cha:.4f} сек\n\n"
)

if time_cha < time_aes:
    diff = (time_aes / time_cha)
    result_msg += f"🚀 ChaCha20 швидше в {diff:.2f} разів!"
else:
    diff = (time_cha / time_aes)
    result_msg += f"🚀 AES-256 швидше в {diff:.2f} разів (апаратне
прискорення?)"

self.result_crypto_enc.setText(result_msg)
self.lbl_execution_time.setText("Тест завершено ✓")
self.status_label_crypto.setText("Бенчмарк готовий")

QMessageBox.information(self, "Бенчмарк завершено", result_msg)

except Exception as e:
    QMessageBox.critical(self, "Помилка бенчмарку", str(e))

# --- ДОПОМІЖНІ ФУНКЦІЇ ФОТО/АУДІО ---
def select_source_image(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Оберіть картинку",
"", "Images (*.png)")
    if file_path:
        self.source_image_path = file_path
        name = os.path.basename(file_path)
        self.btn_select_img.setText(f"✓ {name}")
        self.btn_select_img.setStyleSheet(

```

```

        "background-color: #2c3e50; border: 1px solid #28a745; color:
#28a745; font-weight: bold;")

        self.btn_clear_img.show() # <--- ПОКАЗУЄМО КНОПКУ ОЧИСТКИ

        self.btn_run_photo_embed.setEnabled(True)
        self.update_capacity_bar()

def update_capacity_bar(self):
    """Розраховує, скільки місця зайнято в картинці (сума msg1 + msg2)"""
    if not self.source_image_path:
        return

    try:
        # 1. Рахуємо максимальну ємність картинки
        with Image.open(self.source_image_path) as img:
            width, height = img.size

        # (Ширина * Висота * 3 канали) / 8 біт = кількість байтів
        max_bytes = (width * height * 3) // 8
        max_bytes -= 4 # Резерв під заголовок

        # 2. Рахуємо розмір ПЕРШОГО повідомлення
        current_bytes = 0

        if self.radio_text.isChecked():
            text = self.text_input_photo.text()
            current_bytes += len(text.encode('utf-8'))
        elif self.secret_file_path:
            if os.path.exists(self.secret_file_path):
                current_bytes += os.path.getsize(self.secret_file_path)

        # 3. Рахуємо розмір ДРУГОГО повідомлення (НОВЕ!)
        # Перевіряємо, чи увімкнено подвійне дно
        if hasattr(self, 'chk_double_stego') and
self.chk_double_stego.isChecked():
            text2 = self.text_input_photo_2.text()
            current_bytes += len(text2.encode('utf-8'))

        # 4. Оновлюємо прогрес-бар
        percent = 0
        if max_bytes > 0:
            percent = int((current_bytes / max_bytes) * 100)

        self.capacity_bar.setValue(percent)
        self.lbl_capacity_info.setText(f"{current_bytes} / {max_bytes}
байт")

        # Зміна кольору
        if percent < 50:
            color = "#28a745" # Зелений
        elif percent < 90:
            color = "#ffc107" # Жовтий
        else:
            color = "#dc3545" # Червоний

        self.capacity_bar.setStyleSheet(f"""
QProgressBar {{
    border: 1px solid #555;

```

```

        border-radius: 5px;
        text-align: center;
        background-color: #2b2b2b;
        color: white;
    }}
    QProgressBar::chunk {{
        background-color: {color};
    }}
    """)

    # Блокування кнопки при переповненні
    if percent > 100:
        self.btn_run_photo_embed.setEnabled(False)
        self.lbl_capacity_info.setStyleSheet("color: #dc3545; font-
weight: bold;")
        self.lbl_capacity_info.setText(f"ПЕРЕПОВНЕНО! {current_bytes} /
{max_bytes} байт")
    else:
        # Кнопка активна тільки якщо вибрана картинка
        self.btn_run_photo_embed.setEnabled(True if
self.source_image_path else False)
        self.lbl_capacity_info.setStyleSheet("color: #aaa;")

    except Exception as e:
        print(f"Capacity Error: {e}")

