Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедри кібербезпеки
та захисту інформації
_____ Іван ПАРХОМЕНКО
«___» _____ 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)
спеціальність *125 Кібербезпека та захист інформації*
(код і назва спеціальності)
освітній ступень *магістр*
освітньо-наукова програма *Кібербезпека*

на тему: Метод автоматичного моніторингу цілісності файлів для захисту авторського права

Виконавець: студент ІІ курсу, групи КБм-21

_____ **Марк ГАВРИЩУК**
(підпис) (прізвище ім´я по-батькові)

	Прізвище, ініціали	**Підпис**
Науковий керівник	Леся БАРАНОВСЬКА	
Нормоконтроль	Юрій БАБЕНКО	

Київ 2025

Міністерство освіти і науки України
Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.О. завідувача кафедри
кібербезпеки та захисту інформації

_____ Іван ПАРХОМЕНКО

«25» жовтня 2024 року

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека та захист інформації*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача КБм-21 (група) Гаврищука Марка Вячеславовича (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Метод автоматичного моніторингу цілісності файлів для захисту авторського права

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 4 від 24.10.2024 р.

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Процес забезпечення цілісності файлів програмного забезпечення для захисту авторського права
Предмет досліджень	Методи автоматичного моніторингу цілісності файлів з використанням хеш-функцій та систем оповіщення.
Мета	Розробка методу автоматичного моніторингу цілісності файлів програмного забезпечення для захисту авторського права

Вихідні дані для проведення роботи	Методи захисту авторського права, алгоритми водяних знаків, дослідження використання блокчейну

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна	Інтеграція технології блокчейн до існуючих методів моніторингу цілісності файлів.
Практична цінність	Програма автоматичного моніторингу цілісності файлів з інтеграцією технології блокчейн та API VirusTotal

4. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	**Строки виконання робіт (початок-кінець)**
Уточнення постановки задачі	25.10.2024 – 29.12.2024
Аналіз літературних джерел	30.12.2024 – 12.02.2024
Дослідити існуючі методи забезпечення цілісності файлів.	13.02.2025 – 31.03.2025
Розробити метод автоматичної перевірки цілісності файлів з використанням хеш-функцій та блокчейну.	1.04.2025 – 14.04.2025
Створити базу даних для зберігання контрольних сум.	15.04.2025 – 18.04.2025
Інтегрувати технологію блокчейн до методу перевірки цілісності файлів.	19.04.2025 – 30.04.2025
Тестування розробленого методу на реальних даних	1.05.2025 – 11.05.2025
Оформлення пояснювальної записки згідно методичних рекомендацій	11.05.2024 – 15.05.2025
Подача пакету документів на розгляд ЕК	15.05.2025 – 19.05.2025

Завдання видала _____ Леся БАРАНОВСЬКА
(підпис) (прізвище, ініціали)

Завдання прийняв
до виконання _____ Марк ГАВРИЩУК
(підпис) (прізвище, ініціали)

Дата видачі завдання: 25.10.2024 р.

Термін подання дипломної роботи до ЕК 19.05.2025

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Метод автоматичного моніторингу цілісності файлів для захисту авторського права»: 93 сторінки, 2 таблиці, 23 рисунки, 52 літературних джерела.

Об’єктом дослідження є процес забезпечення цілісності файлів програмного забезпечення для захисту авторського права.

Метою роботи є розробка методу автоматичного моніторингу цілісності файлів програмного забезпечення для захисту авторського права.

Методи дослідження: аналіз існуючих підходів, моделювання хеш-функцій, вивчення, вивчення інтеграції блокчейн технології, аналіз можливостей API-взаємодії з VirusTotal.

Предметом дослідження є методи автоматичного моніторингу цілісності файлів з використанням хеш-функцій та систем оповіщення.

У роботі досліджено технологію блокчейн, водяні знаки, хеш-функції. Проведено аналіз існуючих методів моніторингу цілісності файлів.

