Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В. о. завідувач кафедри
кібербезпеки та захисту
інформації
_____ Іван ПАРХОМЕНКО
«13» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)
спеціальність 125 Кібербезпека
(код і назва спеціальності)
освітній ступень бакалавр
освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Стеганографічний метод приховування інформації в електронних книгах формату FB2»

Виконавець: студент IV курсу, групи КБ-42

_____ **Андрій ШОКАЛО**
(підпис) (ім'я прізвище)

	Підпис	**Ім'я, прізвище**
Керівник		Юрій БАБЕНКО
Нормоконтроль		Інна МИХАЛЬЧУК

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В. о. завідувач кафедри
кібербезпеки та захисту
інформації
_____ Іван
ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої-професійної програми)

Студенту **КБ-42** (група) **Шокало Андрію Романовичу** (прізвище ім'я по батькові)

Тема кваліфікаційної роботи Стеганографічний метод приховування інформації в електронних книгах формату FB2.

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Структура форматованих електронних книжок FictionBook 2.0 (FB2), принципи стеганографічних методів модифікації зображень та невідображуваних полів, засоби обробки XML-файлів та бібліотеки для роботи з документами і зображеннями.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Огляд методів цифрової та комп'ютерної стеганографії, формалізація узагальненої моделі стегосистеми, опис форматних можливостей приховування даних у структурі FB2, розробка алгоритмів вбудовування та витягування інформації, реалізація програмного рішення

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Програмне рішення, яке спрямоване на вбудовування прихованої інформації в е-книжки формату FB2.

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: 29 листопада 2024 року

Завдання видав		Юрій БАБЕНКО
	(підпис)	(ім'я, прізвище)
Завдання прийняв до виконання		Андрій ШОКАЛО
	(підпис)	(ім'я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 18.12.2024	*виконано*
2	Аналіз літератури	19.12.2024 – 25.12.2024	*виконано*
3	Обґрунтування вибору рішення	27.12.2024 – 04.01.2025	*виконано*
4	Аналіз застосування стеганографічних методів та їх класифікація	05.01.2025 – 16.01.2025	*виконано*
5	Дослідження графічних методів приховування інформації	30.01.2025 – 06.02.2025	*виконано*
6	Дослідження текстових методів приховування інформації	07.01.2025 – 14.02.2025	*виконано*
7	Аналіз та адаптація методу найменш значущих бітів до контексту текстової стеганографії	15.02.2025 – 21.02.2025	*виконано*
8	Вивчення структури FB2-файлів як потенційного стеганографічного контейнера	01.03.2025 – 14.03.2025	*виконано*
9	Розробка та обґрунтування методу приховування даних у FictionBook 2.0	17.03.2025 – 11.04.2025	*виконано*
10	Реалізація програмного засобу для демонстрації працездатності розробленого методу	12.04.2025 – 16.05.2025	*виконано*

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
11	Оформлення пояснювальної записки	12.05.2025 – 21.05.2025	*виконано*
12	Підготовка до захисту кваліфікаційної роботи	22.05.2025 – 13.06.2025	*виконано*

Завдання видав _____ Юрій БАБЕНКО
(підпис) (ім’я, прізвище)

Завдання прийняв до виконання _____ Андрій ШОКАЛО
(підпис) (ім’я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, має 75 сторінок основного тексту та 40 рисунків. Крім того, робота містить 3 додатки із загальною кількістю сторінок 8. Список використаних джерел містить 30 найменувань та займає 4 сторінки.

Метою кваліфікаційної роботи є дослідження та впровадження методів вбудовування прихованої інформації в електронні книги формату FB2.

Для досягнення поставленої мети були визначені такі завдання:

- проаналізувати предметну область і структуру FB2-документів як об’єкта прихованого вбудовування;
- обґрунтувати вибір стеганографічних алгоритмів для XML-структур і растрових зображень;
- побудувати алгоритми вбудовування й витягування інформації з елементів <binary> та <custom-info>;
- реалізувати програмний прототип для автоматизованої обробки FB2-файлів;
- протестувати модуль щодо збереження цілісності файлу та коректності витягування прихованих даних.

Об’єктом дослідження є електронні книги формату FB2 як цифрові носії, потенційно прихованої інформації.

Предметом дослідження є методика та програмна реалізація вбудовування прихованої інформації в структурні елементи електронних книг формату FB2.

Практичною цінністю є програмне рішення, яке спрямоване на вбудовування прихованої інформації із використанням стеганографічного алгоритму LSB та використання невідображуваних полів.

Ключові слова: цифрова стеганографія, електронні книги, FB2, прихована інформація, алгоритм LSB.

ЗМІСТ

ВСТУП

У сучасних умовах стрімкого розвитку цифрових технологій усе більшу актуальність набувають питання захисту інформації, зокрема в частині прихованого передавання даних та ідентифікації джерел витоку інформації. На тлі поширення електронного контенту особливої уваги заслуговує напрям цифрової стеганографії, який дозволяє забезпечувати конфіденційний обмін повідомленнями без виявлення самого факту передавання інформації.

Одним із перспективних, проте недостатньо досліджених, носіїв прихованої інформації є електронні книги у форматі FB2 (FictionBook 2.0). Структура цього формату, побудована на основі XML, передбачає можливість вбудовування не лише графічних ресурсів, але й довільних допоміжних метаданих. Така особливість відкриває нові можливості для реалізації стеганографічних алгоритмів, зокрема через модифікацію обкладинки (графічного блоку) або тегів, що не впливають на відображення тексту під час читання [1-3].

Актуальність теми обумовлена тим, що електронні книги активно використовуються як у комерційній, так і в освітній сферах, а також є поширеним об’єктом нелегального копіювання. Застосування стеганографії у таких документах дозволяє, з одного боку, організовувати прихований канал обміну повідомленнями в умовах відсутності безпечного з’єднання, а з іншого – впроваджувати механізми цифрового маркування з метою виявлення джерел поширення неліцензованих копій.

Метою роботи є розробка методів та алгоритмів для вбудовування прихованої інформації в електронні книги формату FB2 із використанням стеганографічних методів, зокрема на основі LSB–модифікації зображень обкладинки та приховування і вбудовування інформації в тегах метаданих.

До *теоретичних методів дослідження* належать*:*

- аналіз наукових і технічних джерел з тематики стеганографії та форматів електронних документів;
- порівняльний метод для оцінки ефективності різних підходів до приховування інформації;
- метод синтезу для формування узагальненої концепції побудови стеганоконтейнера на основі формату FB2.

До *практичних методів дослідження* відносяться*:*

- моделювання структури електронного документа як стеганоконтейнера;
- проєктування алгоритмів стеганографічного вбудовування й витягування інформації;
- тестування працездатності розробленого програмного забезпечення, зокрема перевірка збереження функціональності та цілісності файлів після внесення прихованих даних.

Область застосування розробленого рішення охоплює:

- організацію прихованого документообігу в інформаційно–обмежених середовищах;
- цифрове маркування електронних видань з метою виявлення витоків інформації;
- створення інструментів навчального або демонстраційного призначення у галузі стеганографії та інформаційної безпеки.

Вимоги до програмних та апаратних засобів:

- розробка програмного забезпечення передбачена мовами Python із використанням бібліотек для роботи з XML і зображеннями (наприклад, xml, OpenCV);

Таким чином, тематика роботи відповідає сучасним тенденціям розвитку засобів приховування інформації та має практичну цінність у контексті цифрової безпеки.

РОЗДІЛ 1
ОБ'ЄКТ ЗАХИСТУ: ПРИХОВАНА ІНФОРМАЦІЯ В ЕЛЕКТРОННИХ КНИГАХ ФОРМАТУ FICTIONBOOK

1.1 Електронні книги як носії інформації та їхні особливості

Розповсюдження інформації завжди було одним із важливих питань для людства з давніх віків. Першим кроком для вирішення даної проблеми було створення засобів для передачі повідомлень – писемності, як засоби перетворення вербальної мови у графічну форму та відповідних матеріалів, як фізичних носіїв для збереження даних. Тогочасним ефективним рішенням стали книги, у зв'язку з їхніми перевагами – відносно легке виробництво, можливість відтворювання, а починаючи з XVI сторіччя, достатньо масштабним явищем.

Внаслідок розвитку електронних та комунікаційних технологій у XX столітті, почали розглядатись ідеї щодо розповсюдження інформації, зокрема книг, засобами автоматизованих та інформаційно–комунікаційних систем. Були зроблені спроби оцифрування (перенесення з фізичного в цифровий простір) книжок, а результат такої операції – почали називати електронними книгами.

Спочатку вони не мали великої відомості серед масового читача через недостатній розвиток засобів і методів оцифрування, тогочасні складності поширення. Але з плином часу електронні книги зайняли велику позицію в інформаційній сфері, ставши ефективним та зручним інструментом для створення, надійного зберігання та швидкого розповсюдження інформації. До їхніх сильних сторін відносять – висока доступність, зручність використання за різних умов та можливість взаємодії з текстом – саме вони зробили їх популярним цифровим форматом у різноманітних сферах: науці, освіті, культурі, бізнесі. Для зручного продовження кваліфікаційної роботи, будемо використовувати наступне точне визначення.

Електронна книга, крім того згадується як електронний текст або електронний документ – це цифровий документ з певною стандартизованою структурою, що містить текстову та/або графічну інформацію та призначений для відображення на електронних пристроях з дисплеєм: планшетах, комп’ютерах, смартфонах [4]. Раніше вважалось, що до електронних книг відносять виключно книги – що пройшли процес оцифрування, але на даний момент є екземпляри, що були створенні виключно в цифровому просторі.

Також треба зауважити , що під терміном «електронна книга», можуть мати на увазі програмне або апаратне забезпечення з плоским дисплеєм для зберігання та відображення оцифрованої книжки. Однак в контексті даної кваліфікаційної роботи, даний тип пристроїв будуть згадуватись під назвою "електронний зчитувач", для уникнення плутаниць.

Для більшого розуміння сутності, особливостей, деталей та механізмів розглянемо історію виникнення такого явища, як електронні книги, та, за можливістю, відмітимо їхню суттєву еволюцію.

1.1.1 Історія виникнення та еволюції електронних книжок

Як згадувалось на початку підрозділу, поява на світ носія текстової інформації, відомий нам як електронна книга, відбулась у XX столітті завдяки розвитку обчислювальної техніки, що в цілому надало більше можливостей для обробки, зберігання та розповсюдження інформації.

Перша спроба реалізації концепції електронної книжки рахується 1949 року, у вигляді пристрою з назвою “Enciclopedia Mecanica”, що був створений та запатентований іспанською вчителькою, письменницею та вченою Анхелою Руїс Роблес. Даний пристрій являв собою механічну енциклопедію, яка реалізовувала можливість читати книги у змінному форматі за допомогою обертання барабанів, що провертали стрічки з текстом [5]. Хоча даний пристрій і не був електронним, він вважається предком сучасних електронних зчитувачів та натхненником електронних книг.

Перша фактична реалізація електронної книги була здійснена 4 липня 1971 року американським студентом Майклом С. Хартом. У зазначеному випадку була вручну оцифрована Декларація незалежності США. Для розповсюдження даного екземпляру книги між університетами Харт скористався засобами мережі ARPANET, відправивши посилання на файл [6]. Дослідники вважають конкретно цей текст першою відомою електронною книгою. В подальшому, його ініціатива перетворилась на багаторічний проєкт – Проєкт Гутенберг, який добровільно та безкоштовно здійснював оцифрування книжок [7].

У перше десятиліття власного існування електронні книжки були представлені в форматі звичайних текстових документів, що уникало можливість впровадження форматування, розмітки та розміщення сторонніх, нетекстових, елементів [8]. Таким чином, вони могли зберігати лише текстові дані та працювати у будь–яких текстових редакторах.

Іноді, для малого забезпечення форматизації та стилістичного оформлення, електронні книги зберігались у мові розмітки HTML. Але у зв’язку з тимчасовим розвитком цієї мови, засоби для кращого візуального подання були дуже обмеженні [9].

Покращенню ситуації сприяв розвиток комп’ютерних технологій у 1980–х – 1990–х роках, що викликав виникненню перших форматів файлів, адаптованих під структуроване збереження документів. Важливо виокремити появу Portable Document Format (PDF) за авторством компанії Adobe [10]. Спочатку він був створений як формат, що забезпечував унифікований метод друкування документів з збереженням форматування, використання різних шрифтів, додання ілюстрацій і навіть елементів дизайну з різних пристроїв, але надалі швидко набрав популярність як формат електронної книги. Саме розвиток технологій дозволив адаптувати їх під потреби бізнесу, науки, юриспруденції, тощо.

У проміжок час між 1998 та 2008 роками, було реалізовано серійне виробництво електронних зчитувачів, але збільшення масового використання майже унеможливлювалось через відсутність сумісності між різними

пристроями унаслідок браку єдиного стандарту для електронних текстів, неможливість відображення мультимедійних елементів [11]. Винятком були зчитувачі, які підтримували формат PDF – який і так є стандартизованим і відкритим.

Саме для вирішення цих обставин і, відповідно, популяризації електронних зчитувачів, було заподіяно кроки в напрямі виникнення нових відкритих форматів електронних книжок. Більшість тогочасних і сучасних форматів засновано на базі мови розмітки XML, що забезпечує чітку структуру, гнучкість, кросплатформність. Особливу популярність здобув формат EPUB, що вирізнявся своєю відкритістю, пристосованістю до різних розмірів дисплеїв, здатністю до динамічного переформатування тексту, можливістю інтеграції мультимедіа та підтримкою зручної навігації всередині книги [12].

Отже, шлях розвитку електронних книг – від елементарних текстових файлів до комплексних, упорядкованих цифрових контейнерів – є свідченням формування самостійного електронного носія інформації. Цей носій не тільки вміщує значні обсяги інформації, а й здатен зберігати дані, що потребують захисту від несанкціонованого копіювання.

1.1.2 Загальна структура електронної книги

Якщо розглядати електрону книгу з технічної точки зору, то нескладно помітити наявність єдиної структури, яка поєднує в собі форматизовану текстову частину, метадані носія і мультимедійні елементи, які вбудовуються в текст. Виходячи з цього, вона є не виключно текстовим контейнером, а гнучким засобом відображення тексту, здатним адаптуватись до різних умов, відповідно до різноманітних вимог від користувачів.

Перше на що потрібно звернути увагу, що основний вміст файлу – текстовий блок, формується з декількох логічних відокремлених розділів, підрозділів та секцій. Для їхньої розмітки використовують, частіше всього, технології на базі мови розмітки XML. Саме завдяки цьому підходу стає

можливим однозначно вказувати не лише заголовки й абзаци, але й виноски, цитати чи навіть складні структурні елементи – наприклад таблиці чи формули, що, своєю чергою, впливає на точність відображення та зручність навігації.

Попри той факт, що для звичайного користувача у електронній книзі існує тільки текст, вони також містять відокремлений набір службових даних про автора, видавця, дату та місце написання, тощо. Даний набір даних називається метаданими. Також до них можуть входити такі технічні атрибути, як ISBN, посилання на веб–ресурс, DRM–сертифікат. Виокремимо той факт, що саме завдяки метаданим є можливість автоматизування пошуку, каталогізації та контролю легальності поширення публікацій.

Крім того, електронні книги оснащені навігаційними механізмами: змістом, закладками й гіперпосиланнями, які забезпечують швидкий перехід між будь–якими фрагментами документа. Водночас ці компоненти часто виокремлюють у спеціальні XML–файли або вбудовані блоки розмітки, що спрощує їхню підтримку та оновлення без втручання в основний текст.

Обкладинка та ілюстрації, хоча й вторинні стосовно змісту, відіграють не менш важливу роль. Як правило, це растрові або векторні зображення, які зберігаються окремими файлами або вбудованими бінарними даними у межах контейнера. Механізми розмітки відіграють ключову роль у відображенні зображень та можуть варіюватись в залежності від формату книги.

Якщо розглядати зі сторони оформлення, то у більшості випадків відповідальність за це належить електронним зчитувачам. Але все–таки самі електронні книги також мають можливість персоналізувати стилі відображення тексту. Конфігураційні файли є відділеними від самого тексту та мультимедійних елементів, що забезпечує:

1) підвищення адаптації книжки під різні засоби відображення;

2) можливість власного налаштування стилізації книжок завдяки таблицям стилів.

Останнім пунктом іде ймовірна присутність файлу маніфесту – опису ієрархії всіх наявних ресурсів в електронній книзі. При відсутності цього файлу,

ні один програмний зчитувач не буде в змозі коректно інтерпретувати зміст видання.

1.1.3 Додаткові механізми електронних книжок

Більшість сучасних електронних книг містять не тільки текстових даних та мультимедійних елементів, а також велику кількість рівнів захисту від несанкціонованого доступу (НСД) до вмісту. Одним із прикладів такого захисту є часткове шифрування змісту електронної книги сучасними засобами криптографії, а саме:

1) симетричні алгоритми шифрування (AES, RC4);
2) асиметричні алгоритми шифрування (RSA, ECC).

При використанні даного виду захисту від НСД, можливість перегляду змісту книги без відповідного ключа технічно унеможливлюється [13].