    def clear_photo_selection(self):
        """Очищає вибір головної картинки"""
        self.source_image_path = None
        self.btn_select_img.setText("📁 Оберіть файл зображення (PNG)")
        self.btn_select_img.setStyleSheet("background-color: #3c3f41; border:
1px dashed #666; color: #aaa;")
        self.btn_clear_img.hide() # Ховаємо хрестик
        self.btn_run_photo_embed.setEnabled(False)
        self.update_capacity_bar() # Скидаємо індикатор

    def clear_audio_selection(self):
        """Очищає вибір аудіофайлу"""
        self.source_audio_path = None
        self.btn_select_audio.setText("🎵 Оберіть файл аудіо (WAV)")
        self.btn_select_audio.setStyleSheet("background-color: #3c3f41; border:
1px dashed #666; color: #aaa;")
        self.btn_clear_audio.hide()
        self.btn_run_audio_embed.setEnabled(False)

    def clear_secret_file(self):
        """Очищає вибір секретного файлу"""
        self.secret_file_path = None
        self.lbl_secret_file.setText("Файл не обрано")
        self.lbl_secret_file.setStyleSheet("font-size: 10px; color: #aaa;")
        self.btn_clear_file.hide()
        self.update_capacity_bar()

    def select_stego_image(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "Оберіть картинку",
"", "Images (*.png)")
        if file_path:
            self.stego_image_path = file_path

```

```

        name = os.path.basename(file_path)
        self.btn_select_stego_img.setText(f"✓ {name}")
        self.btn_select_stego_img.setStyleSheet(
            "background-color: #2c3e50; border: 1px solid #ffc107; color:
#ffc107; font-weight: bold;")

        self.btn_clear_stego_img.show() # <--- ПОКАЗУЄМО ХРЕСТИК

        self.btn_run_photo_extract.setEnabled(True)

    def clear_stego_image_selection(self):
        """Очищає вибір стего-картинки"""
        self.stego_image_path = None
        self.btn_select_stego_img.setText("📁 Оберіть зображення з секретом")
        self.btn_select_stego_img.setStyleSheet("background-color: #3c3f41;
border: 1px dashed #666; color: #aaa;")
        self.btn_clear_stego_img.hide()
        self.btn_run_photo_extract.setEnabled(False)
        self.result_display_photo.clear() # Очищаємо поле результату

    def toggle_input_mode(self):
        """Перемикає видимість полів і блокує подвійне дно для файлів"""
        if self.radio_text.isChecked():
            # Режим ТЕКСТ
            self.container_text.show()
            self.container_file.hide()

            # Дозволяємо подвійне дно
            self.chk_double_stego.setEnabled(True)
            self.chk_double_stego.setStyleSheet("color: #ffa500; font-weight:
bold;")
        else:
            # Режим ФАЙЛ
            self.container_text.hide()
            self.container_file.show()

            # Блокуємо і вимикаємо подвійне дно (бо файли завеликі для цього)
            self.chk_double_stego.setChecked(False)
            self.chk_double_stego.setEnabled(False)
            self.chk_double_stego.setStyleSheet("color: #555; font-weight:
bold;") # Робимо сірим

    def toggle_double_stego_ui(self, checked):
        """Показує або ховає поля для другого повідомлення"""
        if checked:
            self.group_msg2.show()
        else:
            self.group_msg2.hide()
            self.text_input_photo_2.clear()
            self.pass_input_photo_enc_2.clear()

    def select_secret_file(self):
        path, _ = QFileDialog.getOpenFileName(self, "Оберіть файл", "", "All
Files (*.*)")
        if path:
            self.secret_file_path = path
            name = os.path.basename(path)
            self.lbl_secret_file.setText(f"Обрано: {name}")

```

```

self.lbl_secret_file.setStyleSheet("color: #00ff00;")

self.btn_clear_file.show() # <--- ПОКАЗУЄМО КНОПКУ ОЧИСТКИ

self.update_capacity_bar()

def process_embed_photo(self):
    # 1. Збір даних для ПЕРШОГО повідомлення
    pass1 = self.pass_input_photo_enc.text()
    if not pass1:
        QMessageBox.warning(self, "Помилка", "Введіть пароль для
повідомлення №1!")
        return

    data1 = None
    if self.radio_text.isChecked():
        txt = self.text_input_photo.text()
        if not txt:
            QMessageBox.warning(self, "Помилка", "Введіть текст
повідомлення №1!")
            return
        data1 = txt.encode('utf-8')
    else:
        if not self.secret_file_path:
            QMessageBox.warning(self, "Помилка", "Оберіть файл для
повідомлення №1!")
            return
        try:
            with open(self.secret_file_path, 'rb') as f:
                data1 = f.read()
        except Exception as e:
            QMessageBox.critical(self, "Error", f"Помилка файлу: {e}")
            return