Наукова новизна: інтеграція технології блокчейн до існуючих методів моніторингу цілісності файлів.

Практичною цінністю є програма автоматичного моніторингу цілісності файлів з інтеграцією технології блокчейн та API VirusTotal.

Результати здійснених у дипломній роботі досліджень можуть бути використані для покращення існуючих методів автоматичного моніторингу файлів та для захисту авторського права.

Для досягнення мети поставлено наступні завдання:

- Дослідити існуючі методи забезпечення цілісності файлів та їхні обмеження.
- Розробити метод автоматичної перевірки цілісності файлів з використанням хеш-функцій.
- Створити базу даних для зберігання контрольних сум файлів.

- Інтегрувати технологію блокчейн до методу перевірки цілісності файлів.
- Провести тестування розробленого методу на реальних даних для оцінки її ефективності.

Напрямки для подальших досліджень: використання блокчейну в інших технологіях пов’язаних з інформаційною та кібербезпекою.

Ключові слова: блокчейн, цілісність, моніторинг, хеш, водяний знак, інтерфейс.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ПЗ – Програмне забезпечення
API – Application Programming Interface
(D)DoS – (Distributed) Denial-of-Service
DMCA – Digital Millennium Copyright Act
WIPO – World Intellectual Property Organization
GDPR – General Data Protection Regulation
ХЦ – Хмарний центр
СК – Смарт-контракт

ЗМІСТ

ВСТУП

У сучасному цифровому світі програмне забезпечення (ПЗ) є одним із ключових елементів функціонування бізнесу, державних установ та приватних користувачів. Однак, з розвитком технологій зростають і ризики, пов’язані з порушенням цілісності файлів, що призводить до піратства, шкідливих модифікацій та втрати довіри до програмних продуктів. Захист авторського права на програмне забезпечення стає все більш важливим завданням, оскільки несанкціоноване копіювання, розповсюдження та модифікація ПЗ завдають значних економічних збитків розробникам і створюють загрозу для інформаційної безпеки користувачів.

Однією з ключових проблем у захисті авторського права є відсутність ефективних механізмів автоматичного моніторингу цілісності файлів. Існуючі методи захисту, такі як цифрові підписи або контрольні суми, часто вимагають ручного втручання або не забезпечують оперативного реагування на порушення. Це робить їх недостатньо ефективними в умовах, коли зловмисники використовують складні методи для обходу захисту. Крім того, багато розробників ПЗ не мають доступу до спеціалізованих інструментів для моніторингу цілісності файлів, що робить їх програмні продукти вразливими до атак.

Іншою проблемою є відсутність інтеграції механізмів перевірки цілісності з системами оповіщення. Навіть якщо зміни у файлах виявлено, відсутність оперативного сповіщення може призводити до затримки у реагуванні, що значно збільшує ризики для користувачів. Це особливо критично в умовах, коли шкідливі модифікації ПЗ можуть призводити до витоку конфіденційних даних, фінансових втрат або порушення роботи критично важливих систем.

У цьому контексті виникає необхідність у розробці нових методів захисту авторського права, які б поєднували автоматичний моніторинг цілісності файлів з оперативним оповіщенням про порушення. Такий підхід дозволить не лише виявляти несанкціоновані зміни у файлах ПЗ, але й забезпечити швидке реагування на потенційні загрози. Це особливо актуально для розробників програмного

забезпечення, які прагнуть захистити свої продукти від піратства та шкідливих модифікацій, а також для користувачів, які потребують гарантії безпеки використовуваного ПЗ.

Метою даної роботи є розробка методу автоматичного моніторингу цілісності файлів програмного забезпечення, який би забезпечував ефективний захист авторського права через інтеграцію механізму перевірки контрольних сум із системою оповіщення.