Іншим методом захисту від НСД, а також забезпечення цілісності і автентичності, є використання механізму цифрового підпису, який використовувався сумісно з шифруванням. У більшості випадків, такий підпис вбудовувався або в метадані електронного видання, або в файл маніфесту, який згадувався в минулому підрозділі. Перевірка цифрового підпису здійснювалась кожного разу при відкритті книги через зчитувач. Крім того, механізми DRM реалізуються через систему сертифікатів і ліцензійних ключів, що визначають допустимі дії над файлом: від читання і прокручування сторінок – до копіювання фрагментів, друку чи пересилання [13]. Усе це забезпечує правовласникам ефективний інструментарій для захисту інтелектуальної власності в цифровому середовищі.

Іншим значимим механізмом електронного тексту є попереднє генерування індексів ключових слів та фраз, що використовується для підвищення швидкості та зручності навігації у документі. Такий індекс формується ще на етапі підготовки публікації, коли відповідний скрипт або модуль автоматично аналізує зміст твору, виокремлює семантично значущі

лексеми, після чого створює структурований довідник для більш швидкого пошуку. Завдяки роботі цього механізму пошукові запити, що вводяться користувачем у вікні зчитувача, не обробляються шляхом повного перегляду вмісту документа, а натомість звертаються до цього попередньо створеного індексу, що значно зменшує час відгуку системи та знижує навантаження на обчислювальні ресурси пристрою.

Здебільшого, візуально електронна книга зазвичай сприймається як цілісний файл, її внутрішня структура передбачає механізм зберігання та вбудовування посторонніх шрифтів [14]. Це дозволяє забезпечити коректне відтворення нестандартних символів, спеціальних графічних знаків, математичних формул, а також тексту, набраного мовами з алфавітами, що відсутні у більшості кодувань. У такий спосіб вирішується проблема відсутності відповідних символів у системних шрифтах кінцевого пристрою – незалежно від того, про який електронний зчитувач іде мова. Завдяки вбудованим шрифтам забезпечується повна відповідність візуального оформлення тексту авторському задуму.

Нарешті, одним із найменш очевидних, але вкрай корисних функціональних елементів сучасних електронних видань є можливість модульного оновлення вмісту. У багатьох випадках зчитувачі та дистрибуційні платформи підтримують функцію довантаження додаткових глав, оновлень, виправлень, ілюстрацій або навіть локалізованих версій вмісту. При цьому зберігаються всі персоналізовані налаштування користувача – зокрема, поточна позиція читання, позначені закладки, особисті нотатки тощо. Це перетворює електронну книгу на динамічний об’єкт, який може змінюватися з плином часу без необхідності повторного завантаження всього файлу. Видавці, у свою чергу, отримують можливість оперативного реагування на виявлені помилки або актуалізацію вмісту, не порушуючи логіки поширення.

1.2 Стеганографія та можливі причини її використання в електронних книжках

Стеганографію можна описати як сукупність методів та засобів для забезпечення прихованого передавання інформації, яке здійснюється завдяки алгоритмам вбудовування однієї інформації в носій іншої, зовсім непов'язаної з першою. Головною метою стеганографії полягає не тільки в збереженні конфіденційності повідомлення, як у випадку з шифруванням, а до того ж і приховування факту його існування та передачі від зовнішніх спостерігачів та систем аналізу, тобто забезпечення доступності [15].

Варто зауважити, що носії інформації, вбудованої стеганографічними методами, називаються стегоконтейнерами. У більшості випадків, стегоконтейнерами виступають мультимедійні дані (представленні у вигляді зображень, аудіо– та відеофайлів).

Для зручності вбудовування та витягування прихованих повідомлень використовується сукупність засобів та методів, спрямованих на створення та функціонування прихованого каналу інформації та забезпечення обміну стегоконтейнерами – так звані стеганографічні системи (стеганосистеми). Приклад роботи такої стеганосистеми зображений на рис. 1.1.

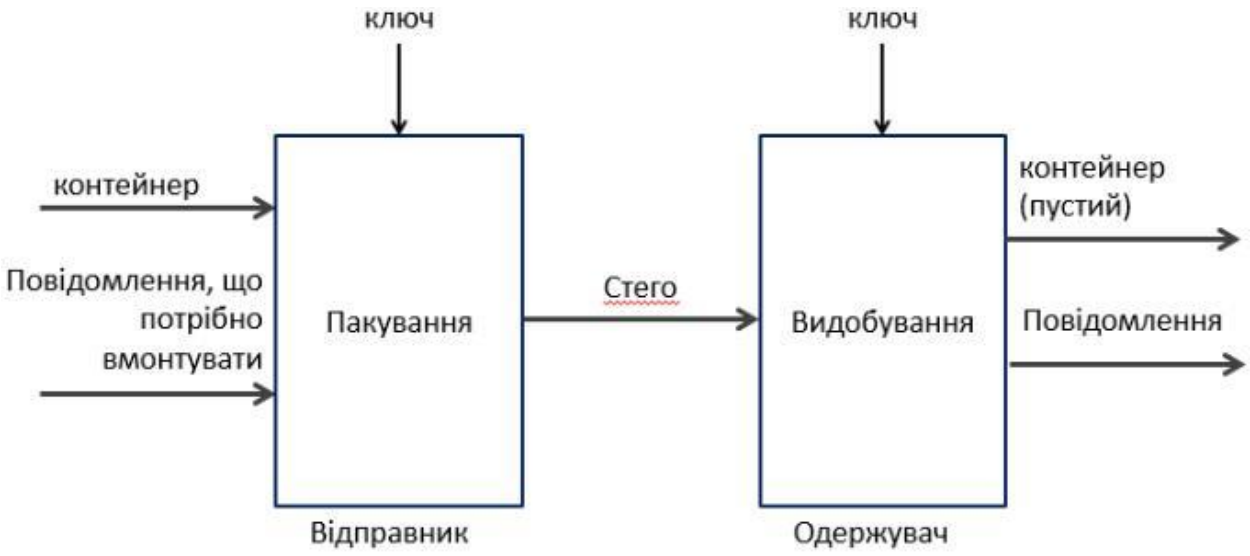

Рисунок 1.1 – Загальний приклад роботи стеганографічної системи

В контексті сучасної стеганографії, розглядають два окремих види стеганографії, з різними типами контейнерів для приховування та, відповідно, різними алгоритмами вбудовування та витягування інформації – цифрову та комп’ютерну. Про них детальніше в наступному підрозділі.

1.2.1 Різновиди стеганографії

Цифрова стеганографія – різновид сучасної стеганографії, яка заснований на вбудовуванні прихованої інформації в цифрові об’єкти, як правило мультимедійного формату – зображення, аудіо, відео, текстури 3D–моделей і т. д. [16]. Даному різновиду стеганографії характерна непомітна модифікація цифрового об’єкту через внесення незначних спотворень, які знаходяться нижче порогу чутливості органів людини відтворенні стегоконтейнеру. До прикладів цифрової стеганографії відносять:

- *Ехо–стеганографія в аудіо файлах.*

Цей метод передбачає вбудовування даних в аудіо–контейнер шляхом введення в нього ехо–сигналу. Дані приховуються зміною трьох параметрів ехо–сигналу: початкової амплітуди, швидкості загасання та зсуву.

- *Приховування в кольорових каналах відео*.

Цей метод передбачає вбудовування даних в відео–контейнер шляхом модифікації одного з кольорових каналів (наприклад, синього) в межах його допустимого діапазону амплітуди.

Приклад використання цифрової стеганографії з таємним повідомленням “Kiev National University” зображений на рис. 1.2.

Рисунок 1.2 – Приклад використання цифрової стеганографії
(зліва – пустий контейнер; справа – контейнер з повідомленням)

Комп’ютерна стеганографія – різновид сучасної стеганографії, яка заснований на експлуатації особливостей комп’ютерних та файлових систем, операційних середовищ та форматів комп’ютерних файлів для подальшого вбудовування в них інформації [16]. Суттєвою різницею від цифрової стеганографії полягає у використанні технічних аспектів обробки і зберіганні файлів, такими як – метадані, службові поля, імена файлів, специфіка файлових систем, тощо. До прикладів комп’ютерної стеганографії належать:

- *Приховування даних у зарезервованих полях форматів файлів*

Суть методу полягає в тому, що частина поля розширень, не заповнена інформацією про розширення, за замовчуванням заповнюється нулями. Відповідно ми можемо використовувати цю «нульову» частину для запису своїх даних. Недоліком цього методу є низький ступінь скритності і малий обсяг переданої інформації [17].

- *Використання полів, що не візуалізуються під час рендерингу*

Цей метод ґрунтується на спеціальних «невидимих» полях для отримання виносок, покажчиків. До прикладу, написання чорним шрифтом на чорному тлі. До недоліків відносять маленьку продуктивність, невеликий обсяг переданої інформації.

Відповідно, використання стеганографічних засобів в структурі електронних книжок обґрунтована не лише з точки зору прихованого

передавання повідомлень, а й як механізм для цифрового маркування, підвищення стійкості авторських прав, контролю цілісності цифрового контенту та унеможливлення несанкціонованого поширення даних у відкритих середовищах.

На завершення варто зазначити, що стеганографія в електронних книжках не обмежується лише цифровими форматами мультимедіа або службовими даними файлів. Вона також охоплює більш складні, комбіновані методи, які поєднують приховування з елементами криптографії [18].

1.2.2 Використання стеганографії

В контексті забезпечення та порушенні стану кібербезпеки, стеганографія відіграє далеко не останню роль. Саме завдяки використанню стеганографічних методів можливо створювати приховані канали зв’язку та здійснювати маркування того чи іншого контенту без видимих ознак втручання. Можна спробувати звести велику кількість можливих способів використання даного методу до наступних загальних сценаріїв:

- *Організація непублічних каналів зв’язку*

При виконанні даного сценарію здійснюється вбудовування секретного повідомлення в легітимні носії інформації або в надлишкові поля мережевих протоколів. Перевагами здійснення такого сценарію є – уникнення детектування звичайними моніторинговими системами, здійснення передачі інформації в умовах цензури або обмеженого доступу до криптографічних засобів [16].

- *Створення цифрових водяних знаків*

Даний сценарій передбачає вбудовування ідентифікаційних міток у мультимедійний контент для отримання можливості встановлення джерела витоку несанкціонованих копій та здійснення відстеження нелегального розповсюдження. За рахунок внесення незначних змін у носіях інформації, непомітних для кінцевого користувача, гарантується незмінність візуальної або акустичної складової файлу [17].

- *Негативні застосування у кіберпросторі*

Як правило, даний сценарій використовується зловмисниками для доставки команд керування шкідливим програмним забезпеченням (stegware) або ексфільтрації даних [18].

Якщо розглядати використання стеганографії в контексті електронних документів, то виникає велика специфіка у зв'язку з використанням мультимедійних елементів в публікаціях. Вбудовування інформації в електронні книги може здійснюватися заради втілення вищезгаданих сценаріїв, так і для окремих, актуальних саме для захисту електронних видань. До цих сценаріїв належать:

- *Приховані текстові маркери*

Здійснення цього сценарію полягає у вставці ряду символів Unicode (такі як нуль–ширині символи або невидимі пробіли) з метою вбудовування службової інформації (ім'я та ідентифікатор покупця, дата замовлення, адреса магазину) для забезпечення ідентифікації можливого каналу витоку, тощо.

- *Приховування в XML–контейнері*

У структурованих форматах секретні дані розміщуються в коментарях або в невикористаних атрибутах XML, що не впливають на відображення контенту. Це дозволяє вбудовувати додаткові метадані без ризику порушення логіки читання або навігації.

- *Стеганографічне маркування копій*

Сценарій базується на вбудовуванні унікальних ідентифікаторів у службові поля документа або в надлишкові біти графічних елементів обкладинки. Така цифрова відмітка дає змогу простежити шлях розповсюдження кожної копії екземпляру електронного видання та встановити відповідальне за витік джерело.

Таким чином, стеганографія в електронних книжках забезпечує мультиплексний захист: від прихованих каналів передачі до цифрового маркування, що, в результаті, підвищує загальний рівень інформаційної безпеки у сфері цифрових публікацій.

1.2.3 Загрози виявлення та стеганоаналіз у контексті електронних книжок

Оскільки основною метою цифрової стеганографії є приховане вбудовування інформації в певний об'єкт, а не ускладнення сприйняття повідомлення, то акцент стеганографічних алгоритмів спрямований саме на складність виявлення наявності прихованої інформації, хоча навіть це не гарантує повної неявності. Саме цей фактор сприяв появі новому галузі дослідження та напряму в інформаційній безпеці – стегоаналізу [19].

Цей напрямок описує методи, при використанні яких можливо здійснити виявлення факту приховування та передачі схованої інформації в стегоконтейнері. Як і у випадку з стеганографією, даний напрям найчастіше всього використовується у роботі з цифровими об'єктами, такими як графічні (зображення, відео) та звукові файли. Для виявлення вбудованих повідомлень використовуються статистичні, евристичні та спектральні методи аналізу, які можуть вказати на наявність нетипових явищ (аномалій) у структурі файлу, який досліджують [19].

В контексті електронних книжок, стегоаналіз не настільки розвинений, як в інших напрямах використання, у зв'язку з малим використанням засобів стеганографії з електронними книгами в якості стегоконтейнеру та, логічно, низькою зацікавленістю у рамках дослідження стегоаналітиків. З цього виходить, що на даний момент відсутні інструменти та методи для здійснення стегоаналізу в електронних книжках.

Але й досі, аналітики можуть здійснювати аналіз за загальним змістом та структурою книжки, намагаючись виявити аномалії, що можуть вказувати на можливість вбудовування інформації. Про це можуть свідчити наступні чинники:

- наявність надмірної кількості метаданих та/або їхній нестандартний формат;
- використання невпорядкованих та/або нестандартних коментарів;

- порушення стилістичного відображення книжки;
- аномальний розмір чи формат певних блоків даних.

Якщо узагальнити згадані вище чинники, будь–які значні та помітні порушення або відхилення у структурі, відображенні, параметричних значень файлу можуть бути сприйнятті як потенційні індикатори вбудовування інформації.

Саме спираючись на це, спеціаліст з стегоаналізу може запідозрити використання стеганографії в даному файлі та почати здійснювати пошук блоку даних або структури, де могли бути використані стеганографічні методи. Для виявлення, стегоаналітик може спробувати наступні підходи:

- Статистичний аналіз метаданих – порівняння кількості та структури полів із типовими зразками;
- Перевірка вбудованих ресурсів – аналіз розміру графічних об'єктів або шрифтів на відповідність стандартам;
- Виявлення аномалій у бітових потоках – оцінка ентропії та виявлення «здуття» даних у мультимедіа.

Водночас, з огляду на наявні загрози виявлення прихованої інформації та розвиток методів стеганоаналізу, розробникам стеганографічних алгоритмів доцільно враховувати низку практичних рекомендацій, які спрямовані на зниження ймовірності викриття факту вбудовування даних. Наступні методи були побудовані завдяки теоретичному аналізу типових методів стегоаналізу та загальній структурі та поведінці цифрових об'єктів.

- Розпорошення інформації по всьому об’єму документа;
- Адаптивне вбудовування з урахуванням контексту вмісту;
- Маскування під типовий формат;
- Імітація стандартних параметрів.

Таким чином, поєднання грамотного стеганографічного дизайну з розумінням методів стеганоаналізу створює багаторівневий захист: зовнішній спостерігач (стегоаналітик), навіть знаючи алгоритм вбудовування, зустрінеться

з проблемою виявлення самого факту приховування, а автентифікований отримувач – без перешкод витягне оригінальне повідомлення.

1.3 Формат електронних книг FB2 та його можливості для використання стеганографії

Формат FB2 (FictionBook 2.0) є відкритим XML-базованим форматом електронних книг, розробленим насамперед для художньої літератури. Він призначений для уніфікованого подання цифрових версій книг у вигляді структурованого XML-документу, що забезпечує сумісність з різними пристроями та полегшує програмну обробку [1, 2].

За фактичним визначенням, формат FB2 – це просто один файл із розширенням .fb2 (або архів .fb2.zip), що містить весь вміст книги (текст, зображення та метадані) у вигляді XML.

Така організація даних дає змогу вільно форматувати текст та зберігати ілюстрації (PNG, JPEG) без залучення додаткових файлів. Також даний формат підтримує багато елементів форматування: витримки, цитування, таблиці, виноски, початкові епіграфи тощо. При цьому вміст файлу залишається відкритим та без механізмів шифрування змісту – FB2 файли не містять вбудованих механізмів захисту авторських прав, що робить їх вільно змінюваними та конвертованими.

Фактично FB2 не містить жодних вбудованих механізмів DRM. Це обумовлено відкритою природою XML-формату та розрахунком на вільний обмін цифровими книгами. Таке рішення дозволяє читачам безперешкодно копіювати й передавати FB2–книги, проте водночас ускладнює контроль за несанкціонованим поширенням контенту. У цілому, відсутність жорстких механізмів захисту авторських прав у FB2 є однією з його ключових властивостей: формат передбачає повну свободу доступу до вихідної структури книги.