    # 2. Збір даних для ДРУГОГО повідомлення (якщо треба)
    is_double = self.chk_double_stego.isChecked()
    data2 = None
    pass2 = None

    if is_double:
        txt2 = self.text_input_photo_2.text()
        pass2 = self.pass_input_photo_enc_2.text()

        if not txt2 or not pass2:
            QMessageBox.warning(self, "Помилка", "Для подвійного дна
заповніть повідомлення №2 та його пароль!")
            return

        if pass1 == pass2:
            QMessageBox.warning(self, "Помилка безпеки", "Паролі повинні
бути різними!")
            return

        data2 = txt2.encode('utf-8')

    # 3. Вибір куди зберігати
    save_path, _ = QFileDialog.getSaveFileName(self, "Зберегти результат
як...", "secret_image.png",
                                                "Images (*.png)")

```

```

        if not save_path:
            return

# 4. ПРОЦЕС ШИФРУВАННЯ
try:
    self.status_label_photo.setText("Обробка...")
    QApplication.processEvents()

    # ЕТАП А: Шифруємо і ховаємо перше повідомлення
    enc_data1 = self.crypto.encrypt(data1, pass1)

    # Ховаємо msg1 у source -> save_path
    success1 = self.stego.embed(self.source_image_path, enc_data1,
save_path, pass1)

    if not success1:
        QMessageBox.critical(self, "Помилка", "Повідомлення №1 завелике
або сталася помилка.")
        self.status_label_photo.setText("Помилка X")
        return

    # ЕТАП Б: Якщо є друге повідомлення
    if is_double and data2 and pass2:
        self.status_label_photo.setText("Додавання другого дна...")
        QApplication.processEvents()

        enc_data2 = self.crypto.encrypt(data2, pass2)

        # УВАГА: Тепер джерелом є вже збережений save_path, і пишемо ми
знову в save_path
        success2 = self.stego.embed(save_path, enc_data2, save_path,
pass2)

        if not success2:
            QMessageBox.warning(self, "Увага",
                                "Повідомлення №1 збережено, але №2 не
влізло (колізія або розмір).")

            self.status_label_photo.setText("Готово ✓")
            QMessageBox.information(self, "Успіх", f"Дані успішно сховано у
файл:\n{save_path}")

            # Очистка полів для безпеки
            self.text_input_photo.clear()
            self.pass_input_photo_enc.clear()
            if is_double:
                self.text_input_photo_2.clear()
                self.pass_input_photo_enc_2.clear()

except Exception as e:
    QMessageBox.critical(self, "Error", str(e))

def process_extract_photo(self):
    password = self.pass_input_photo_dec.text()
    if not password:
        QMessageBox.warning(self, "Помилка", "Введіть пароль!")
        return
    try:

```

```

self.status_label_photo.setText("Аналіз...")
QApplication.processEvents()

# Витягуємо (БУЛО)
# extracted_data = self.stego.extract(self.stego_image_path)

# Витягуємо (СТАЛО - передаємо password!)
extracted_data = self.stego.extract(self.stego_image_path,
password)
if not extracted_data:
    QMessageBox.warning(self, "Невдача", "Контейнер порожній.")
    self.status_label_photo.setText("Пусто X")
    return

decrypted_data = self.crypto.decrypt(extracted_data, password)
if decrypted_data:
    self.status_label_photo.setText("Успішно ✓")