РОЗДІЛ 1
ТЕОРИТИЧНІ ВІДОМОСТІ ЩОДО ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕННЯ

1.1. Актуальність проблеми цифрового піратства та шкідливих модифікацій ПЗ

Цифрове піратство та шкідливі модифікації ПЗ мають серйозні економічні наслідки. Вони призводять до втрат доходів для розробників, знижують довіру до легального програмного забезпечення та збільшують витрати на боротьбу з кіберзлочинністю. Крім того, порушення авторських прав може призводити до правових наслідків для користувачів, які використовують піратське ПЗ. Наприклад, у багатьох країнах передбачені суворі санкції за використання неліцензійного програмного забезпечення, включаючи штрафи та конфіскацію обладнання.

Шкідливі модифікації ПЗ становлять серйозну загрозу для інформаційної безпеки. Вони можуть призводити до витоку конфіденційних даних, порушення роботи критично важливих систем та введення в оману користувачів. Наприклад, модифіковане ПЗ може містити бекдори, які дозволяють зловмисникам отримувати несанкціонований доступ до систем. Це особливо небезпечно для організацій, які використовують програмне забезпечення для управління критичною інфраструктурою, фінансовими операціями або особистими даними користувачів.

Цифрове піратство також має соціальні наслідки. Воно сприяє формуванню культури неуважності до інтелектуальної власності, що ускладнює боротьбу з порушеннями авторських прав. Крім того, використання піратського ПЗ може призводити до втрати довіри до цифрових технологій серед користувачів, особливо в умовах, коли вони стикаються з наслідками шкідливих модифікацій.

Україна, як країна з ринком програмного забезпечення, що розвивається, також стикається з проблемою цифрового піратства. Згідно з даними BSA, рівень піратства в Україні залишається високим, що створює додаткові виклики для розвитку

індустрії програмного забезпечення. Крім того, Україна є частиною європейського правового поля, що вимагає дотримання міжнародних стандартів захисту інтелектуальної власності, таких як GDPR та директиви ЄС.

Проблема цифрового піратства та шкідливих модифікацій ПЗ є надзвичайно актуальною у сучасному світі. Вона має серйозні економічні, правові, інформаційні та соціальні наслідки, які впливають як на розробників програмного забезпечення, так і на користувачів. У зв'язку з цим розробка ефективних методів захисту ПЗ від піратства та модифікацій є важливим завданням для забезпечення інформаційної безпеки та підтримки розвитку індустрії програмного забезпечення.

У сучасному цифровому середовищі, що характеризується високим рівнем комп'ютеризації, глобалізації інформаційних процесів і стрімким розвитком технологій, однією з найгостріших проблем є цифрове піратство. Цей феномен охоплює нелегальне копіювання, розповсюдження та використання програмного забезпечення без відповідного ліцензійного дозволу з боку правовласників. Цифрове піратство вважається одним із найбільш поширених та стійких викликів, що постає перед галуззю інформаційних технологій, зокрема індустрією розробки та поширення програмного забезпечення.

Особливо критичною є ситуація у країнах із ринками, що розвиваються, де недостатній рівень цифрової грамотності, обмежений доступ до якісного програмного забезпечення та економічні чинники сприяють процвітанню піратських практик. У деяких регіонах частка неліцензійного програмного забезпечення сягає катастрофічних показників це 70–80%. У такому середовищі легальні розробники та власники авторських прав втрачають контроль над поширенням своїх продуктів, а користувачі, своєю чергою, стають уразливими до численних кіберризиків.

Варто наголосити, що цифрове піратство це не лише правопорушення у сфері інтелектуальної власності. Воно також створює реальні загрози для інформаційної безпеки. Неліцензійне ПЗ часто супроводжується модифікаціями, які несанкціоновано вносяться до його коду сторонніми особами. У багатьох випадках мова йде про вбудування шкідливих компонентів, таких як троянські програми,

бекдори, кейлогери, програми для несанкціонованого майнінгу криптовалюти (криптомайнери) або шпигунські інструменти.

У світі є тенденція, коли популярне програмне забезпечення, поширюване через файлообмінні мережі або сумнівні вебресурси, містило приховані шкідливі модулі. Ці модифікації ставили під загрозу як приватні дані користувачів, так і стабільність роботи їхніх пристроїв. У результаті цього несанкціонованого втручання страждає цілісність програмного забезпечення, одна з ключових властивостей, що забезпечує надійне функціонування інформаційних систем.