1.3.1 Функціональні можливості формату

Формат FictionBook2 характеризується гнучкою кросплатформеною сумісністю та широкими можливостями форматування, що забезпечує його популярність серед користувачів різних програмно–апаратних платформ. Завдяки відкритості стандарту, на сьогоднішній день розроблено значну кількість програмних засобів для читання та конвертації документів у форматі FB2.

Серед настільних застосунків найпоширенішими є Calibre, Cool Reader, FBReader, Haali Reader, STDU Viewer та Athenaeum. У середовищі мобільних пристроїв і пристроїв для читання електронних книг (електронних зчитувачів) підтримка формату реалізована в прошивках пристроїв PocketBook, Cybook, BeBook, а також у низці спеціалізованих застосунків.

FB2–файли можуть бути відкриті безпосередньо у багатьох програмах або легко конвертовані у формати EPUB, MOBI, PDF та інші за допомогою кросплатформеного програмного забезпечення, зокрема Calibre. Крім того, формат є повністю сумісним із Unicode та допускає вказівку довільного кодування XML–документа (наприклад, UTF–8 або Windows–1251), що забезпечує коректне відображення текстів будь–якими мовами.

1.3.2 Елементи структури файлу FB2 придатні для стеганографії

Опис структурних складових FB2 починається з кореневого елемента <FictionBook>. Під ним опціонально може іти блок <stylesheet>, однак найчастіше він не використовується (програми–зчитувачі за замовчуванням застосовують власні стилі). Основні відділи документа розташовані в такому порядку: <description>, <body> та, опціонально, будь–яку кількість <binary> [3]. Загальна структура документу даного формату зображена на рис. 1.3.

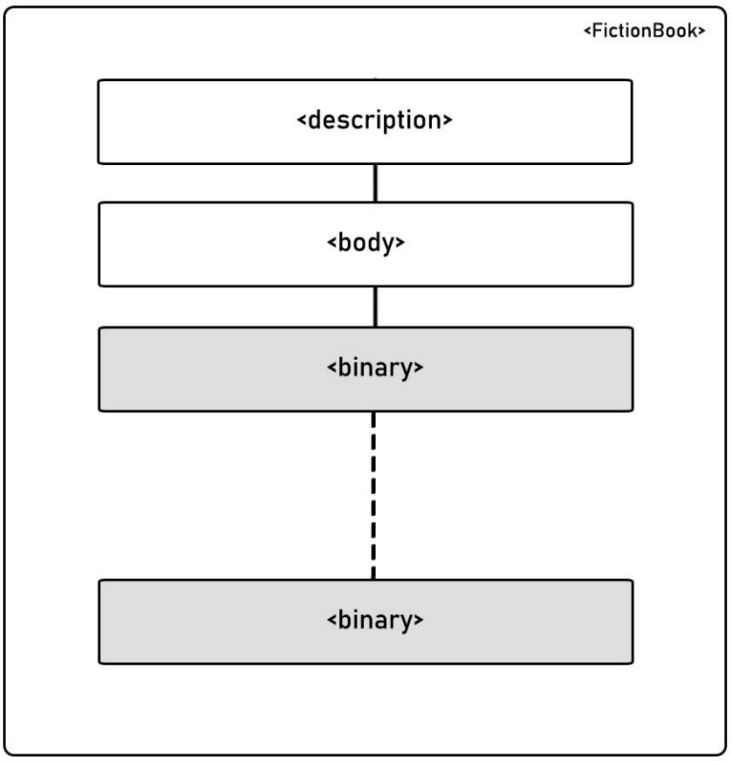

Рисунок 1.3 – Загальна структура формату FictionBook2
(сірим відмічені опціональні розділи)

Розділ <description>, що є одним із ключових структурних компонентів документа формату FB2, призначений для зберігання метаданих, які описують основні характеристики електронної книги [20]. Його внутрішня структура (зобр. на рис 1.4), згідно зі специфікацією FictionBook 2.0, передбачає наявність низки підсекцій, серед яких найпоширенішими є:

- <title–info> – інформація про змістове наповнення та авторів книги;
- <document–info> – технічні дані, що стосуються створення документа;
- <publish–info> – відомості про публікацію, якщо такі наявні;
- <custom–info> – розширювана секція, призначена для довільних додаткових даних, не передбачених основною схемою.

У блоці <title–info> зазвичай визначаються такі елементи: жанр видання, персональні дані авторів, назва твору, анотація, мова твору та дата публікації.

Крім того, у межах цієї підсекції може бути задано графічне зображення обкладинки книги, що реалізується за допомогою елемента <coverpage>, який, у свою чергу, містить тег <image> із посиланням на ідентифікатор відповідного об’єкта в секції <binary> (наприклад, #cover.jpg). Саме така схема посилання дозволяє коректно відобразити обкладинку в більшості сучасних програм для читання електронних книг.

Варто приділити увагу підрозділу <custom–info>, у якому дозволено використовувати довільні, нестандартизовані поля, завдяки тому, що більшість програм–зчитувачів ігнорує наявність даного елементу. Згідно стандарту FictionBook, цей елемент використовується для збереження користувацької або службової інформації та може за потреби бути розширеним. Через те що зміст даних полів не відображається при читанні електронних книжок стандартними зчитувачами та, з цього виходить, необізнані користувачі не звертають увагу при використанні книжки. Саме ця властивість робить підрозділ <custom–info> потенційно корисними у якості місця зберігання прихованої інформації.

Таким чином, структурна гнучкість та підтримка відкритих форматів у межах розділу <description> формату FB2 надають широкі можливості як для легального опису книги, так і для інтеграції додаткових функцій, зокрема пов’язаних із інформаційною безпекою та маркуванням контенту.

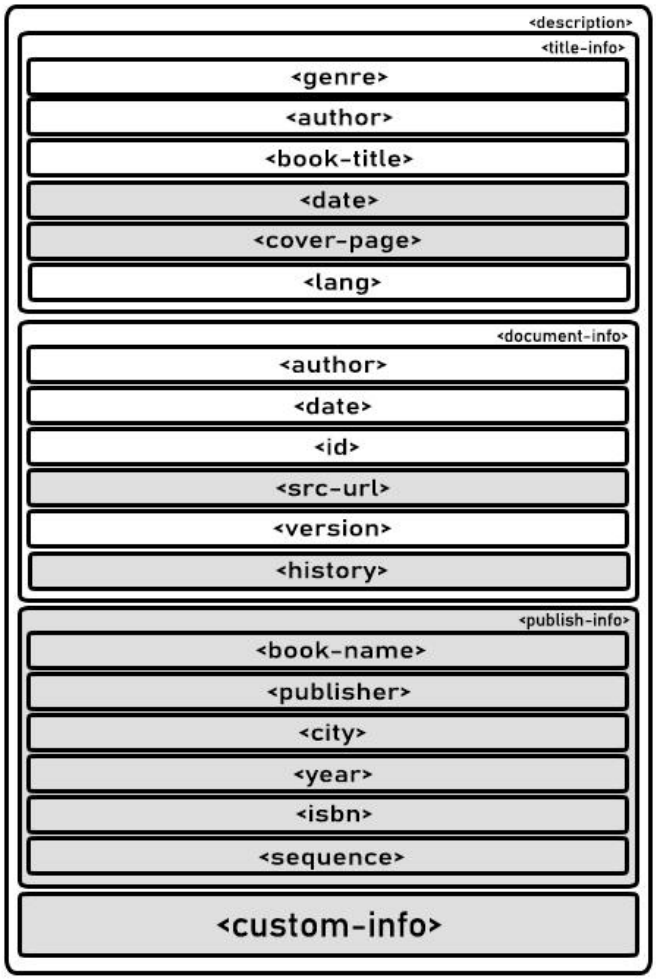

Рисунок 1.4 – Структура розділу <description>
(сірим відмічені опціональні секції та/або елементи)

Окрему функціональну роль у структурі файлу формату FB2 відіграють так звані секції <binary>, призначенням яких є зберігання вбудованих двійкових об’єктів у текстовому представленні [21]. Названі елементи використовуються для вбудовування графічних ресурсів, такі як – ілюстрації, зображення обкладинок або декоративних елементів – що використовують при оформленні текстового контенту електронної книги.

Відповідно до специфікації формату, дані елементи повинні бути у форматі PNG або JPEG та завчасно бути закодовані у стандарті кодування Base64. Саме такі вимоги дозволяють інтегрувати зображення в структуру електронного документу без потреби зберігання у вигляді окремих файлів.

Кожен елемент <binary> повинен містити два основні атрибути:

– id – унікальний ідентифікатор, який використовується для посилання на вміст цього елемента з інших частин документа (зокрема з тегу <image> у розділі <coverpage> або всередині розділів <section>);

– content–type – MIME–тип файлу, який визначає формат даних, такі як image/jpeg чи image/png.

Вміст елемента <binary>, хоча і поданий у текстовій формі, фактично є бінарною інформацією, що була закодована для сумісності з XML. Внаслідок цього зберігання графічних даних у форматі Base64 дозволяє уникнути зовнішніх залежностей – уся необхідна мультимедійна інформація міститься безпосередньо всередині FB2–файлу.

Отже, секції <binary> виконують критично важливу функцію в архітектурі FB2–документа: вони не лише забезпечують графічну складову твору, а й створюють потенційну точку для реалізації стеганографічного приховування інформації. Вбудоване зображення може бути використане як носій закодованого повідомлення, що розширює можливості формату у сфері інформаційної безпеки та цифрового маркування.

1.3.3 Можливості прихованого вбудовування інформації

Структурна побудова формату надає декілька можливостей для стеганографічного вбудовування інформації, що не була передбачені розробниками книжки. Згідно минулому підпункту, розділ <description> має в собі секцію <custom–info> для зберігання будь–яких довільних користувацьких даних. Саме використання цього сегменту структури дозволить розмістити додаткові інформаційні поля, які будуть за зоною сприйняття читачів та базових програм–зчитувачів, що можна використати для передачі прихованого повідомлення або нанесення цифрового маркування. Також варто відмітити, що даний формат, у зв’язку з походженням від мови XML, дозволяє використовувати типові коментарі всередині книги. Виходячи з цього, велику

кількість даних, повідомленнь або підказок можна залишити у зазначених вище місцях, без порушення відображення книжки у читача.

Продовжуючи тему можливостей для вбудовування, не можна не помітити потенціал розділу <binary>, як носія прихованої інформації. Хоча зазвичай туди поміщають зображення, формат не обмежує кількість або призначення цих блоків. Наприклад, можна вказати <binary> з нетиповим id і content–type, і вкладати будь–які дані у Base64, які програма читання просто пропустить як невідомий блок. Інший сценарій – використовувати реальний графічний файл: приховати повідомлення у пікселях зображення перед кодуванням і вбудувати таке «замасковане» зображення в <binary> (тобто самі зображення служать стегоконтейнером). Також у метаданих (наприклад, у <document–info> або <publish–info>) можна вказувати нестандартні поля, де сховати дані у вигляді нерозрізненого тексту чи чисел.

Загалом гнучка структура FB2 – велика кількість тегів і атрибутів, можливість використання коментарів та розділу <binary> – створюють потенціал для приховування інформації всередині книги. Хоча при звичайному читанні ці додаткові дані будуть непомітні, спеціальні програми можуть їх витягувати. Такий метод стеганографії використовує те, що формат FB2 не обмежує строго своє наповнення, а багато полів у метаданих та бінарному розділі можуть містити будь–яку довільну інформацію.

Висновки до першого розділу

У першому розділі було розглянуто електронні книги як сучасні цифрові носії, здатні виступати як об’єкт зберігання, передавання та навіть приховання інформації. Проаналізовано історичні аспекти розвитку електронних книжок – від перших концептуальних пристроїв до сучасних форматів, які поєднують структурованість, мобільність і високий ступінь інтеграції з апаратними та програмними платформами.

Особливу увагу приділено структурі електронних книг як багаторівневим об’єктам, що включають текстовий вміст, метадані, мультимедійні елементи, механізми захисту авторських прав і засоби навігації. Було акцентовано на потенціалі таких структур не лише для надання інформації користувачу, але й для реалізації додаткових – у тому числі прихованих – функцій.

Окрема частина розділу присвячена теоретичному та прикладному аналізу стеганографії – науки про приховання інформації – у контексті електронних книжок. Визначено типові підходи цифрової та комп’ютерної стеганографії, показано доцільність їх використання в текстових форматах документів і розглянуто практичні сценарії – від створення прихованих каналів до цифрового маркування публікацій.

Нарешті, проаналізовано відкритий формат FictionBook 2.0 (FB2) як приклад електронної книги, який, завдяки своїй XML–базованій структурі, низькому рівню обмежень і широким можливостям форматування, відкриває значний потенціал для стеганографічного застосування. Описано технічні властивості формату, його логічну організацію та специфічні ділянки, що можуть бути використані для прихованого вбудовування інформації.

Таким чином, результати цього розділу створюють концептуальну основу для подальшого формування стеганографічного методу, орієнтованого на формат FB2, та обґрунтовують вибір даного формату як перспективного об’єкта дослідження в галузі прихованого передавання даних.

РОЗДІЛ 2
РОЗРОБКА МЕТОДИКИ ВБУДОВУВАННЯ ПРИХОВАНОЇ ІНФОРМАЦІЇ В ФАЙЛИ FB2

2.1 Метод знаходження місця для вбудовування інформації в електронній книжці формату FB2

У сфері інформаційної безпеки надзвичайно важливою передумовою для ефективного застосування стеганографічного захисту є наявність у цифровому об’єкті такого логічного і технічно обґрунтованого фрагмента структури, який би дозволяв приховано вбудовувати додаткову інформацію без порушення цілісності чи функціональності самого носія. У цьому контексті формат електронних книг FictionBook, що базується на структурі XML–документа, виявляється досить сприятливим для реалізації стеганографічних рішень, оскільки він передбачає декілька внутрішньо допустимих та гнучких механізмів включення користувацьких або інших даних, що потребують приховування.

Одним із таких рішень, які застосовуються у межах прихованого захисту електронних публікацій у форматі FB2, є використання елемента <custom–info>, який належить до секції <description> [20]. Даний елемент структури, як згадувалось в позаминулому підрозділі, призначений для збереження додаткових метаданих, що не були описані в специфікації формату, з чого можна дойти до висновку, що дане місце може бути використане для вбудовування прихованих повідомлень без можливості виявлення звичайним користувачем та порушенням роботи файлу даного формату.

З іншої сторони, не менш цікавим та ефективним альтернативним варіантом є використання стеганографічних алгоритмів для вбудови інформації в зображення обкладинки видання, який, як відомо, зберігається у елементі <binary>. І того більше, виходячи зі змісту підрозділу 1.3.3, графічний блок обкладинки зберігається у форматі кодування Base64, що дозволить проводити

стеганографічні операції без потреби працювати ззовні файлу електронної книжки.

Незважаючи на структурну відкритість і формальну простоту XML–документів, забезпечення повноцінного доступу до всіх їхніх складових, а також підтримка точного редагування у форматі FictionBook вимагає застосування спеціалізованих інструментів [1-3]. Тому, для реалізації автоматизованої обробки таких документів доцільно використовувати XML–парсери, які не лише забезпечують коректну інтерпретацію вмісту, але й підтримують специфічну для FictionBook структуру та простір імен. Саме такий підхід сприяє як підвищенню ефективності розробки стеганографічних модулів, так і забезпеченню стійкості захисту інформації в умовах динамічного інформаційного середовища.

2.1.1 Знаходження елементу <custom–info>

З метою первинної обробки даних програмний модуль розпочинає свою роботу з ініціалізації XML–аналізатора. Даний компонент відповідає за зчитування головного елемента <FictionBook>, при цьому обов'язково враховується простір імен, що є невід'ємною характеристикою структури FictionBook. Зазначена процедура має ключове значення, оскільки саме вона забезпечує всебічний доступ до внутрішньої ієрархічної організації документа. Як наслідок, аналізатор автоматично проводить верифікацію наявності основних структурних одиниць, серед яких особливе місце займає секція <description>.

Після успішного завершення початкової фази ініціалізації, модуль переходить до етапу навігації в межах документа, а саме до підрозділу <description>. У випадку виявлення відсутності цього елемента, ініціалізується завершення роботи програмного модуля з поверненням стану помилки. Іншими словами, новий вузол генерується безпосередньо під час виконання програми, що не тільки відновлює структурну цілісність документа, але й уможливлює запобігання можливих помилок у процесі подальшого внесення інформації. Водночас необхідно забезпечити відповідність новоствореного елементу

вимогам XML–схеми, аби не порушити валідацію документа за стандартом FB2, оскільки найменше відхилення може спричинити відхилення всього файлу під час перевірки цілісності.

Коли секція <description> вважається знайденою, парсер виконує пошук цільових вузлів <custom–info>, використовуючи для цього XPath–запит, сконструйований за шаблоном /FictionBook/description/custom–info. У випадку, якщо виявляється хоча б один елемент <custom–info>, який містить атрибут info–type, значення якого вказує на можливу наявність вбудованого повідомлення, програма відразу повертає відповідний службовий статус, що підтверджує виявлення маркованого інформаційного блоку. Завдяки цьому алгоритм переходить до наступного етапу обробки, оминаючи кроки, пов’язані з генерацією додаткового вузла, що дозволяє уникнути дублювання інформації.