    # Питаємо користувача, що робити з даними
    msg_box = QMessageBox()
    msg_box.setWindowTitle("Дані знайдено!")
    msg_box.setText(f"Розшифровано {len(decrypted_data)} байтів.")
    msg_box.setInformativeText("Це текст чи файл?")
    btn_text = msg_box.addButton("Це Текст (Показати)",
QMessageBox.ActionRole)
    btn_file = msg_box.addButton("Це Файл (Зберегти)",
QMessageBox.ActionRole)
    msg_box.exec()

    if msg_box.clickedButton() == btn_text:
        try:
            text_msg = decrypted_data.decode('utf-8')
            self.result_display_photo.setText(text_msg)
        except:
            QMessageBox.warning(self, "Помилка", "Це не схоже на
текст! Краще збережіть як файл.")
        else:
            # Зберігаємо файл
            save_path, _ = QFileDialog.getSaveFileName(self, "Зберегти
розшифрований файл", "extracted_file",
"*.*)")
            if save_path:
                with open(save_path, 'wb') as f:
                    f.write(decrypted_data)
                QMessageBox.information(self, "Успіх", "Файл
збережено!")
            else:
                self.result_display_photo.setText("ПОМИЛКА ПАРОЛЯ")
                self.status_label_photo.setText("Невірний пароль X")
                QMessageBox.warning(self, "Помилка", "Невірний пароль!")
        except Exception as e:
            QMessageBox.critical(self, "Error", str(e))

# Аудіо методи
def select_source_audio(self):

```

```

        file_path, _ = QFileDialog.getOpenFileName(self, "Оберіть аудіо", "",
"Audio (*.wav)")
        if file_path:
            self.source_audio_path = file_path
            name = os.path.basename(file_path)
            self.btn_select_audio.setText(f"✓ {name}")
            self.btn_select_audio.setStyleSheet(
                "background-color: #2c3e50; border: 1px solid #6f42c1; color:
#6f42c1; font-weight: bold;")

            self.btn_clear_audio.show() # <--- ПОКАЗУЄМО ХРЕСТИК

            self.btn_run_audio_embed.setEnabled(True)

    def select_stego_audio(self):
        file_path, _ = QFileDialog.getOpenFileName(self, "Оберіть аудіо", "",
"Audio (*.wav)")
        if file_path:
            self.stego_audio_path = file_path
            name = os.path.basename(file_path)
            self.btn_select_stego_audio.setText(f"✓ {name}")
            self.btn_select_stego_audio.setStyleSheet(
                "background-color: #2c3e50; border: 1px solid #ffc107; color:
#ffc107; font-weight: bold;")

            self.btn_clear_stego_audio.show() # <--- ПОКАЗУЄМО ХРЕСТИК

            self.btn_run_audio_extract.setEnabled(True)

    def clear_stego_audio_selection(self):
        """Очищає вибір стего-аудіо"""
        self.stego_audio_path = None
        self.btn_select_stego_audio.setText("🎵 Оберіть аудіо з секретом")
        self.btn_select_stego_audio.setStyleSheet("background-color: #3c3f41;
border: 1px dashed #666; color: #aaa;")
        self.btn_clear_stego_audio.hide()
        self.btn_run_audio_extract.setEnabled(False)
        self.result_display_audio.clear()

    def process_embed_audio(self):
        text = self.text_input_audio.text()
        password = self.pass_input_audio_enc.text()
        if not text or not password:
            QMessageBox.warning(self, "Помилка", "Введіть текст і пароль!")
            return
        save_path, _ = QFileDialog.getSaveFileName(self, "Зберегти як...",
"secret_audio.wav", "Audio (*.wav)")
        if not save_path: return
        try:
            self.status_label_audio.setText("Шифрування...")
            encrypted_data = self.crypto.encrypt(text.encode('utf-8'),
password)
            success = self.audio_stego.embed(self.source_audio_path,
encrypted_data, save_path)
            if success:
                QMessageBox.information(self, "Успіх", f"Аудіо
збережено:\n{save_path}")
                self.status_label_audio.setText("Готово ✓")

```