Шкідливі модифікації ПЗ мають багатогранні наслідки.

По-перше, вони порушують довіру користувачів до технологій загалом.

По-друге, вони здатні спричиняти критичні інциденти в інфраструктурі підприємств та державних установ.

Програмне забезпечення, що використовується для обробки фінансових транзакцій, управління виробничими процесами або зберігання конфіденційної інформації, стає об'єктом особливого інтересу для зловмисників. Навіть незначне втручання у його код може призвести до витоку даних, блокування систем або інших небажаних ефектів.

Крім того, економічні наслідки цифрового піратства не обмежуються лише втратою прибутку. Поширення неліцензійного ПЗ зменшує мотивацію до інновацій серед розробників, оскільки ризик комерційної невдачі зростає внаслідок неможливості контролювати обсяги продажів.

У довгостроковій перспективі це може негативно впливати на динаміку технологічного прогресу. Також важливо враховувати витрати держави та компаній на боротьбу з наслідками піратства це створення систем цифрового захисту, впровадження ліцензійного моніторингу, проведення освітніх кампаній тощо.

Не можна оминути увагою і соціальний аспект проблеми. У країнах з високим рівнем цифрового піратства часто спостерігається низький рівень поваги до інтелектуальної власності як цінності.

Формується своєрідна культура нехтування авторськими правами, що ускладнює просування легального програмного забезпеченя та сприяє легітимізації піратських практик у свідомості суспільства.

Україна, як держава, що активно інтегрується у європейський цифровий простір, також стикається з численними викликами у цій сфері. Згідно з тими ж даними BSA, рівень використання неліцензійного ПЗ в Україні залишається на підвищеному рівні, що становить реальну загрозу для стабільного розвитку вітчизняного ринку інформаційних технологій.

При цьому, українське законодавство зобов'язане відповідати європейським стандартам, таким як Загальний регламент захисту даних (GDPR) або директиви ЄС щодо цифрового ринку, що створює додаткові юридичні зобов'язання для державних структур та бізнесу.

Таким чином, цифрове піратство та шкідливі модифікації ПЗ є не лише технічною або юридичною проблемою, а й багатовимірним явищем, що охоплює економічну, соціальну, культурну та безпекову площини.

Для мінімізації наслідків цього явища необхідна комплексна стратегія, яка включає освітню роботу з населенням, технічні заходи з виявлення та попередження загроз, а також посилення юридичної відповідальності за використання неліцензійного програмного забезпечення. Усе це повинно відбуватися в рамках більш широкої політики кібербезпеки, спрямованої на підвищення рівня довіри до цифрових рішень і збереження конкурентоздатності галузі програмного забезпечення.

1.2 Механізми забезпечення цілісності файлів та перевірка цифрового підпису

Механізми забезпечення цілісності файлів

Забезпечення цілісності файлів є одним із ключових аспектів кібербезпеки, особливо в контексті захисту програмного забезпечення (ПЗ) та фінансових даних. Цілісність файлів означає, що дані не були змінені, пошкоджені або підроблені під

час зберігання, передачі чи обробки. Для забезпечення цілісності використовуються різні механізми, серед яких найпоширенішими є хеш-функції, цифрові підписи та блокчейн-технології.

Хеш-функції це математичні алгоритми, які перетворюють дані будь-якого розміру в унікальний рядок фіксованої довжини, який називається хешем. Наприклад, алгоритм SHA-256 генерує хеш довжиною 256 біт. Якщо хоча б один біт у файлі змінюється, хеш також змінюється, що дозволяє легко виявити порушення цілісності. Хеш-функції широко використовуються для перевірки цілісності файлів, зокрема в системах контролю версій (наприклад, Git) та для забезпечення безпеки передачі даних через мережу [1].