Інакше кажучи, якщо в документі відсутній елемент <custom–info> з відповідним атрибутом або такий елемент не містить релевантних даних, модуль виконує вставку нового вузла. Цей вузол супроводжується обов’язковим атрибутом info–type з умовним значенням "stego", а в окремих випадках – також і атрибутом lang="uk". Задля пришвидшення роботи алгоритму вбудовування та витягування, вставка здійснюється після секції <publish–info>, а у випадку її відсутності, в кінці секції <description>, що дозволяє зберегти логічну послідовність документа. Наглядний приклад алгоритму пошуку даного елементу зображений на рис. 2.1.

Рисунок 2.1 – Загальний вигляд пошуку елементу <custom–info>

2.1.2 Знаходження елементу <binary> зі змістом обкладинки

З метою первинної обробки даних програмний модуль розпочинає свою роботу з ініціалізації XML–аналізатора. Даний компонент відповідає за зчитування головного елемента <FictionBook>, при цьому обов'язково враховується простір імен, що є невід'ємною характеристикою структури FictionBook. Зазначена процедура має ключове значення, оскільки саме вона забезпечує всебічний доступ до внутрішньої ієрархічної організації документа. Як наслідок, аналізатор автоматично проводить верифікацію наявності основних структурних одиниць, серед яких особливе місце займає секція <description>.

Після успішного завершення початкової фази ініціалізації, модуль переходить до етапу навігації в межах документа, а саме до підрозділу <description>. У випадку виявлення відсутності цього елемента, ініціалізується завершення роботи програмного модуля з поверненням стану помилки. Іншими

словами, новий вузол генерується безпосередньо під час виконання програми, що не тільки відновлює структурну цілісність документа, але й уможливлює запобігання можливих помилок у процесі подальшого внесення інформації. Водночас необхідно забезпечити відповідність новоствореного елементу вимогам XML–схеми, аби не порушити валідацію документа за стандартом FB2, оскільки найменше відхилення може спричинити відхилення всього файлу під час перевірки цілісності.

Коли секція <description> вважається знайденою, парсер виконує пошук вузлів <coverpage>, які знаходяться у підсекції <title–info>. Модуль фактично продовжує роботу, та знаходить елемент <image>, що вказує на обкладинку та в якому наявний атрибут xlink:href. Згідно специфікації формату FictionBook, саме цей атрибут відповідає за лінкування елементів <image> та <binary>, тому записуємо його значення в окрему змінну та продовжуємо роботу.

Після виявлення відповідного атрибута, з його значення програмно вилучається сам ідентифікатор, який, у свою чергу, використовується для пошуку відповідного вузла <binary> у структурі всього XML–документа. Проте якщо з’ясується, що блок <coverpage> відсутній, або ж якщо виявлений атрибут xlink:href не заданий, чи містить некоректне посилання, модуль реєструє стан, за якого обкладинка вважається такою, що відсутня. Модуль завершує роботу функції та надсилає повідомлення о помилці, після чого подальша можлива обробка відповідної ділянки документа припиняється.

У ситуації, коли елемент <binary> успішно знайдено, починається детальніший аналіз: модуль звертається до його атрибута content–type, щоби переконатися в тому, що графічний формат відповідає очікуваним стандартам, таким як image/jpeg або image/png. Після підтвердження допустимості формату вміст, закодований у форматі Base64, декодується у тимчасовий двійковий файл. Це, у свою чергу, надає змогу оцінити фізичні характеристики зображення, зокрема його розмір, що безпосередньо впливає на вибір методів стеганографічного вбудовування інформації. Наглядний приклад алгоритму пошуку даного елементу зображений на рис. 2.2.

Рисунок 2.2 – Загальний вигляд алгоритму пошуку елементу <binary> який містить обкладинку

Після завершення перевірки функція модуля повертає атрибути зі значенням ідентифікатора та типу контенту зображення (прикладом послужить coverimage.jpg та image/jpeg, тотож), вказуючи як дані, пов'язані напряму з бінарним представленням зображення обкладинки, що знадобиться під час вбудовування повідомлення.

Зрештою, результатом виконання описаної процедури є повернення пари змінних, що містять посилання на конкретний вузол <binary>, значення його атрибута content–type, а також безпосередньо сам вміст цього вузла. Така структура слугує основою для наступних етапів вбудовування інформації в документ.

2.2 Розгляд та вибір стеганографічних алгоритмів для вбудовування інформації в зображення обкладинки

Під час створення методики стеганографічного захисту електронних книжок виникає необхідність ретельного вибору алгоритму, який би одночасно забезпечував прийнятний компроміс між такими ключовими характеристиками, як ємність, сприйнятлива невидимість і стійкість до стандартних перетворень зображення. Саме тому у межах даного підрозділу детально розглянуто два класичні підходи, кожен із яких сформувався внаслідок специфічних історичних та технічних передумов, що, своєю чергою, зумовили особливості їх реалізації та застосування в сучасних інформаційних системах.

Зважаючи на характер поставленої задачі, ці підходи заслуговують на окрему увагу ще й тому, що їх практична ефективність була підтверджена у різних галузях, включно з цифровим захистом авторських прав, приховуванням маркерної інформації та цифровою ідентифікацією. Отже, кожен із описаних варіантів може бути потенційно адаптований для створення програмного засобу, призначеного для вбудовування прихованої інформації у структуру електронних книжок формату FictionBook, при цьому забезпечуючи заданий рівень захисту без помітного впливу на якість сприйняття цифрового контенту користувачем.

2.2.1 Метод зміни найменш значущого біта

Метод найменш значущого біта (LSB) є одним із найпоширеніших, а водночас і найпростіших способів реалізації цифрової стеганографії [15]. Його ефективність ґрунтується на маніпулюванні бітовим представленням пікселів у растрових зображеннях, де прихована інформація інтегрується безпосередньо в колірні компоненти зображення на рівні окремих бітів. Саме завдяки своїй технічній простоті, алгоритм широко використовується в освітніх проєктах, прототипах систем цифрового маркування та низці практичних рішень, що не вимагають високої криптостійкості.

У випадку 24–бітного кольорового зображення кожен піксель описується трьома байтами – по одному для кожного з основних колірних каналів: червоного, зеленого та синього [23]. З урахуванням розбиттям на три канали, то в цілому можливо за рахунок одного пікселю непомітно модифікувати по три біти даних – по кожному найменш значущому біту на окремий кольоровий канал. Приймаючи до уваги можливості людського зору, а саме поріг чутливості до кольорів, можливо зазначити, що одночасна зміна усіх трьох бітів ніяк не скажеться на сприйнятті зображенні. Унаслідок цього зображення зберігає візуальну цілісність, а внесені зміни залишаються малопомітними навіть при детальному перегляді. На рис. 2.3 зображений загальний принцип роботи алгоритму.

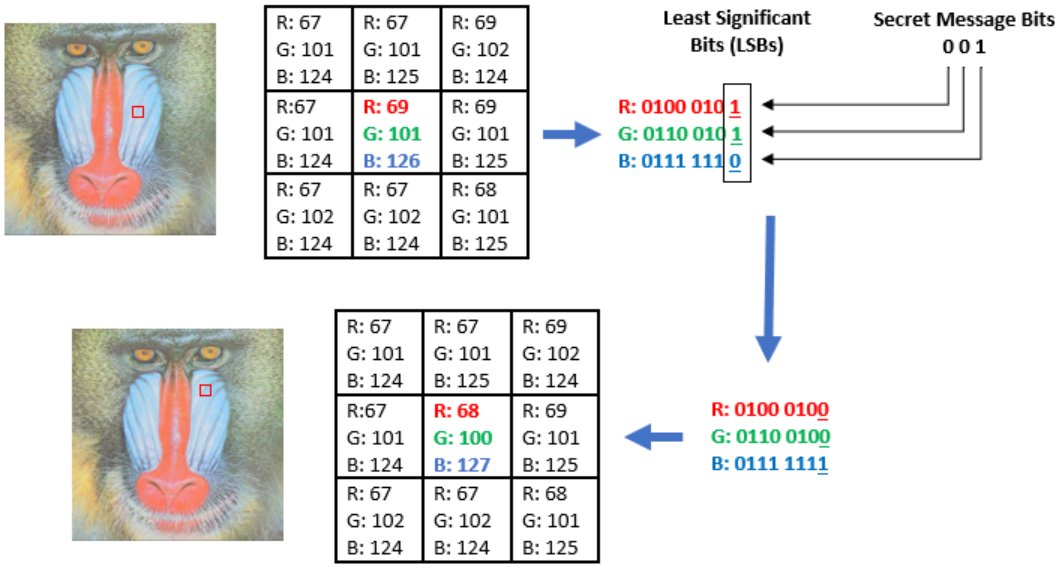

Рисунок 2.3 – Принцип та приклад роботи методу LSB

Однак варто зазначити, що найчастіше при реалізації LSB–методу, все ж таки використовується лише один кольоровий канал, тобто 1 з 24 бітів пікселю. Дотримання таких обмежень дозволяє, якщо не уникнути, то мінімізувати ризик виникнення візуальних артефактів, зберігаючи при цьому допустиму ємність для прихованого повідомлення. Наприклад, у зображенні з роздільною здатністю 800 на 600 пікселів, загальна кількість пікселів становить 480 000. При дотриманні умови вбудовування одного біта в кожен канал RGB, загальний обсяг даних, які

можливо приховати, досягає 1,44 Мбіт, що еквівалентно приблизно 180 КБ текстової інформації – доволі вагомий показник для задач прихованого обміну повідомленнями.

Механізм вбудовування інформації передбачає послідовну зміну значень найменш значущих бітів згідно з бітовим потоком зашифрованого або відкритого повідомлення. Як правило, використовуються два варіанти розміщення бітів:

1) Вбудовування у послідовному порядку, при якому кожен наступний біт повідомлення записується в черговий піксель, починаючи з першого. Використання даного варіанту дозволяє закодувати більше повідомлення [24];

2) Вбудовування у псевдовипадковому порядку, за якого порядок пікселів, у які вбудовується інформація, визначається генератором псевдовипадкових чисел, ініціалізованим певним визначеним і оговореним ключем. Використання даного варіанту підвищує рівень ентропії та зменшує ймовірність виявлення вбудованого сигналу в ході стеганоаналізу.

Проте, попри свою простоту і гнучкість, метод LSB має низку недоліків, головним з яких є низька стійкість до впливу зовнішніх перетворень. Навіть незначне стискання зображення за допомогою алгоритмів із втратою (наприклад, JPEG–компресія), зміна роздільної здатності, обрізка або застосування фільтрів (як–от розмиття чи покращення контрасту) можуть призвести до часткової або повної втрати прихованої інформації. З цієї причини даний метод найдоцільніше використовувати у випадках, коли графічний файл не піддається подальшій обробці або використовується лише у «контрольованому» середовищі передачі.

Крім того, LSB–метод є вразливим до статистичних методів виявлення прихованої інформації. Зокрема, RS–аналіз, χ^2–тест, тестування рівномірності бітових розподілів та інші відомі підходи дають змогу з високою ймовірністю виявити факт наявності вбудованих даних у зображенні. Для зниження ризику детектування на практиці застосовуються так звані адаптивні методи вбудовування, що уникають рівномірного розподілу даних по всіх пікселях та

надають перевагу менш однорідним ділянкам зображення, де бітові зміни менш помітні.

Таким чином, хоча метод найменш значущого біта не позбавлений обмежень і не може вважатися універсальним для всіх сценаріїв застосування, він залишається одним із найефективніших та найпростіших інструментів для прихованого передавання інформації, особливо в умовах, де не очікується агресивна обробка графічного носія та забезпечується достатній рівень контролю за каналом поширення. Його актуальність зберігається і в сучасних умовах, коли поєднання LSB із додатковими захисними заходами (наприклад, шифруванням або псевдовипадковим маскуванням) дозволяє створити доволі стійкі та ефективні стеганографічні системи.

2.2.2 Метод накладання розширеного спектру

Серед стеганографічних алгоритмів вбудовування інформації розглянутих в цьому підрозділі – метод розширеного спектра має один із найвищих рівнів стійкості до зовнішнього впливу, за рахунок використовування завадостійкого кодування [25]. Метод використовування розширеного спектру прийшов у стеганографію зі сфери телекомунікаційного зв'язку, особливо в радіосфері, де схожі технології застосовувались для забезпечення більш захищеної та стійкої до перешкод передачі даних. У цифровій стеганографії даний метод реалізується шляхом «розсіювання» закодованої інформацію серед усіх пікселів зображення з використанням псевдовипадкових послідовностей, які відіграють роль модулюючого сигналу.

На початку приховане повідомлення піддається попередньому кодуванню за допомогою коригувальних кодів, такі як Боуза–Чоудхурі–Хоквінгема або, схожим, Ріда–Соломона, що дозволить відновидити вміст повідомлення у разі часткової втрати або спотворення даних. Після цього закодоване повідомлення розбивається на окремі мали сегменти, які будуть вбудовуватись в зображення не локалізовано, а розподілено по всій площі графічного блоку. Кожен сегмент

поєднується з псевдошумовою послідовністю, яка генерується на основі секретного ключа і має вигляд цифрового сигналу з високою ентропією. Завдяки цьому процесу формується закодований потік, що інтегрується у значення яскравості пікселів або конкретних колірних каналів (наприклад, синього – через його меншу чутливість для людського ока).

Ключовою особливістю при використанні цього методу є той факт, що без знання секретного ключа синхронізація з модулюючим сигналом є неможливою, а отже – виявлення та декодування прихованого повідомлення також стає надзвичайно складним. Це забезпечує високий рівень непомітності: навіть у випадку наявності фактів прихованого вбудовування, зловмисник без ключа не має змоги відновити оригінальну інформацію.

Ще однією перевагою методу розширеного спектра є його виключна стійкість до широкого спектра зовнішніх впливів, серед яких:

– Виникнення або додавання стороннього шуму, що імітує перешкоди при зберіганні або передачі файлу;

– Будь–який вид стиснення з втратами, який зазвичай руйнує вбудовану інформацію у просторовій області;

– Часткова зміна країв або певного фрагменту зображення, що може траплятися при зміні формату;

– Різноманітні геометричні перетворення, зокрема обертання, масштабування, зсув.

Завдяки глобальному розподілу інформації в межах усього зображення, зміна або втрата меншої частини пікселів не має фактичного впливу на цілісність прихованого повідомлення. Ситуація зворотня, оскільки втрата частини носія компенсується за рахунок надмірності, закладеної у корегувальних кодах.

Проте зазначений метод не позбавлений і певних обмежень. Зокрема, його ємність істотно поступається іншим методам: зазвичай вона не перевищує кількох кілобіт на зображення зі стандартною роздільною здатністю (наприклад, 1024 на 768 пікселів). Такий обсяг може бути недостатнім для зберігання великих

повідомлень, однак цілком придатний для передавання ключів, цифрових ідентифікаторів, водяних знаків або службової інформації.

Таким чином, метод розширеного спектра доцільно розглядати як універсальний інструмент для стеганографії в умовах ризикованого інформаційного середовища. Він забезпечує максимальну стійкість, високу ступінь непомітності, а також надає можливість часткового відновлення прихованих даних навіть після значних перетворень носія. Саме тому він вважається одним із найбільш надійних методів у сучасній практиці прихованого передавання інформації, хоча й потребує більш складної реалізації та використання допоміжних механізмів корекції помилок.

2.3 Побудова алгоритму вбудовування та витягування прихованої інформації

При розробці програмного модуля, одним із фундаментально важливих складових є розробка і побудова чітких алгоритмів, особливо це стосується таких важких процесів, як стеганографічні перетворення. Заздалегідь виконане проектування та реалізація таких алгоритмів є критично важливим процесом у формуванні загальної архітектури програми та дозволяє досягти логічної завершеності. У свою чергу, воно значною мірою спрощує подальшу розробку і впровадження програмного коду, забезпечує структурованість під час його тестування, оптимізує процес верифікації результатів і дозволяє швидко локалізувати та усувати потенційні помилки.

Завдяки здійсненню формалізованого представлення кожного кроку забезпечується прозорість логіки роботи системи. В наслідок цього, розробник має можливість попередньо оцінити обчислювальну складність та ефективність алгоритму, виявити потенційні вузькі місця, підібрати різні методи реалізації, передбачити поведінку програми в умовах нестандартних або виключних ситуацій, а отже – зменшити ймовірність некоректного функціонування при обробці реальних даних.

У цьому підрозділі детально проаналізовано два окремі підходи до прихованого обміну даними: насамперед буде розглянуто алгоритм вбудовування та наступного витягування інформації безпосередньо в зображення обкладинки файлу формату FictionBook, унаслідок чого весь процес інтеграції відбувається всередині бінарного вмісту елемента <binary>. Потім, незважаючи на схожість загальної мети, буде окреслено інший спосіб, який передбачає використання розширеного розділу з користувацькою інформацією – так званого елемента <custom–info> – де дані спочатку шифруються й кодуються, а тільки потім додаються до структури XML.