```

        self.text_input_audio.clear()
    else:
        QMessageBox.critical(self, "Помилка", "Помилка вбудовування.")
        self.status_label_audio.setText("Помилка X")
except Exception as e:
    QMessageBox.critical(self, "Error", str(e))

def process_extract_audio(self):
    password = self.pass_input_audio_dec.text()
    if not password:
        QMessageBox.warning(self, "Помилка", "Введіть пароль!")
        return
    try:
        self.status_label_audio.setText("Аналіз...")
        extracted_data = self.audio_stego.extract(self.stego_audio_path)
        if not extracted_data:
            QMessageBox.warning(self, "Невдача", "У звуці нічого не
знайдено.")
            self.status_label_audio.setText("Пусто X")
            return
        decrypted_data = self.crypto.decrypt(extracted_data, password)
        if decrypted_data:
            self.result_display_audio.setText(decrypted_data.decode('utf-
8'))
            self.status_label_audio.setText("Успішно ✓")
            QMessageBox.information(self, "Успіх", "Повідомлення
знайдено!")
        else:
            self.result_display_audio.setText("ПОМИЛКА")
            self.status_label_audio.setText("Невірний пароль X")
    except Exception as e:
        QMessageBox.critical(self, "Error", str(e))

```

Лістинг коду stego_audio.py

```

import wave
import struct
import os

class AudioStego:
    """
    Клас для приховування даних у аудіофайлах формату WAV.
    Використовує метод LSB (Least Significant Bit) для байтів аудіоданих.
    """

    def __init__(self):
        # 4 байти (32 біти) для запису розміру повідомлення
        self.header_size = 4

    def embed(self, audio_path: str, data: bytes, output_path: str) -> bool:
        """
        Вбудовує байти (data) у .wav файл.
        """
        try:
            # 1. Відкриваємо аудіофайл
            song = wave.open(audio_path, mode='rb')

            # Читаємо параметри (кількість кадрів, частота)
            frame_bytes = bytearray(list(song.readframes(song.getnframes())))
            params = song.getparams() # Зберігаємо параметри, щоб відтворити
            файл потім
            song.close()

            # 2. Підготовка даних
            data_len = len(data)
            # Додаємо розмір даних на початок
            full_data = struct.pack('>I', data_len) + data

            # Перетворюємо дані в біти
            bits = []
            for byte in full_data:
                for i in range(8):
                    bits.append((byte >> (7 - i)) & 1)

```

```

        # Перевірка ємності
        if len(bits) > len(frame_bytes):
            raise ValueError(f"Аудіофайл замалий! Потрібно {len(bits)}
байтів, є {len(frame_bytes)}.")

        # 3. Вбудовування (LSB)
        # Ми змінюємо останній біт кожного байта аудіоданих
        for i, bit in enumerate(bits):
            # frame_bytes[i] & 254 -> занулюємо останній біт (11111110)
            # | bit -> вставляємо наш біт (0 або 1)
            frame_bytes[i] = (frame_bytes[i] & 254) | bit

        # 4. Збереження нового файлу
        modified_frames = bytes(frame_bytes)
        with wave.open(output_path, 'wb') as fd:
            fd.setparams(params)
            fd.writeframes(modified_frames)

        return True

    except Exception as e:
        print(f"Помилка Audio Stego: {e}")
        return False

def extract(self, audio_path: str) -> bytes:
    """
    Витягує дані з .wav файлу.
    """
    try:
        song = wave.open(audio_path, mode='rb')
        frame_bytes = bytearray(list(song.readframes(song.getnframes())))
        song.close()

        # 1. Витягуємо LSB з кожного байта
        extracted_bits = [frame_bytes[i] & 1 for i in
range(len(frame_bytes))]

        # 2. Читаємо заголовок (перші 32 біти = розмір)
        header_bits = extracted_bits[:self.header_size * 8]

        # Перетворюємо біти заголовка в число
        header_bytes_val = 0
        for bit in header_bits:
            header_bytes_val = (header_bytes_val << 1) | bit

        data_len = header_bytes_val

        # Перевірка на адекватність
        if data_len > (len(extracted_bits) - 32) // 8:
            return None

        # 3. Читаємо саме повідомлення
        start = self.header_size * 8
        end = start + (data_len * 8)
        data_bits = extracted_bits[start:end]

        # Перетворюємо біти назад у байти
        data_bytes = bytearray()
        for i in range(0, len(data_bits), 8):

```

```

        byte_val = 0
        for bit in data_bits[i:i + 8]:
            byte_val = (byte_val << 1) | bit
        data_bytes.append(byte_val)

    return bytes(data_bytes)

except Exception as e:
    print(f"Помилка Audio Extract: {e}")
    return None

```

Додаток Г.

Лістинг коду crypto.py