Цифрові підписи це механізм, який дозволяє перевірити не лише цілісність файлу, але й автентичність його джерела.

Цифровий підпис створюється за допомогою асиметричного шифрування: відправник використовує свій приватний ключ для підпису файлу, а отримувач може перевірити підпис за допомогою відкритого ключа. Цей механізм забезпечує високу надійність, оскільки підробка цифрового підпису є практично неможливою без доступу до приватного ключа [2].

Блокчейн-технології також можуть бути використані для забезпечення цілісності файлів. У цьому випадку хеш файлу зберігається в блокчейні, що робить його незмінним і захищеним від підробки. Це особливо корисне для захисту авторського права, оскільки дозволяє фіксувати час створення та змін файлів [3].

Перевірка цифрового підпису

Перевірка цифрового підпису включає кілька етапів:

1. Генерація хешу: відправник обчислює хеш файлу за допомогою хеш-функції (наприклад, SHA-256).

2. Шифрування хешу: хеш шифрується приватним ключем відправника, що створює цифровий підпис.

3. Перевірка підпису: отримувач розшифровує підпис за допомогою відкритого ключа відправника та порівнює отриманий хеш із хешем файлу. Якщо хеші збігаються, це означає, що файл не був змінений і дійсно належить відправнику [4].

Процес роботи цифрового підпису зображений на рисунку 1.1.