2.3.1 Алгоритм вбудовування та витягування прихованої інформації в обкладинку книги

В даному випадку, процедура вбудовування інформації розпочинається з ідентифікації обкладинки, поданої у кодуванні Base64 в межах елементу <binary> оброблюваної книги формату FB2. Відповідно після отримання Base64–рядку, першим кроком буде його декодування до вигляду послідовності байтів. Далі, за необхідності, йде зміна формату растрового зображення до типу, якому не характерні втрати при стисненні (наприклад, як PNG), щоб забезпечити цілісність повідомлення після вбудовування.

Після отримання цілого зображення обкладинки у відповідному форматі необхідно здійснити оцінювання максимальної довжини повідомлення (кількість символів), що може бути приховане. Задля цього необхідно потрібно помножити загальну площу зображення (кількість пікселів) на кількість кольорових каналів і на два (кількість бітів повідомлення, які будуть вбудовуватись) та поділити отриманий результат на вісім (максимальну кількість біт, яку використовує ASCII символ). У випадку коли зображення недостатньо містке для приховування повідомлення, програмний модуль має завершити роботу функції та повернути помилку. В зворотному випадку, функція продовжує роботу, та здійснить перетворення повідомлення у послідовність бітів.

У самому процесі запису повідомлення кожні два біти із сформованого потоку послідовно накладається на два найменш значущі біти відповідного кольорового каналу кожного пікселя. Оскільки бітова маніпуляція відбувається по двом, менш значимим позиціям, це призводить до мінімальних візуальних артефактів й при цьому забезпечує достатній рівень стеганографічного приховання. У міру поступового проходження по всьому масиву пікселів бітовий потік цілком вбудовується в середовище зображення.

Після завершення запису даних слід зберегти модифіковане растрове зображення у відповідному форматі, після чого виконати нове кодування результату у Base64 та повернути цей рядок у структуру FB2 – безпосередньо в елемент <binary>. Наприкінці варто запустити процедуру перевірки коректності FB2–документа, що включає валідацію XML–схеми та тестовий перегляд обкладинки, аби гарантувати відсутність пошкоджень. Наглядний приклад алгоритму вбудовування інформації в даний елемент зображений на рис. 2.4.

Рисунок 2.4 – Блок–схема алгоритму вбудовування повідомлення в обкладинку

У випадку витягування прихованого повідомлення, алгоритм майже не відрізняється від алгоритму вбудовування. Спочатку здійснюється зчитування вмісту елементу <binary>, що містить обкладинку; подалі йде декодування вмісту з формату Base64 в впорядкований масив байтів та представленні у відповідному форматі растрового файлу. Важливішим етапом йде послідовне зчитування найменшого біту кожного кольорового компоненту, як потенційних бітів прихованого повідомлення. Після накопичення всіх можливих бітів, здійснюється остання операція – декодування послідовності бітів у текстове повідомлення кодування ASCII. Наглядний приклад алгоритму витягування інформації з даного елементу зображений на рис. 2.5.

Рисунок 2.5. Блок–схема алгоритму витягування повідомлення з обкладинки

2.3.2 Алгоритм вбудовування та витягування прихованої інформації в елементі книги <custom–info>

Реалізація даного алгоритму вбудовування починається з підготовки повідомлення – воно конвертується до бінарного представлення (якщо це текст кодування ASCII), а за потреби виконуються інші необхідні перетворення. Саме отримання масиву байтів необхідне для наступного кроку – шифрування.

Далі відповідно до вимог безпеки визначається симетричний ключ одного з варіантів AES (у нашому випадку це 128 біт), і здійснюється шифрування даних у вибраному режимі (для забезпечення конфіденційності виберемо CBC).

Внаслідок чого з'являється криптотекст, який у подальшому кодується в форрмат Base64, для забезпечення компактності повідомлення.

На стадії інтеграції першочергово перевіряється наявність елемента <custom–info> у межах розділу <description> FB2–файлу. У разі, якщо зазначений контейнер повністю відсутній або наявний, але не містить жодних вкладених підпунктів, дозволяється виконати вбудовування шляхом створення нового елемента <custom–info>, вмістом якого стає зашифрований та закодований у формат Base64 рядок. Натомість, якщо контейнер <custom–info> уже існує та містить хоча б один підпункт з відповідним атрибутом info-type, процес вбудовування не ініціюється, а користувач інформується про неможливість продовження операції через потенційне порушення логіки або структури вже наявної користувацької інформації. Наглядний приклад алгоритму вбудовування інформації в даний елемент зображений на рис. 2.6.

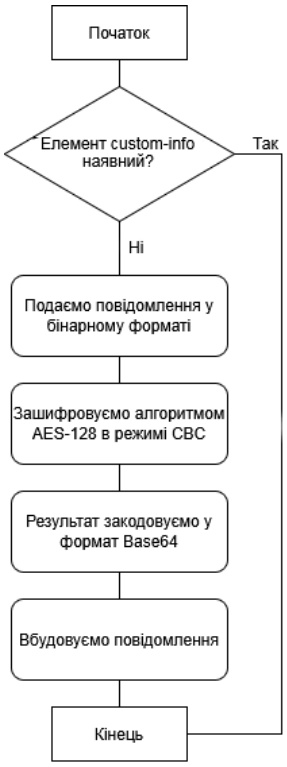

Рисунок 2.6 – Блок–схема алгоритму вбудовування повідомлення в елемент <custom–info>

Коли потрібно витягти прихований фрагмент, спочатку виконується пошук <custom–info info–name="steg–hidden"> у документі. Зчитування його текстового

вмісту й наступне декодування з Base64 повертають сирий криптотекст. Потім застосовується процедура розшифрування з використанням того самого симетричного ключа, що й при вбудовуванні; у разі відповідності ключів дані успішно перетворюються на вихідний бінарний набір, який згодом декодується в текст ASCII або у двійковий формат. У протилежному випадку–якщо ключ неправильний або елемент відсутній–система прагматично генерує повідомлення про невдалу спробу, інформуючи користувача про неможливість відновлення прихованої інформації. Наглядний приклад алгоритму витягування інформації з даного елементу зображений на рис. 2.7.

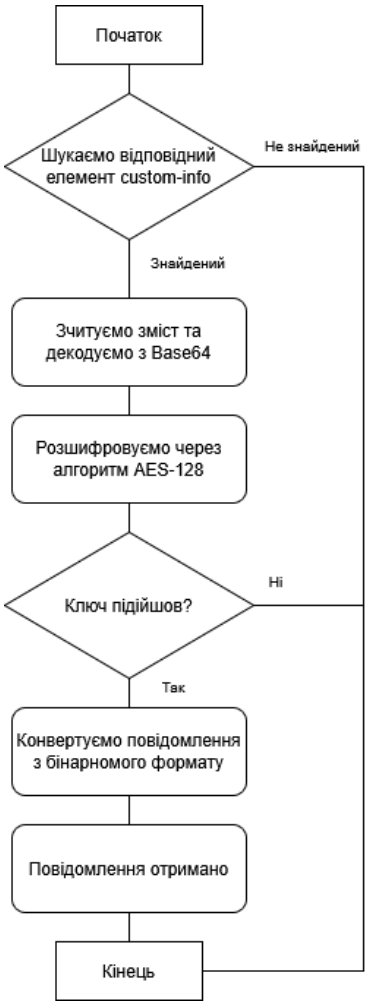

Рисунок 2.7 – Блок–схема алгоритму витягування повідомлення з елементу <custom–info>

Висновки до другого розділу

У другому розділі було проведено низку детальних досліджень, результати яких всебічно проаналізовано та використано як основу для розробки ефективних методичних підходів до стеганографічного вбудовування інформації в електронні книги формату FB2. Насамперед обґрунтовано доцільність вибору двох основних зон, придатних для ін’єкції даних: це елемент <custom–info>, що входить до секції <description>, а також графічний блок обкладинки, представлений як елемент <binary>. Такий вибір зумовлений не лише зручною структурною позицією цих вузлів у загальній архітектурі документа, але й тим, що сучасні засоби XML–парсингу дозволяють легко і без ручного втручання виявляти відповідні елементи. Це значно спрощує підготовку до вбудовування даних і робить увесь процес більш стабільним, керованим та відтворюваним за необхідності.

Наступним логічним кроком стало формулювання чітких вимог до алгоритмів стеганографічної ін’єкції. Враховуючи потребу в детермінованості, зворотності, стійкості до змін носія, а також простоті реалізації й достатньому рівні захисту вбудованих повідомлень, було розроблено дві пари алгоритмічних процедур, які орієнтовані на роботу з різними типами вмісту. Перша – на основі класичного LSB–методу, який застосовується до зображення обкладинки. Цей підхід вирізняється простою реалізацією, високою швидкістю та великою ємністю для прихованих даних, однак виявився вразливим до агресивного стиснення графіки, що може суттєво обмежити його практичне використання в реальних умовах. Друга пара алгоритмів орієнтована на текстову частину документа: вона передбачає симетричне шифрування повідомлення алгоритмом AES із подальшим кодуванням результату у форматі Base64. Незважаючи на необхідність ретельного управління ключами, такий підхід забезпечує високу конфіденційність, стійкість інформації до аналізу та сумісність із форматами XML.

Опис процедур охоплює повну послідовність дій як для вбудовування, так і для витягування прихованих даних. У випадку графічного блока відбувається попереднє декодування зображення з Base64, його перетворення у растровий формат, після чого змінюються окремі біти згідно зі змістом повідомлення. При роботі з елементом <custom–info> передбачається перевірка наявності відповідного вузла або створення нового, шифрування вхідних даних і їхнє кодування у формат, придатний для інтеграції в XML–структуру документа, без порушення її цілісності.

У підсумку запропонована методика поєднує структурований підхід до виявлення зон для ін’єкції з чітко визначеними алгоритмічними процедурами, що в сукупності забезпечує як високий ступінь автоматизації, так і надійність, гнучкість і безпеку при реалізації програмного прототипу для стеганографічного вбудовування даних.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ МЕТОДІВ ПРИХОВУВАННЯ

3.1 Розробка функцій для обробки структури файлів формату FictionBook

Для реалізації практичної частини розробки необхідно попередньо визначити відповідне програмне забезпечення, яке буде використовуватись у процесі створення застосунку. У якості головного інструмента розробки обрано мову програмування обрано Python версії 3.13 [26], що зумовлено її широкими функціональними можливостями, високим рівнем читабельності коду, а також підтримкою кросплатформеності, що забезпечує запуск програми на різних операційних системах без необхідності значних модифікацій.

Серед ключових переваг Python варто відзначити наявність розвиненої екосистеми вбудованих і сторонніх бібліотек, що значно прискорює процес розробки програмного забезпечення, а також спрощує інтеграцію необхідних функцій без потреби створення низькорівневого коду з нуля. Крім того, мова дозволяє зберігати компактність і логічну структурованість проєкту, що важливо при розробці застосунків, які працюють із різними типами даних.

Для опрацювання файлів формату FictionBook 2.0 (FB2), доцільно застосовувати засоби обробки XML–документів, оскільки даний формат побудований саме на XML–структурі (див. підрозділ 1.3). У рамках проєкту використовується вбудований модуль ElementTree, який виконує функції XML–парсера та забезпечує інструменти для зручного зчитування, створення, модифікації та навігації по структурі електронного документа [27]. Такий вибір обумовлений простотою використання, стабільністю роботи та сумісністю з іншими бібліотеками Python.

3.1.1 Визначення простору імен XML–документу

Під час формування FB2–документів, окрім застосування власного розширення файлу для вказівки формату, розробник має можливість додатково визначати тип структури документа за допомогою простору імен який зазначається у кореневому елементі XML–файлу (рис. 3.1). Така практика відповідає загальноприйнятим підходам до валідації XML–документів.

Рисунок 3.1 – Приклад вказаного простору імен в FictionBook

Проте, незважаючи на формальну стандартизованість використання просторів імен, на практиці можуть виникати труднощі, пов’язані з використанням скорочених іменних префіксів замість повних URI–посилань. Хоча такі префікси слугують лише умовними маркерами для зручності розробника, XML–парсери вимагають чіткої відповідності між префіксом та його оголошенням, що має бути присутнє у кореневому вузлі [3]. У випадках, коли ця відповідність порушується, або коли парсер не може знайти належного простору імен для певного префікса, обробка документа унеможливлюється або супроводжується помилками.

Для запобігання виникненню даних помилок, необхідно забезпечити автоматичне визначення простору імен та відповідних префіксів. Задля цього побудуємо функцію (рис. 3.2), яка буде розглядати зміст кореневого елементу та, відповідно, наявні у ньому теги й буде повертати просто посилання на відповідний простір імен.

```python
def get_namespace(root):
    """Автоматично визначає простір імен із кореневого елемента."""
    if root.tag.startswith("{"):
        return root.tag.split("}")[0].strip("{")
    return ""
```

Рисунок 3.2 – Функція для визначення простору імен

Після визначення простору імен, який використовується у структурі електронного документа, подальша робота з елементами XML повинна здійснюватися з урахуванням цієї специфікації. З цією метою доцільно застосовувати допоміжну функцію (рис. 3.3), яка формує повне посилання елемента з ймовірним посиланням на відповідний простір імен.

```python
def ns_tag(tag, ns):
    """Повертаємо тег із простором імен (при наявності)."""
    return f"{{{ns}}}{tag}" if ns else tag
```

Рисунок 3.3 – Функція для коректного вказування тегу для роботи з парсером

3.1.2 Зчитування та запис зображення обкладинки

Наступним кроком, в контексті роботи зі змістом та структурою файлу формату FictionBook 2.0, іде процес зчитування та запису бінарного представлення зображення обкладинки. Для забезпечення зчитування, запису та зміни формату зображення скористуємось бібліотекою машинного зору – OpenCV [28].

Для роботи зі змістом самої електронної книжки, парсером спочатку. здійснюється відкриття електронної книжки та її кореневого елементу, водночас з визначенням її простору імен (рис. 3.4), для коректного розуміння структури книги.

```
tree = ET.parse(file_path)
root = tree.getroot()
ns = get_namespace(root)
```

Рисунок 3.4 – Відкриття файлу книги та її кореневого елементу

На наступному етапі обробки документа виконується пошук елемента coverpage, який міститься в структурі метаданих електронної книги та виконує функцію вказівника на графічне зображення обкладинки (див. підрозділ 2.1). Парсер здійснює зчитування та запам’ятовування ідентифікатору бінарного об’єкта (рис. 3.5), у якому безпосередньо зберігається зображення – зокрема, для знаходження відповідного елемента binary із вмістом обкладинки, який підлягатиме стеганографічній обробці.

```
for cover in root.findall(f".//{ns_tag('coverpage', ns)}"):
    for img in cover.findall(f".//{ns_tag('image', ns)}"):
        href = img.attrib.get('{http://www.w3.org/1999/xlink}href', "")
        if href.startswith('#'):
            cover_image_ids.append(href[1:])
```

Рисунок 3.5 – Пошук посилання на зміст обкладинки

Відповідно до цього, одразу виконується пошук бінарного елемента, що містить зображення обкладинки електронного видання, за збереженим раніше ідентифікатором (рис. 3.6).

```
binary_elem = None
for binary in root.findall(f".//{ns_tag('binary', ns)}"):
    if binary.attrib.get("id") == cover_id:
        binary_elem = binary
        break
```

Рисунок 3.6 – Пошук необхідного бінарного елементу

На наступному етапі, незалежно від того, яка саме операція передбачається – зчитування чи запис, – обов'язково виконується перетворення зображення між різними форматами подання даних. Залежно від контексту, це може бути перехід від Base64–представлення до бінарного формату (рис. 3.7), або навпаки – кодування бінарного зображення у вигляді Base64–рядка для збереження його у структурі XML–документа (рис. 3.8).

Після завершення етапу конвертації до відповідного формату, ініціюється основна операція обробки зображення – зокрема, зчитування або запис, що виконується із застосуванням модуля OpenCV (далі – cv2).

У разі потреби, наприклад, коли формат зображення не відповідає вимогам алгоритму чи очікуванням парсера, додатково виконується конвертація графічного файлу з одного формату до іншого – найчастіше з PNG у JPEG.

```python
base64_data = binary_elem.text.strip()
image_data = base64.b64decode(base64_data)

nparr = np.frombuffer(image_data, np.uint8)
img = cv2.imdecode(nparr, cv2.IMREAD_UNCHANGED)

if '.' in cover_id:
    output_filename = output_path or cover_id.rsplit('.', 1)[0] + '.png'
else:
    output_filename = output_path or cover_id + '.png'

success = cv2.imwrite(output_filename, img)
if not success:
    print("Помилка при збереженні файлу.")
    return False
```

Рисунок 3.7 – Процес зчитування представлення обкладинки та конвертації з Base64 в бінарний вигляд

```python
base64_data = base64.b64encode(buffer).decode('utf-8')
wrapped_data = "\n".join(textwrap.wrap(base64_data, 76))
binary_elem.text = wrapped_data
binary_elem.set("content-type", "image/png")

tree.write(file_path, encoding="utf-8", xml_declaration=True)
```

Рисунок 3.8 – Процес конвертації представлення з бінарного вигляду в Base64 та запис в елемент binary

3.1.3 Створення з записом та зчитування елементу custom–info

Наступним кроком, в контексті роботи зі змістом та структурою файлу формату FictionBook 2.0, іде процес зчитування та запису бінарного представлення зображення обкладинки.