```

import os
import zlib
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives.ciphers.aead import ChaCha20Poly1305

class CryptoHandler:
    """
    Клас для обробки шифрування.
    Підтримує:
    1. AES-256 (CTR mode) - Класичний стандарт.
    2. ChaCha20-Poly1305 - Сучасний потоковий шифр (швидший на мобільних).
    """

    def __init__(self):
        self.backend = default_backend()
        self.salt_size = 16

    def _derive_key(self, password: str, salt: bytes, length=32) -> bytes:
        """Генерує ключ із пароля"""
        kdf = PBKDF2HMAC(
            algorithm=hashes.SHA256(),
            length=length,
            salt=salt,
            iterations=100000,
            backend=self.backend
        )
        return kdf.derive(password.encode('utf-8'))

    def encrypt(self, data: bytes, password: str, algorithm="AES") -> bytes:
        """
        Шифрує дані обраним алгоритмом.
        algorithm: "AES" або "ChaCha20"
        """
        # 1. Стиснення
        compressed_data = zlib.compress(data, level=9)

```

```

# 2. Генерація солі
salt = os.urandom(self.salt_size)
key = self._derive_key(password, salt)

if algorithm == "ChaCha20":
    # ChaCha20 вимагає Nonce (12 байт)
    nonce = os.urandom(12)
    cipher = ChaCha20Poly1305(key)
    ciphertext = cipher.encrypt(nonce, compressed_data, None)

    # Маркер для розпізнавання при дешифруванні (1 байт = 2)
    algo_marker = b'\x02'
    return algo_marker + salt + nonce + ciphertext

else:
    # AES-256 CTR (за замовчуванням)
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CTR(iv),
backend=self.backend)
    encryptor = cipher.encryptor()
    ciphertext = encryptor.update(compressed_data) +
encryptor.finalize()

    # Маркер для AES (1 байт = 1)
    algo_marker = b'\x01'
    return algo_marker + salt + iv + ciphertext

def decrypt(self, encrypted_container: bytes, password: str) -> bytes:
    """
    Автоматично визначає алгоритм (за першим байтом) і дешифрує.
    """
    try:
        # Читаємо маркер алгоритму (перший байт)
        algo_marker = encrypted_container[0:1]
        data_body = encrypted_container[1:]

        salt = data_body[:self.salt_size]
        remaining = data_body[self.salt_size:]

        key = self._derive_key(password, salt)

        decrypted_compressed = None

        if algo_marker == b'\x02': # Це ChaCha20
            nonce = remaining[:12]
            ciphertext = remaining[12:]

            cipher = ChaCha20Poly1305(key)
            decrypted_compressed = cipher.decrypt(nonce, ciphertext, None)

        elif algo_marker == b'\x01': # Це AES
            iv = remaining[:16]
            ciphertext = remaining[16:]

            cipher = Cipher(algorithms.AES(key), modes.CTR(iv),
backend=self.backend)
            decryptor = cipher.decryptor()
            decrypted_compressed = decryptor.update(ciphertext) +
decryptor.finalize()

```

```

        else:
            raise ValueError("Невідомий формат шифрування!")

        # Розтискання
        return zlib.decompress(decrypted_compressed)

    except Exception as e:
        print(f"Помилка дешифрування: {e}")
        return None

```

Додаток Д.

Лістинг коду stego.py

```

import hashlib
import random
from PIL import Image

class StegoHandler:
    """
    Клас для стеганографії в зображеннях.
    Підтримує 'Правдоподібне заперечення' (Double Steganography)
    за рахунок вибору пікселів на основі хешу пароля.
    """

    def __init__(self):
        # 4 байти (32 біти) для запису розміру повідомлення
        self.header_size = 4

    def _get_indices_generator(self, width, height, password):
        """
        Генерує послідовність унікальних координат пікселів (індексів)
        на основі пароля. Це гарантує, що різні паролі дадуть різні пікселі.
        """
        total_pixels = width * height * 3 # 3 канали (R, G, B)

        # 1. Створюємо seed з пароля
        hash_obj = hashlib.sha256(password.encode('utf-8'))
        seed_val = int(hash_obj.hexdigest(), 16)