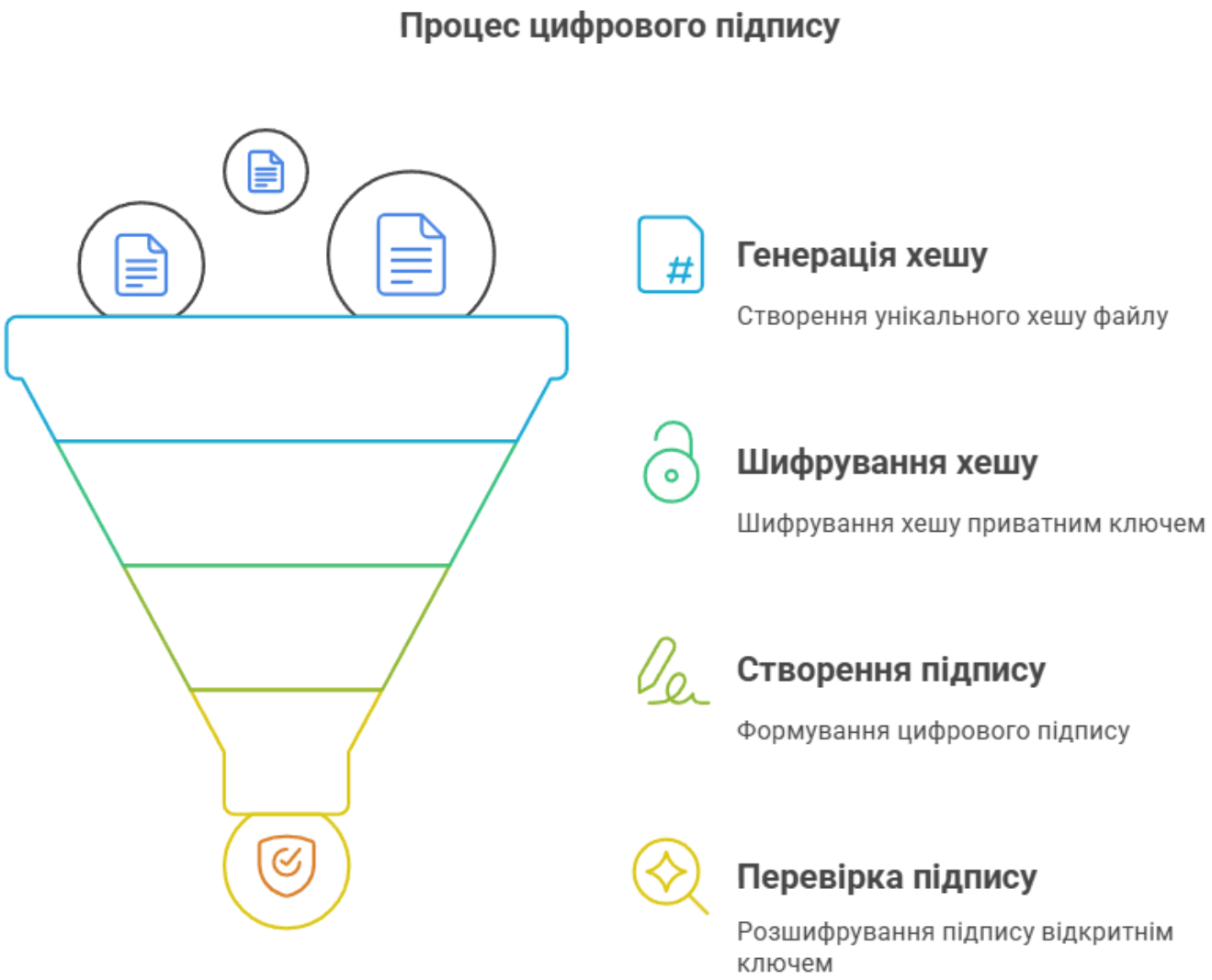

Рисунок 1.1 – Процес цифрового підпису

Цифровий підпис працює наступним чином:

1. Вхідні дані: файл, приватний ключ відправника, відкритий ключ отримувача.

2. Генерація хешу: файл → хеш-функція → хеш.

3. Шифрування хешу: хеш + приватний ключ → цифровий підпис.

4. Перевірка підпису: цифровий підпис + відкритий ключ → розшифрований хеш.

5. Порівняння хешів: якщо хеші збігаються, файл цілісний і автентичний [5].

1.3 Законодавчі вимоги та стандарти щодо захисту ПЗ (DMCA, WIPO, GDPR, ISO 27001)

DMCA (Digital Millennium Copyright Act)

Digital Millennium Copyright Act (DMCA) це закон, прийнятий у США у 1998 році, який регулює захист авторських прав у цифровій сфері. Основна мета DMCA запобігти несанкціонованому копіюванню та розповсюдженню цифрового контенту, такого як програмне забезпечення, музика, фільми та інші медіафайли. Закон забороняє обхід технічних засобів захисту (DRM Digital Rights Management), які використовуються для захисту авторських прав. Наприклад, якщо розробник ПЗ встановлює захист від копіювання, DMCA забороняє будь-які спроби зламати цей захист [6].

Однією з ключових особливостей DMCA є процедура "notice and takedown" (повідомлення та видалення). Ця процедура дозволяє власникам авторських прав повідомляти інтернет-платформи про порушення, після чого платформи зобов’язані видалити контент, який порушує авторські права. Наприклад, якщо хтось завантажує піратську копію програмного забезпечення на YouTube, власник ПЗ може подати скаргу, і YouTube зобов’язаний видалити цей контент. DMCA також передбачає суворі санкції за порушення, включаючи штрафи та позови в суді.

WIPO (World Intellectual Property Organization)

World Intellectual Property Organization (WIPO) це міжнародна організація, яка займається захистом інтелектуальної власності, включаючи авторські права, патенти та торгові марки. WIPO була заснована у 1967 році і налічує понад 190 країн-учасниць. Однією з головних цілей WIPO є розробка міжнародних стандартів та угод, які регулюють захист інтелектуальної власності у всьому світі [7].

Однією з найважливіших угод, розроблених WIPO, є Бернська конвенція. Ця конвенція встановлює мінімальні стандарти захисту авторських прав, які повинні дотримуватися країнами-учасницями. Наприклад, конвенція передбачає, що авторські права автоматично належать творцю з моменту створення твору, і немає необхідності реєструвати твір для отримання захисту. WIPO також надає технічну допомогу країнам, які розвиваються, для впровадження ефективних механізмів захисту інтелектуальної власності .