В іншому випадку, залежно від вибору користувача, ініціюється первинний пошук елемента custom–info з потенційним вмістом прихованого повідомлення у структурі документа. Якщо елемент з певним атрибутом info–type уже існує та був успішно знайдений у межах відповідного простору імен, функція негайно припиняє виконання та повертає відповідне повідомлення, що сигналізує про наявність такого елемента (рис. 3.9).

```
has_custom_info = any(
    'info-type' in elem.attrib and elem.attrib['info-type'] in info_types
    for elem in custom_info_elements
)
```

Рисунок 3.9 – Процес пошуку існуючого елементу

У разі ж, якщо шуканий елемент не виявлено, функція продовжує виконання та створює новий елемент custom–info із заздалегідь визначеним маскуючим атрибутом, обраним користувачем із доступного переліку варіантів. Такий атрибут, як правило, імітує службову або технічну інформацію, що дозволяє органічно інтегрувати приховані дані у структуру метаданих без викликання підозри. Одразу після створення елемента здійснюється запис повідомлення, яке має бути приховане у текстовому вмісті, в межах новоствореного блоку (рис. 3.10).

```
custom_info = ET.Element(ns_tag("custom-info", ns))
custom_info.set("info-type", info_type)
custom_info.text = message

description_elem.append(custom_info)
tree.write(file_path, encoding="utf-8", xml_declaration=True)
```

Рисунок 3.10 – Створення елементу custom–info та запис прихованого повідомлення

У випадку зчитування раніше вбудованого повідомлення, система ініціює пошук наявного елемента custom–info у межах структурного блоку метаданих документа. Після знаходження здійснюється перевірка на предмет наявності текстового вмісту. Якщо такий вміст присутній, виконується зчитування повідомлення та його подальше форматування, яке включає очищення від надлишкових символів: зайвих пробілів, табуляцій, розривів рядків або непотрібної пунктуації (рис. 3.11).

```
for custom_info in root.findall(f".//{ns_tag('custom-info', ns)}"):
    info_type = custom_info.attrib.get('info-type')
    if info_type in info_types and custom_info.text:
        return custom_info.text.strip()
return ""
```

Рисунок 3.11 – Процес зчитування та форматування повідомлення з елементу custom–info

3.2 Впровадження стеганографічного та криптографічного методу для графічних та текстових компонентів

Одним із ключових та концептуально важливих етапів розробки програмного засобу для приховування інформації в електронних книгах формату FictionBook є реалізація алгоритмів стеганографічного вбудовування та криптографічного захисту даних. Відповідне виконання розробки даного етапу і визначає рівень функціональної спроможності застосунку до забезпечення конфіденційності, цілісності та непомітності переданої інформації. В залежності,

від обраних методів та їхньої реалізації залежить ефективність усього механізму приховування.

У процесі реалізації програмного рішення особлива увага приділяється поєднанню криптографічних засобів з методами цифрової стеганографії, що дозволяє створити багаторівневу архітектуру безпеки, де кожен рівень доповнює інший, забезпечуючи надійне приховування повідомлення навіть у разі часткового доступу до даних або аналізу структури документа.

Для вбудовування текстового повідомлення в блок, що відповідає за графічний вміст, обкладинки електронного видання та демонстрації можливостей стеганографії, буде використаний алгоритм LSB з редагуванням останніх двох бітів – оскільки він є простим у реалізації, має високу ємність для повідомлення та немає спостерігаємого впливу на якість зображення.

Оскільки стегоконтейнер може бути, так чи інакше, зламаний або пошкоджений, також потрібно попіклуватись про неможливість сприйняття повідомлення, яке може підлягати витоку. Задля цього скористуємось засобами сучасної криптографії – симетричним блоковим алгоритмом шифрування AES з ключем розміром в 128 біт (далі – AES–128) [30].

3.2.1 Побудова алгоритму вбудовування та витягування інформації методом LSB

Починаємо роботу з зчитування зображення обкладинки засобами бібліотеки cv2 з перевіркою, коректності відкриття файлу. Після цього здійснює перевірку формату масиву з байтами зображення та, у випадку невідповідності, корегуємо тип і формат. Також одразу зчитуємо загальну кількість зображення – висоту, ширину, кількість каналів (рис 3.12).

```python
img = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
if img is None:
    print("Не вдалося відкрити зображення.")
    return False

# Перевірка формату масиву і типу
if img.dtype != np.uint8:
    img = img.astype(np.uint8)

height, width, channels = img.shape
```

Рисунок 3.12 – Зчитування обкладинки та приведення до коректного формату і типу

Відразу після цього етапу, приступаємо до роботи з текстовим повідомленням. Визначаємо кінець повідомлення за рахунок додавання нуль–термінатора – спецсимвола для визначення кінця рядку. До того ж здійснимо перевірку його довжини, щоб бути впевненими, що воно вміститься в стегоконтейнер без втрати інформації або артефактів (рис 3.13).

```python
# Додаємо маркер та null-термінатор
message = "[[STEG]]" + message + chr(0)
max_message_length = (height * width * channels * 2) // 8  # 2 біти на канал, 8 біт в 1 байті

if len(message) > max_message_length:
    print(f"Повідомлення занадто довге. Максимум: {max_message_length - 1} символів.")
    return False
```

Рисунок 3.13 – Обробка повідомлення та перевірка його довжини

Вслід за успішним проходженням перевірки, проводимо перетворення текстового повідомлення в масив байтів та визначаємо їхню загальну кількість (рис 3.14).

```python
message_bits = []
for char in message:
    bits = format(ord(char), '08b')
    message_bits.extend([int(bit) for bit in bits])
```

Рисунок 3.14 – Переведення з текстового в бінарне представлення

Далі приступаємо до вбудовування бітів повідомлення в пікселі зображення. Проходимо по кожному пікселю та його кольорових каналам і здійснюємо послідовне вставлення двох бітів повідомлення в останні два біти кожного байта кольорового значення. Перед записом нових бітів очищуємо два найменш значущих бітів початкового значення за допомогою побітової операції AND. Далі модифіковане значення каналу з новими бітами записується назад у зображення. Після завершення вбудовування всієї послідовності бітів зображення зберігається, і функція припиняє виконання (рис 3.15).

```
for row in range(height):
    for col in range(width):
        for ch in range(channels):
            if bit_index + 2 <= total_bits:
                # Візьмемо 2 біти для запису
                bits_to_write = (message_bits[bit_index] << 1) | message_bits[bit_index + 1]

                # Очищаємо 2 молодших біти і записуємо нові
                original_val = img[row, col, ch]
                new_val = (original_val & 0b11111100) | bits_to_write
                img[row, col, ch] = new_val

                bit_index += 2
            else:
                # Всі біти вбудовані, зберігаємо і виходимо
                cv2.imwrite(image_path, img)
                return True
```

Рисунок 3.15 – Вбудовування бітів повідомлення

Операція по витягуванню прихованого повідомлення схожа за принципом з вбудовуванням повідомлення. Спочатку пікселі зображення перебираються по координатах, і з кожного кольорового каналу (синього, зеленого та червоного) послідовно зчитуються два найменш значущі біти. Отримані біти розділяються на окремі значення шляхом застосування побітових операцій зсуву та маскування. Зібрані біти додаються до загального списку для подальшого формування байтів і реконструкції текстового повідомлення. Такий підхід дозволяє відновити початкову інформацію з точністю до кожного біта (рис 3.16).

```python
for y in range(h):
    for x in range(w):
        pixel = image[y, x]
        for channel in range(3):  # B, G, R
            bits_from_channel = pixel[channel] & 0b11  # беремо 2 молодших біти
            bits.append((bits_from_channel >> 1) & 1)
            bits.append(bits_from_channel & 1)
```

Рисунок 3.16 – Витягування бітів повідомлення

3.2.2 Реалізація шифрування прихованого повідомлення алгоритмом AES–128 для подальшого вбудовування

Розпочинаємо з опису функції шифрування текстового повідомлення. На першому етапі програма перевіряє довжину ключа: вона повинна складати рівно 16 символів, що відповідає 128–бітній довжині для симетричного блоку шифру AES. Якщо умова не виконується – функція припиняє роботу з відповідним повідомленням про помилку (рис 3.17).

```python
if len(key) != 16:
    raise ValueError("Ключ повинен мати довжину 16 символів (128 біт).")
```

Рисунок 3.17 – Перевірка валідності ключа шифрування

Далі обидва аргументи – повідомлення та ключ – перетворюються з рядкового у байтовий формат, що необхідно для подальшої криптографічної обробки. Одразу після цього генерується випадковий 16–байтовий вектор ініціалізації (IV), який забезпечує унікальність кожної сесії шифрування навіть при використанні однакового ключа (рис 3.18).

```
key_bytes = key.encode('utf-8')
message_bytes = message.encode('utf-8')
```

Рисунок 3.18 – Перетворення даних у байти та генерація випадкового вектора ініцілізації

Використовуючи бібліотеку PyCryptodome [29], ініціалізується об’єкт шифру AES у режимі CBC (Cipher Block Chaining), після чого текст повідомлення шифрується з додаванням паддінгу згідно з розміром блоку. Отриманий зашифрований фрагмент об'єднується з вектором ініціалізації, утворюючи єдину структуру даних. У завершальній фазі ця структура кодується у формат Base64, що дозволяє безпечно зберігати чи передавати зашифроване повідомлення у текстовому вигляді (рис 3.19).

```
# Ініціалізація шифру та шифрування
cipher = AES.new(key_bytes, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(pad(message_bytes, AES.block_size))

# Комбінуємо IV та шифротекст
combined = iv + ciphertext
```

Рисунок 3.19 – Створення блоку шифротексту

У зворотному процесі розшифрування використовується функція decrypt_message. Тут одразу виконується перетворення ключа у байти та повторна перевірка його довжини. Після цього вхідний рядок у форматі Base64 декодується у байтовий масив, з якого окремо виділяються ініціалізаційний вектор (перші 16 байтів) та власне шифротекст (рис 3.20).

```
iv = encrypted[:16]
ciphertext = encrypted[16:]
```

Рисунок 3.20 – Виділення вектора ініціалізації та зашифрованого тексту з повідомлення

На завершальному етапі створюється об’єкт AES з тими самими параметрами, що й під час шифрування, і відбувається декодування повідомлення з наступним видаленням доданого паддінгу. Результатом функції є початковий текст повідомлення у звичайному рядковому представленні (рис 3.21).

```
cipher = AES.new(key_bytes, AES.MODE_CBC, iv)

# Розшифрування з видаленням паддінгу
decrypted = unpad(cipher.decrypt(ciphertext), AES.block_size)

return decrypted.decode('utf-8')
```

Рисунок 3.21 – Розшифрування повідомлення та повернення текстового результату

Завдяки такому підходу, реалізовано надійну симетричну криптографічну обгортку, придатну для поєднання зі стеганографічними методами – зокрема, для безпечного вбудовування зашифрованого повідомлення в структуру електронної книги.

3.3 Побудова графічного інтерфейсу програмного засобу

Для забезпечення зручності роботи середньостатистичного користувача необхідно створити інтуїтивно зрозумілий інтерфейс, що спрощує сприйняття та взаємодію з програмним засобом. Зазвичай це досягається розробкою простого і

зрозумілого графічного інтерфейсу користувача, який повністю задовольняє основні потреби.

У якості інструменту для створення такого інтерфейсу було обрано бібліотеку tkinter, яка надає всі необхідні засоби для побудови простого, зручного та сучасного візуального середовища [30]. Перевагами tkinter є її легкість у використанні, вбудована підтримка у стандартній бібліотеці Python, що дозволяє не встановлювати додаткове програмне забезпечення, а також можливість швидко створювати кросплатформенні графічні інтерфейси з мінімальними затратами ресурсів.

Під час розробки цього розділу кваліфікаційної роботи було створено графічний інтерфейс, який відповідає принципам простоти, мінімалізму та функціональності. Побудований інтерфейс зображено на рис. 3.22.

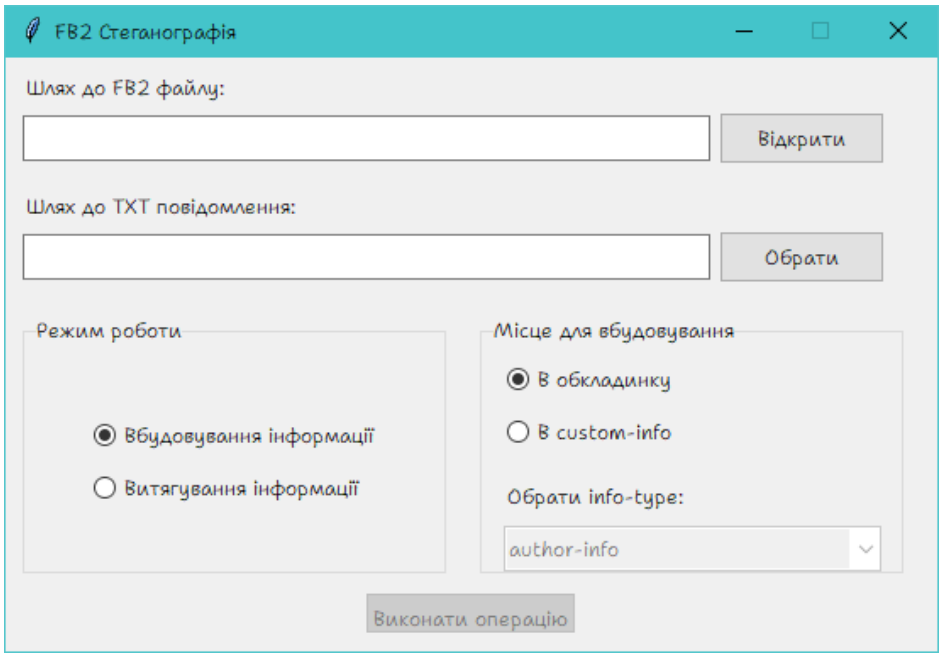

Рисунок 3.22 – Загальний вигляд графічного інтерфейсу програмного засобу

Були створенні вікна для введення 128 бітного ключу шифрування для алгоритму AES (рис. 3.23) та відображення результатів операцій витягування (рис. 3.24-3.25).