        # 2. Ініціалізуємо генератор
        rng = random.Random(seed_val)

        # 3. Генеруємо координати.
        # Для оптимізації ми не робимо список з мільйонів чисел,
        # а використовуємо set для перевірки унікальності.
        # Це працює швидко для текстових повідомлень.
        used_indices = set()

        while True:
            # Вибираємо випадковий індекс
            idx = rng.randint(0, total_pixels - 1)

```

```

        # Якщо ми вже писали в цей піксель - пропускаємо (шукаємо інший)
        if idx not in used_indices:
            used_indices.add(idx)
            yield idx

def embed(self, image_path, data, output_path, password):
    """
    Вбудовує дані в зображення, використовуючи пароль для вибору позицій.
    """
    try:
        img = Image.open(image_path)
        width, height = img.size
        pixels = img.load() # Матриця пікселів [x, y] = (r, g, b)

        # 1. Підготовка даних (Розмір + Самі дані)
        data_len = len(data)
        full_data = data_len.to_bytes(self.header_size, byteorder='big') +
data

        # Перетворюємо байти в біти (0 та 1)
        bits = []
        for byte in full_data:
            for i in range(8):
                bits.append((byte >> (7 - i)) & 1)

        required_bits = len(bits)
        total_capacity = width * height * 3

        if required_bits > total_capacity:
            print("Помилка: Дані занадто великі для цього зображення!")
            return False

        # 2. Отримуємо генератор випадкових позицій
        indices_gen = self._get_indices_generator(width, height, password)

        # 3. Записуємо біти
        # Нам треба знати, як мапити лінійний індекс (0..N) на (x, y,
channel)
        for bit_to_write in bits:
            idx = next(indices_gen)

            # Конвертуємо лінійний індекс у (x, y, канал)
            # idx // 3 -> номер пікселя
            # idx % 3 -> номер каналу (0=R, 1=G, 2=B)
            pixel_idx = idx // 3
            channel = idx % 3

            y = pixel_idx // width
            x = pixel_idx % width

            # Отримуємо поточний піксель (r, g, b)
            r, g, b = pixels[x, y]
            channels = [r, g, b]

            # Змінюємо потрібний канал (LSB)
            # (val & ~1) -> занулює останній біт
            # | bit_to_write -> записує наш біт
            channels[channel] = (channels[channel] & ~1) | bit_to_write

```

```

        # Оновлюємо піксель
        pixels[x, y] = tuple(channels)

    # 4. Зберігаємо
    img.save(output_path)
    return True

except Exception as e:
    print(f"Помилка Embed: {e}")
    return False

def extract(self, image_path, password):
    """
    Витягує дані, знаючи пароль (і відповідно позиції пікселів).
    """
    try:
        img = Image.open(image_path)
        width, height = img.size
        pixels = img.load()

        indices_gen = self._get_indices_generator(width, height, password)

        def read_bits(count):
            """Читає count бітів з позицій, визначених генератором"""
            read_data_bits = []
            for _ in range(count):
                idx = next(indices_gen)

                pixel_idx = idx // 3
                channel = idx % 3
                y = pixel_idx // width
                x = pixel_idx % width

                val = pixels[x, y][channel]
                read_data_bits.append(val & 1)
            return read_data_bits

        def bits_to_bytes(bits_list):
            """Перетворює список бітів у байти"""
            bytes_list = bytearray()
            for i in range(0, len(bits_list), 8):
                byte_val = 0
                for bit in bits_list[i:i + 8]:
                    byte_val = (byte_val << 1) | bit
                bytes_list.append(byte_val)
            return bytes_list

        # 1. Читаємо заголовок (розмір) - перші 32 біти
        header_bits = read_bits(self.header_size * 8)
        header_bytes = bits_to_bytes(header_bits)
        data_len = int.from_bytes(header_bytes, byteorder='big')

        # Санітарна перевірка: якщо розмір неадекватний, значить пароль не
        той

        # (бо ми прочитали випадковий шум замість розміру)
        if data_len <= 0 or data_len > (width * height * 3) // 8:
            return None

        # 2. Читаємо саме повідомлення

```

```
msg_bits = read_bits(data_len * 8)
msg_bytes = bits_to_bytes(msg_bits)

return bytes(msg_bytes)

except Exception as e:
    print(f"Помилка Extract: {e}")
    return None
```