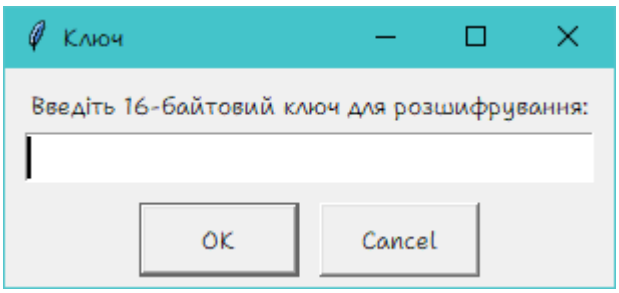

Рисунок 3.23 – Вікно для введення ключу шифрування

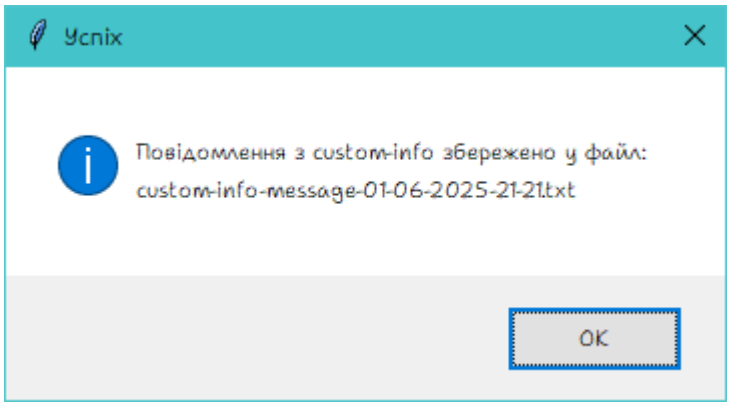

Рисунок 3.24 – Результат витягування інформації з елементу custom-info

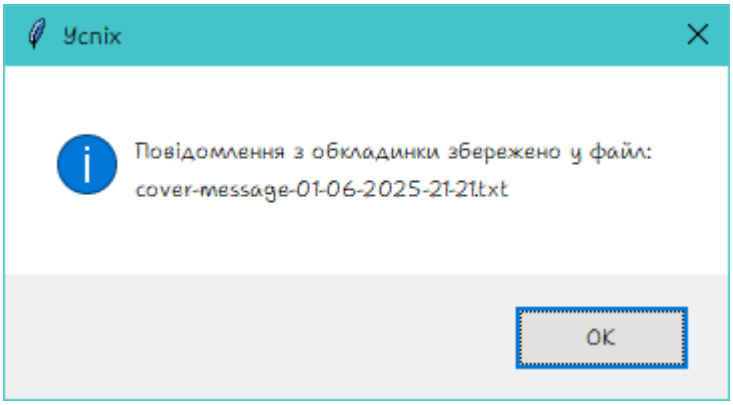

Рисунок 3.25 – Результат витягування інформації з обкладинки

Хоча графічний застосунок створювався з акцентом на простоту і виключення зайвих, непотрібних функцій, усе ж було впроваджено низку корисних механізмів для запобігання помилкам і забезпечення безперебійної роботи користувача з програмою. Зокрема, була реалізована функція, що блокує запуск стеганографічних операцій без вибору файлу електронної книги, а для операції вбудовування – ще й без зазначення файлу з прихованим повідомленням

(рис 3.26). Це суттєво знижує ризик помилок і робить роботу з застосунком надійнішою та зручнішою.

```
state = "disabled"

if not self.fb2_path:
    state = "disabled"
else:
    if self.mode_var.get() == "embed":
        embed_option = self.embed_option_var.get()

        if embed_option == "cover":
            # Потрібні обидва файли: fb2 і message (txt)
            state = "normal" if self.message_path else "disabled"

        elif embed_option == "custom":
            # Потрібні fb2, message (txt) і вибраний тип
            if self.message_path and self.combo_info_types.get():
                state = "normal"
            else:
                state = "disabled"

    elif self.mode_var.get() == "extract":
        # Для вилучення потрібен лише fb2
        state = "normal"
```

Рисунок 3.26 – Перевірка вводу шляху до використаних файлів

Також можна відмітити розроблену функцію, яка не дозволяє здійснювати певні види стеганографічних операцій, в залежності від наявності або відсутності тих чи інших елементів в електронній книзі (рис 3.27). До даних елементів відносяться посилання або файл обкладинки та наявність елементу <custom–info>.

```
has_embed_custom, cover_id = check_structure(self.fb2_path, self.info_types)
# Активуємо обидва радіокнопки, потім блокуємо за умовами
self.embed_cover_rb.config(state="normal")
self.embed_custom_rb.config(state="normal")
if not cover_id or not extract_cover_image(self.fb2_path, cover_id, "cover.png"):
    self.embed_cover_rb.config(state="disabled")
if has_embed_custom:
    self.embed_custom_rb.config(state="disabled")
self.mode_var.set("embed")
self.on_mode_change()
self.update_execute_button()
```

Рисунок 3.27 – Перевірка наявності елементів та блокування відповідних операцій

Також було впроваджено використання маркерів або маскуючих атрибутів як унікальних ідентифікаторів вбудованого повідомлення. Це дозволяє запобігти спробам витягнути приховану інформацію з електронної книги, якщо таких маркерів немає, підвищуючи надійність захисту та цілісність даних.

У випадку з текстовим повідомленням, яке підлягає вбудовуванню в зображення обкладинки, було використано текстовий маркер [[steg]] на початку повідомлення (рис 3.28).

```
message = extract_message_from_image("cover.png")
if message and message.startswith("[[STEG]]"):
    clean_message = message[len("[[STEG]]"):]
```

Рисунок 3.28 – Перевірка стеганографічного маркеру в змісті електронної книжки

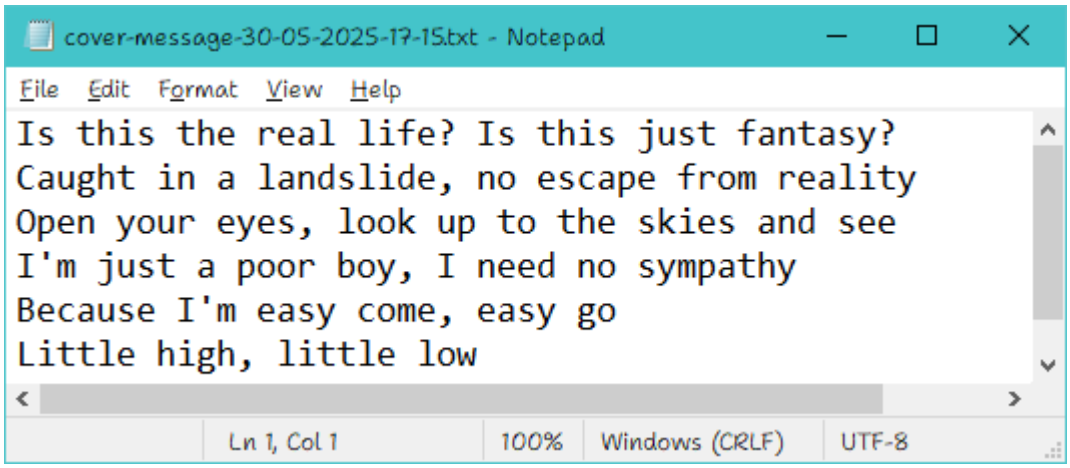

Рисунок 3.29 – Результат витягнення повідомлення

А при спробі витягнути повідомлення з елементу <custom–info> здійснюється перевірка вмісту атрибуту info–type. Якщо він відповідає прикладам вказаним в програмі, повідомлення витягується.

Висновок до розділу 3

У третьому розділі було здійснено практичну реалізацію методів приховування інформації в електронних книгах формату FictionBook 2.0. На основі мови програмування Python реалізовано повнофункціональний програмний засіб, здатний обробляти XML–структуру документів FB2, зчитувати та модифікувати графічні та текстові компоненти книги. Застосовано бібліотеки ElementTree, OpenCV, Cryptodome та tkinter, що забезпечило комплексну взаємодію з вмістом документа та зручний графічний інтерфейс.

Реалізовано функції для коректної обробки простору імен XML–документів, зчитування та конвертації зображення обкладинки, а також вбудовування текстових повідомлень у структурні елементи книги. Основу методу стеганографії склав алгоритм LSB з модифікацією останніх двох бітів зображення, а для криптографічного захисту застосовано симетричний шифр AES–128 у режимі CBC. Таке поєднання дозволяє досягти високого рівня безпеки, конфіденційності та непомітності прихованої інформації.

Окрема увага приділялася впровадженню механізмів захисту від помилок користувача та забезпеченню цілісності даних. Таким чином, у результаті реалізації було створено ефективний інструмент для стеганографічного та криптографічного приховування інформації у форматі FictionBook, що підтверджує практичну доцільність та функціональність запропонованого підходу.

ВИСНОВКИ

У межах виконаної кваліфікаційної роботи було досліджено та реалізовано методи вбудовування прихованої інформації в електронні книги формату FB2 із використанням методів цифрової стеганографії. Основною метою роботи була розробка методів і програмного забезпечення для прихованого вбудовування даних у структуру електронних видань, що досягнута шляхом аналізу формату FB2, вибору відповідних стеганографічних методів та практичної реалізації отриманих рішень.

У процесі дослідження:

– Проведено детальний аналіз електронних книжок як потенційного носія прихованої інформації;

– Визначено найбільш придатні структурні елементи FB2–документів для стеганографічного вбудовування, зокрема елементи <custom–info> та <binary>;

– Обґрунтовано вибір методу LSB як основного для роботи зі зображеннями обкладинок;

– Впроваджено алгоритм шифрування повідомлення за допомогою AES–128 перед його вбудовуванням в елемент користувацької інформації;

– Побудовано формалізовані алгоритми вбудовування та витягування прихованої інформації;

– Реалізовано програмний прототип, що автоматизує процес стеганографічного вбудовування та витягування, включно з графічним інтерфейсом користувача;

– Проведено тестування розробленої системи, що підтвердило збереження функціональності FB2–файлу, візуальної непомітності змін та точність відновлення повідомлення.

Практична цінність роботи полягає у створенні інструменту, який може бути застосований як для забезпечення конфіденційного обміну інформацією,

так і для маркування електронних видань з метою захисту авторських прав та виявлення джерел витоку.

У цілому, результати дослідження підтверджують доцільність використання електронних книг формату FB2 як об’єкта для стеганографії, а також демонструють можливість реалізації ефективних алгоритмів прихованого вбудовування, що не порушують функціональності та структури документа.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. FictionBook Options. Quarto [Електронний ресурс]. – Режим доступу: https://quarto.org/docs/reference/formats/fb2.html. – Дата звернення: 12.05.2025.

2. XML Tree. W3Schools [Електронний ресурс]. – Режим доступу: https://www.w3schools.com/xml/xml_tree.asp. – Дата звернення: 12.05.2025.

3. FictionBook.xsd. GitHub [Електронний ресурс]. – Режим доступу: https://github.com/gribuser/fb2/blob/master/FictionBook.xsd. – Дата звернення: 13.05.2025.

4. What is an e–Book?. Twinkl [Електронний ресурс]. – Режим доступу: https://www.twinkl.com/teaching–wiki/e–book. – Дата звернення: 01.05.2025.

5. Inspirational women: Angela Ruiz Robles. Code Week [Електронний ресурс]. – Режим доступу: https://codeweek.eu/blog/inspirational–women–angela–ruiz–robles/. – Дата звернення: 01.05.2025.

6. Michael S. Hart. EBSCO Information Services [Електронний ресурс]. – Режим доступу: https://www.ebsco.com/research–starters/biography/michael–s–hart. – Дата звернення: 01.05.2025.

7. About Project Gutenberg. Project Gutenberg [Електронний ресурс]. – Режим доступу: https://gutenberg.org/about/. – Дата звернення: 03.05.2025.

8. What is Plain Text? A Beginner's Guide. Lenovo [Електронний ресурс]. – Режим доступу: https://www.lenovo.com/us/en/glossary/plain–text/. – Дата звернення: 04.05.2025.

9. HTML History. W3Schools Tutorials [Електронний ресурс]. – Режим доступу: https://www.w3schools.in/html/history/. – Дата звернення: 04.05.2025.

10. A Brief History of PDF. Parsio Blog [Електронний ресурс]. – Режим доступу: https://parsio.io/blog/a–brief–history–of–pdf/. – Дата звернення: 05.05.2025.

11. A very short history of ebooks. Digital Publishing 101 [Електронний ресурс]. – Режим доступу: https://digitalpublishing101.com/digital–publishing–101/digital–publishing–basics/a–very–short–history/. – Дата звернення: 05.05.2025.

12. EPUB 3 Community Group. W3C [Електронний ресурс]. – Режим доступу: https://www.w3.org/community/epub3/. – Дата звернення: 06.05.2025.

13. Ausderau P. Protection mechanisms for electronic books. Theseus [Електронний ресурс]. – Режим доступу: https://www.theseus.fi/bitstream/handle/10024/74667/Ausderau_Patrick.pdf. – Дата звернення: 20.05.2025.

14. Field Guide to Fonts for Ebooks. Book Industry Study Group [Електронний ресурс]. – Режим доступу: https://www.bisg.org/products/field–guide–to–fonts–for–ebooks. – Дата звернення: 07.05.2025.

15. Кузнецов О. О., Євсеєв С. П., Король О. Г. Стеганографія: навч. посіб. ХНЕУ ім. С. Кузнеця [Електронний ресурс]. – Режим доступу: https://repository.hneu.edu.ua/bitstream/123456789/2289/1/Стеганография.pdf. – Дата звернення: 09.05.2025.

16. Buchwald P., Rostanski M., Maczka K. Network steganography method for user’s identity confirmation in web applications. PAS Journals [Електронний ресурс]. – Режим доступу: https://journals.pan.pl/Content/118525/PDF/Network%20steganography%20method.pdf?handler=pdf. – Дата звернення: 14.05.2025.

17. Watermark для цифрового контенту. Webpromo [Електронний ресурс]. – Режим доступу: https://web–promo.ua/ua/blog/watermark–dlya–cifrovogo–kontentu/. – Дата звернення: 12.05.2025.

18. Stegware (Steganography Malware) – Malware of the Month. Spanning [Електронний ресурс]. – Режим доступу: https://www.spanning.com/blog/stegware–steganography–malware–malware–of–the–month/. – Дата звернення: 11.05.2025.

19. Johnson N. F., Jajodia S. Steganalysis: the investigation of hidden information. IEEE Explore [Електронний ресурс]. – Режим доступу: https://ieeexplore.ieee.org/abstract/document/713394. – Дата звернення: 12.05.2025.

20. Description – FB2. MobileRead [Електронний ресурс]. – Режим доступу: https://wiki.mobileread.com/wiki/FB2#description_2. – Дата звернення: 13.05.2025.

21. Binary – FB2. MobileRead [Електронний ресурс]. – Режим доступу: https://wiki.mobileread.com/wiki/FB2#Binary. – Дата звернення: 13.05.2025.

22. Handrizal, Tarigan J. T., Putra D. I. Implementation of Steganography Modified Least Significant Bit using the Columnar Transposition Cipher and Caesar Cipher Algorithm in Image Insertion. Journal of Physics: Conference Series – 2021. – Vol. 1898, no. 1. – P. 012003. – DOI: https://doi.org/10.1088/1742–6596/1898/1/012003

23. Марценюк О. А. Стеганографічні методи приховування інформації в зображеннях. VI Всеукр. студент. наук.-техн. конф. "Природничі та гуманітарні науки" [Електронний ресурс]. – Режим доступу: https://elartu.tntu.edu.ua/bitstream/123456789/9567/2/Conf_2013v1_Martseniuk_O_A–Stehanohrafichni_metody_92.pdf. – Дата звернення: 17.05.2025.

24. Majid W. LSB Based Steganography, an Analysis Spatial and Frequency Domain – 2010. – Режим доступу: https://www.researchgate.net/publication/327704050_LSB_Based_Steganography_an_Analysis_Spatial_and_Frequency_Domain.

25. Marvel L. M., Boncelet C. G., Retter C. T. Spread spectrum image steganography. IEEE Transactions on Image Processing – 1999. – Т. 8, № 8. – С. 1075–1083. – DOI: https://doi.org/10.1109/83.777088

26. Що нового в Python. Python documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/uk/3.13/whatsnew/index.html. – Дата звернення: 21.05.2025.

27. xml.etree.ElementTree – The ElementTree XML API. Python documentation [Електронний ресурс]. – Режим доступу:

https://docs.python.org/3/library/xml.etree.elementtree.html. – Дата звернення: 21.05.2025.

28. About. OpenCV [Електронний ресурс]. – Режим доступу: https://opencv.org/. – Дата звернення: 21.05.2025.

29. pycryptodome. PyPI [Електронний ресурс]. – Режим доступу: https://pypi.org/project/pycryptodome/. – Дата звернення: 22.05.2025.

30. Python Tkinter. GeeksforGeeks [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/python–gui–tkinter/. – Дата звернення: 23.05.2025.

ДОДАТКИ

Додаток А

ФАЙЛ ГРАФІЧНОГО ІНТЕРФЕЙСУ

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import atexit
from fb2 import *
from stego import *
class FB2StegoApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("FB2 Стеганографія")
        self.geometry("550x365")
        self.resizable(False, False)
        self.fb2_path = None
        self.message_path = None
        self.info_types = [
            "author–info", "publisher–info", "review–info",
            "history–info", "source–info", "document–info"
        ]
        self.create_widgets()
        self.update_execute_button()  # Ініціалізація стану кнопки
        atexit.register(cleanup_temp_files)
    def create_widgets(self):
        top_frame = ttk.Frame(self)
        top_frame.pack(fill=tk.X, padx=10, pady=5)
        # FB2 файл
        ttk.Label(top_frame, text="Шлях до FB2 файлу:").pack(anchor=tk.W)
 fb2_path_row = ttk.Frame(top_frame)
        fb2_path_row.pack(fill=tk.X, pady=2)
        self.entry_path = ttk.Entry(fb2_path_row, width=50)
        self.entry_path.pack(side=tk.LEFT, padx=(0, 5))
        btn_browse = ttk.Button(fb2_path_row, text="Відкрити", command=self.load_fb2_file)
        btn_browse.pack(side=tk.LEFT)
        # TXT повідомлення
        ttk.Label(top_frame, text="Шлях до TXT повідомлення:").pack(anchor=tk.W, pady=(10, 0))
        msg_path_row = ttk.Frame(top_frame)
        msg_path_row.pack(fill=tk.X, pady=2)
        self.entry_msg_path = ttk.Entry(msg_path_row, width=50)
        self.entry_msg_path.pack(side=tk.LEFT, padx=(0, 5))
        self.btn_msg_browse = ttk.Button(msg_path_row, text="Обрати", command=self.load_txt_file)
        self.btn_msg_browse.pack(side=tk.LEFT)
        # Основна частина – режим і місце
        main_frame = ttk.Frame(self)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
        mode_frame = ttk.LabelFrame(main_frame, text="Режим роботи", width=250, height=155)
        mode_frame.pack_propagate(False)
        mode_frame.pack(side=tk.LEFT, anchor=tk.NW, padx=(0, 20))
        self.mode_var = tk.StringVar(value="embed")
        mode_inner = ttk.Frame(mode_frame)
        mode_inner.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        rb_embed = ttk.Radiobutton(mode_inner, text="Вбудовування інформації", variable=self.mode_var,
value="embed", command=self.on_mode_change)
        rb_extract = ttk.Radiobutton(mode_inner, text="Витягування інформації", variable=self.mode_var,
value="extract", command=self.on_mode_change)
        rb_embed.pack(anchor=tk.W, pady=2)
        rb_extract.pack(anchor=tk.W, pady=2)
```

```
        embed_frame = ttk.LabelFrame(main_frame, text="Місце для вбудовування", width=250, height=155)
        embed_frame.pack_propagate(False)
        embed_frame.pack(side=tk.LEFT, anchor=tk.NW)
        embed_inner = ttk.Frame(embed_frame)
        embed_inner.place(relx=0.5, rely=0.5, anchor=tk.CENTER)
        self.embed_option_var = tk.StringVar(value="cover")
        self.embed_cover_rb = ttk.Radiobutton(embed_inner, text="В обкладинку", variable=self.embed_option_var,
value="cover", command=self.on_embed_option_change)
        self.embed_custom_rb = ttk.Radiobutton(embed_inner, text="В custom–info", variable=self.embed_option_var,
value="custom", command=self.on_embed_option_change)
        self.embed_cover_rb.pack(anchor=tk.W, pady=2)
        self.embed_custom_rb.pack(anchor=tk.W, pady=2)
        ttk.Label(embed_inner, text="Обрати info–type:").pack(anchor=tk.W, pady=(10, 0))
        self.combo_info_types = ttk.Combobox(embed_inner, values=self.info_types, state="disabled", width=25)
        self.combo_info_types.current(0)
        self.combo_info_types.pack(pady=(5, 0))
        # Кнопка виконання операції
        action_frame = ttk.Frame(self)
        action_frame.pack(fill=tk.X, padx=10, pady=(0, 10))
        self.btn_execute = ttk.Button(action_frame, text="Виконати операцію", command=self.execute_operation)
        self.btn_execute.pack(anchor=tk.CENTER)
    def load_fb2_file(self):
        path = filedialog.askopenfilename(
            title="Оберіть FB2 файл",
            filetypes=[("FictionBook 2", "*.fb2")]
        )
        if not path:
            return
        self.fb2_path = path
        self.entry_path.delete(0, tk.END)
        self.entry_path.insert(0, path)
        has_embed_custom, cover_id = check_structure(self.fb2_path, self.info_types)
        # Активуємо обидва радіокнопки, потім блокуємо за умовами
        self.embed_cover_rb.config(state="normal")
        self.embed_custom_rb.config(state="normal")
        if not cover_id or not extract_cover_image(self.fb2_path, cover_id, "cover.png"):
            self.embed_cover_rb.config(state="disabled")
        if has_embed_custom:
        self.embed_custom_rb.config(state="disabled")
        self.mode_var.set("embed")
        self.on_mode_change()
        self.update_execute_button()
    def load_txt_file(self):
        path = filedialog.askopenfilename(
            title="Оберіть TXT файл",
            filetypes=[("Text Files", "*.txt")]
        )
        if not path:
            return
        self.message_path = path
        self.entry_msg_path.delete(0, tk.END)
        self.entry_msg_path.insert(0, path)
        self.update_execute_button()
    def on_mode_change(self):
        mode = self.mode_var.get()
        if mode == "embed":
            self.embed_cover_rb.config(state="normal")
            self.embed_custom_rb.config(state="normal")
            self.entry_msg_path.config(state="normal")
            self.btn_msg_browse.config(state="normal")
            self.on_embed_option_change()
```

```
    else:
        self.embed_cover_rb.config(state="disabled")
        self.embed_custom_rb.config(state="disabled")
        self.combo_info_types.config(state="disabled")
        self.entry_msg_path.config(state="disabled")
        self.btn_msg_browse.config(state="disabled")
    self.update_execute_button()
def on_embed_option_change(self):
    option = self.embed_option_var.get()
    if option == "custom":
        self.combo_info_types.config(state="readonly")
    else:
        self.combo_info_types.config(state="disabled")
    self.update_execute_button()
def update_execute_button(self):
    state = "disabled"

    if not self.fb2_path:
        state = "disabled"
    else:
        if self.mode_var.get() == "embed":
            embed_option = self.embed_option_var.get()
            if embed_option == "cover":
                # Потрібні обидва файли: fb2 і message (txt)
                state = "normal" if self.message_path else "disabled"
            elif embed_option == "custom":
                # Потрібні fb2, message (txt) і вибраний тип
                if self.message_path and self.combo_info_types.get():
                    state = "normal"
                else:
                    state = "disabled"
        elif self.mode_var.get() == "extract":
            # Для вилучення потрібен лише fb2
            state = "normal"
    self.btn_execute.config(state=state)
def execute_operation(self):
    import datetime
    from tkinter.simpledialog import askstring
    mode = self.mode_var.get()
    try:
        with open(self.message_path, "r", encoding="utf–8") as f:
            message = f.read()
    except Exception as e:
        message = None  # fallback для extract
        if mode == "embed":
            messagebox.showerror("Помилка", f"Не вдалося зчитати повідомлення з файлу:\n{e}")
            return
    if mode == "embed":
        embed_option = self.embed_option_var.get()
        if embed_option == "cover":
            success = embed_message_in_image("cover.png", message)
            if success:
                # Отримуємо cover_id
                _, cover_id = check_structure(self.fb2_path, self.info_types)
                update_cover_binary(self.fb2_path, cover_id, "cover.png")
                messagebox.showinfo("Успіх", "Повідомлення успішно вбудовано в обкладинку.")
            else:
                messagebox.showerror("Помилка", "Не вдалося вбудувати повідомлення в зображення.")
        elif embed_option == "custom":
            key = askstring("Ключ", "Введіть 16–байтовий ключ:", show="*")
            if not key or len(key.encode("utf–8")) != 16:
```

```
                messagebox.showerror("Помилка", "Ключ має бути рівно 16 байтів (UTF–8).")
                return
            encrypted_message = encrypt_message(message, key)
            current_info_type = self.combo_info_types.get()
            add_custom_info(self.fb2_path, encrypted_message, current_info_type)
            messagebox.showinfo("Успіх", f"Повідомлення вбудовано в custom–info (тип: {current_info_type}).")
        elif mode == "extract":
            has_embed_custom, cover_id = check_structure(self.fb2_path, self.info_types)
            now_str = datetime.datetime.now().strftime("%d–%m–%Y–%H–%M")
            if cover_id and extract_cover_image(self.fb2_path, cover_id, "cover.png"):
                message = extract_message_from_image("cover.png")
                if message and message.startswith("[[STEG]]"):
                    clean_message = message[len("[[STEG]]"):]
                    filename = f"cover–message–{now_str}.txt"
                    with open(filename, "w", encoding="utf–8") as f:
                        f.write(clean_message)
                    messagebox.showinfo("Успіх", f"Повідомлення з обкладинки збережено у файл:\n{filename}")
                else:
                    messagebox.showwarning("Попередження", "Повідомлення не знайдено в обкладинці.")
            if has_embed_custom:
                key = askstring("Ключ", "Введіть 16–байтовий ключ для розшифрування:", show="*")
                if not key or len(key) != 16:
                    messagebox.showerror("Помилка", "Ключ має бути рівно 16 байтів (UTF–8).")
                    return
                encrypted_message = read_custom_info(self.fb2_path, self.info_types)
                if not encrypted_message:
                    messagebox.showwarning("Попередження", "Зашифроване повідомлення не знайдено в custom–info.")
                    return
                message = decrypt_message(encrypted_message, key)
                if not message:
                    messagebox.showerror("Помилка", "Не вдалося розшифрувати повідомлення.")
                    return
                filename = f"custom–info–message–{now_str}.txt"
                with open(filename, "w", encoding="utf–8") as f:
                    f.write(message)
                messagebox.showinfo("Успіх", f"Повідомлення з custom–info збережено у файл:\n{filename}")
def cleanup_temp_files():
    import os
    if os.path.exists("cover.png"):
        try:
            os.remove("cover.png")
            print("Файл cover.png видалено.")
        except Exception as e:
            print(f"Помилка при видаленні cover.png: {e}")
if __name__ == "__main__":
    app = FB2StegoApp()
    app.mainloop()
```

Додаток Б

ФАЙЛ ДЛЯ РОБОТИ ЗІ ЗМІСТОМ Е-КНИЖКИ

```
import xml.etree.ElementTree as ET
def get_namespace(root):
    """Автоматично визначаємо простір імен із кореневого елемента."""
    if root.tag.startswith("{"):
        return root.tag.split("}")[0].strip("{")
    return ""
def ns_tag(tag, ns):
    """Повертаємо тег із простором імен (при наявності)."""
    return f"{{{ns}}}{tag}" if ns else tag
def check_structure(file_path: str, info_types: list[str]) –> tuple[bool, str]:
    try:
        tree = ET.parse(file_path)
        root = tree.getroot()
        ns = get_namespace(root)
        custom_info_elements = root.findall(f".//{ns_tag('custom–info', ns)}")
        has_custom_info = any(
            'info–type' in elem.attrib and elem.attrib['info–type'] in info_types
            for elem in custom_info_elements
        )
        cover_image_ids = []
        for cover in root.findall(f".//{ns_tag('coverpage', ns)}"):
            for img in cover.findall(f".//{ns_tag('image', ns)}"):
                href = img.attrib.get('{http://www.w3.org/1999/xlink}href', "")
                if href.startswith('#'):
                    cover_image_ids.append(href[1:])
        cover_id = ""
        for binary in root.findall(f".//{ns_tag('binary', ns)}"):
            binary_id = binary.attrib.get("id", "")
            if binary_id in cover_image_ids:
                cover_id = binary_id
                break
        return has_custom_info, cover_id
    except ET.ParseError:
        print("Файл не є валідним XML.")
        return False, ""
def add_custom_info(file_path: str, message: str, info_type: str = "hash_user") –> bool:
    tree = ET.parse(file_path)
    root = tree.getroot()
    ns = get_namespace(root)
    description_elem = root.find(f".//{ns_tag('description', ns)}")
    custom_info = ET.Element(ns_tag("custom–info", ns))
    custom_info.set("info–type", info_type)
    custom_info.text = message
    description_elem.append(custom_info)
    tree.write(file_path, encoding="utf–8", xml_declaration=True)
    return True
def read_custom_info(file_path: str, info_types: list) –> str:
    tree = ET.parse(file_path)
    root = tree.getroot()
    ns = get_namespace(root)
    for custom_info in root.findall(f".//{ns_tag('custom–info', ns)}"):
        info_type = custom_info.attrib.get('info–type')
        if info_type in info_types and custom_info.text:
            return custom_info.text.strip()
    return ""
import base64
```

Продовження додатку Б

```
import cv2
import numpy as np
def extract_cover_image(file_path: str, cover_id: str, output_path: str) –> bool:
    tree = ET.parse(file_path)
    root = tree.getroot()
    ns = get_namespace(root)
    binary_elem = None
    for binary in root.findall(f".//{ns_tag('binary', ns)}"):
        if binary.attrib.get("id") == cover_id:
            binary_elem = binary
            break
    if binary_elem is None:
        print("Тег <binary> з id обкладинки не знайдено.")
        return False
    base64_data = binary_elem.text.strip()
    image_data = base64.b64decode(base64_data)
    nparr = np.frombuffer(image_data, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_UNCHANGED)
    if img is None:
        print("Не вдалося розпізнати зображення у binary.")
        return False
    if '.' in cover_id:
        output_filename = output_path or cover_id.rsplit('.', 1)[0] + '.png'
    else:
        output_filename = output_path or cover_id + '.png'
    success = cv2.imwrite(output_filename, img)
    if not success:
        print("Помилка при збереженні файлу.")
        return False
    return True
def update_cover_binary(file_path: str, cover_id: str, image_path: str) –> bool:
    try:
        image = cv2.imread(image_path)
        if image is None:
            print(f"Не вдалося відкрити зображення за шляхом: {image_path}")
            return False
        success, buffer = cv2.imencode(".png", image)
        if not success:
            print("Помилка при кодуванні зображення у PNG.")
            return False
        import textwrap
        base64_data = base64.b64encode(buffer).decode('utf–8')
        wrapped_data = "\n".join(textwrap.wrap(base64_data, 76))
        tree = ET.parse(file_path)
        root = tree.getroot()
        ns = get_namespace(root)
        binary_elem = None
        for binary in root.findall(f".//{ns_tag('binary', ns)}"):
            binary_id = binary.attrib.get("id", "").split(".")[0]
            cover_cleaned = cover_id.split(".")[0]
            if binary_id == cover_cleaned:
                binary_elem = binary
                break
        if binary_elem is None:
            print("Не знайдено тег <binary> з відповідним id.")
            return False
        binary_elem.text = wrapped_data
        binary_elem.set("content–type", "image/png")
        tree.write(file_path, encoding="utf–8", xml_declaration=True)
        print("Зображення успішно оновлено в тегу <binary>.")
        return True
```

Додаток В

ФАЙЛ З СТЕГАНОГРАФІЧНИМИ ФУНКЦІЯМИ

```
import cv2
import numpy as np
def embed_message_in_image(image_path: str, message: str) –> bool:
    try:
        img = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
        # Перевірка формату масиву і типу
        if img.dtype != np.uint8:
            img = img.astype(np.uint8)
        height, width, channels = img.shape
        # Додаємо маркер та null–термінатор
        message = "[[STEG]]" + message + chr(0)
        max_message_length = (height * width * channels * 2) // 8  # 2 біти на канал, 8 біт в 1 байті
        if len(message) > max_message_length:
            print(f"Повідомлення занадто довге. Максимум: {max_message_length – 1} символів.")
            return False
        # Конвертуємо повідомлення в список бітів
        message_bits = []
        for char in message:
            bits = format(ord(char), '08b')
            message_bits.extend([int(bit) for bit in bits])
        bit_index = 0
        total_bits = len(message_bits)
        for row in range(height):
            for col in range(width):
                for ch in range(channels):
                    if bit_index + 2 <= total_bits:
                        # Візьмемо 2 біти для запису
                        bits_to_write = (message_bits[bit_index] << 1) | message_bits[bit_index + 1]
                        # Очищаємо 2 молодших біти і записуємо нові
                        original_val = img[row, col, ch]
                        new_val = (original_val & 0b11111100) | bits_to_write
                        img[row, col, ch] = new_val
                        bit_index += 2
                    else:
                        # Всі біти вбудовані, зберігаємо і виходимо
                        cv2.imwrite(image_path, img)
                        return True
        # Якщо повідомлення довше, ніж вміщує зображення (теоретично не повинно бути)
        print("Недостатньо пікселів для вбудовування всього повідомлення.")
        return False
    except Exception as e:
        print(f"Помилка при вбудовуванні повідомлення: {e}")
        return False
def extract_message_from_image(cover_path: str) –> str:
    try:
        # Зчитуємо зображення
        image = cv2.imread(cover_path)
        if image is None:
            print("Не вдалося зчитати зображення.")
            return ""
        bits = []
        h, w, _ = image.shape
        # Витягуємо біти з кожного каналу пікселів
        for y in range(h):
```

```
            for x in range(w):
                pixel = image[y, x]
                for channel in range(3):  # B, G, R
                    bits_from_channel = pixel[channel] & 0b11  # беремо 2 молодших біти
                    bits.append((bits_from_channel >> 1) & 1)
                    bits.append(bits_from_channel & 1)
        # Групуємо біти в байти
        message_chars = []
        for i in range(0, len(bits), 8):
            byte_bits = bits[i:i + 8]
            if len(byte_bits) < 8:
                break
            byte = 0
            for bit in byte_bits:
                byte = (byte << 1) | bit
            if byte == 0:  # null–термінатор
                break
            message_chars.append(chr(byte))
        return ''.join(message_chars)
    except Exception as e:
        print(f"Помилка під час витягування повідомлення: {e}")
        return ""
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad
import base64
def encrypt_message(message: str, key: str) –> str:
    if len(key) != 16:
        raise ValueError("Ключ повинен мати довжину 16 символів (128 біт).")
    # Перетворення ключа та повідомлення у байти
    key_bytes = key.encode('utf–8')
    message_bytes = message.encode('utf–8')
    # Створення випадкового IV
    iv = get_random_bytes(16)
    # Ініціалізація шифру та шифрування
    cipher = AES.new(key_bytes, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(message_bytes, AES.block_size))
    # Комбінуємо IV та шифротекст
    combined = iv + ciphertext
    return base64.b64encode(combined).decode('utf–8')
from Crypto.Util.Padding import unpad
def decrypt_message(encrypted_data: str, key: str) –> str:
    # Перетворюємо ключ у байти
    key_bytes = key.encode('utf–8')
    # Перевірка довжини ключа
    if len(key) != 16:
        raise ValueError("Ключ повинен бути 16 символів (128 біт)")
    encrypted = base64.b64decode(encrypted_data)
    # Витягуємо IV та шифротекст
    iv = encrypted[:16]
    ciphertext = encrypted[16:]
    # Ініціалізація AES
    cipher = AES.new(key_bytes, AES.MODE_CBC, iv)
    # Розшифрування з видаленням паддінгу
    decrypted = unpad(cipher.decrypt(ciphertext), AES.block_size)
    return decrypted.decode('utf–8')
```