GDPR (General Data Protection Regulation)

General Data Protection Regulation (GDPR) це регламент Європейського Союзу, який регулює захист персональних даних. GDPR був прийнятий у 2016 році і набрав чинності у 2018 році. Основна мета GDPR забезпечити високий рівень захисту персональних даних громадян ЄС, незалежно від того, де ці дані обробляються. GDPR вимагає від організацій забезпечувати цілісність, конфіденційність та доступність даних, зокрема через використання шифрування, хеш-функцій та інших технчних засобів [8].

Однією з ключових вимог GDPR є принцип мінімізації даних. Цей принцип передбачає, що організації повинні збирати лише ті дані, які необхідні для виконання конкретних цілей. Наприклад, якщо веб-сайт збирає дані користувачів для реєстрації, він не повинен збирати зайву інформацію, таку як дата народження або адреса, якщо це не є необхідним. GDPR також передбачає суворі санкції за порушення, включаючи штрафи до 20 мільйонів євро або 4% від глобального обороту компанії.

ISO 27001

ISO 27001 — це міжнародний стандарт інформаційної безпеки, який визначає вимоги до системи управління інформаційною безпекою (СУІБ). Стандарт був розроблений Міжнародною організацією зі стандартизації (ISO) і надає рекомендації щодо забезпечення цілісності, конфіденційності та доступності даних. ISO 27001 включає вимоги до проведення аналізу ризиків, впровадження заходів захисту та регулярного аудиту системи безпеки [9].

Однією з ключових переваг ISO 27001 є його універсальність. Стандарт може бути застосований до будь-якої організації, незалежно від її розміру чи галузі. Наприклад, банки можуть використовувати ISO 27001 для захисту фінансових даних, а IT-компанії, тобто для захисту програмного забезпечення. Сертифікація за ISO 27001 є важливим кроком для організацій, які прагнуть забезпечити високий рівень кібербезпеки та довіру з боку клієнтів .

1.4 Блокчейн-технологія як інструмент перевірки цілісності

Принцип роботи блокчейну полягає у збергіанні даних у вигляді ланцюгу блоків з даними.

Блокчейн це технологія, яка дозволяє зберігати дані у вигляді ланцюжка блоків. Кожен блок містить інформацію про певні дані (наприклад, транзакції або файли), хеш попереднього блоку та власний хеш. Хеш це унікальний ідентифікатор, який генерується за допомогою математичного алгоритму. Якщо дані в блоці змінюються, хеш блоку також змінюється, що порушує ланцюжок і робить зміни очевидними [10].

Однією з ключових особливостей блокчейну є його децентралізація. На відміну від традиційних баз даних, які зберігаються на центральному сервері, блокчейн розподілений між тисячами комп’ютерів по всьому світу. Це робить систему більш стійкою до атак, оскільки зловмисник не може змінити дані, не змінивши їх на всіх комп’ютерах одночасно .

Використання блокчейну для забезпечення цілісності файлів.

Блокчейн може бути використаний для забезпечення цілісності файлів шляхом фіксації їхніх хешів у блокчейні. Це може бути корисним у випадках де потрібно використати цілісність файлів для захисту цілісності файлів. Дана технологія може допомогти з великою точністю у перевірці цілісності даних. Також блокчейн дозволяє моніторити зміни, враховуючи особливості цієї технології. Відсліковування змін може збільшити ефективність реагування на порушення цілісності даних чи

файлів. Це може знизити ризики інцидентів інформаційної безпеки позв'язаних з порушення цілісності фалйів.

Наприклад, якщо розробник програмного забезпечення хоче захистити свою програму від підробки, він може згенерувати хеш файлу та зберегти його в блокчейні. Таким чином він може відсліковувати зміни. Це допоможе йому у випадку якщо зловмисник одного разу спробує наприклад змінити файл чи пошкодити цілісність даних, хеш зміниться, і це буде легко виявити [11]. Процес забезпечення цілісності файлів зображений на рисунку 1.2.

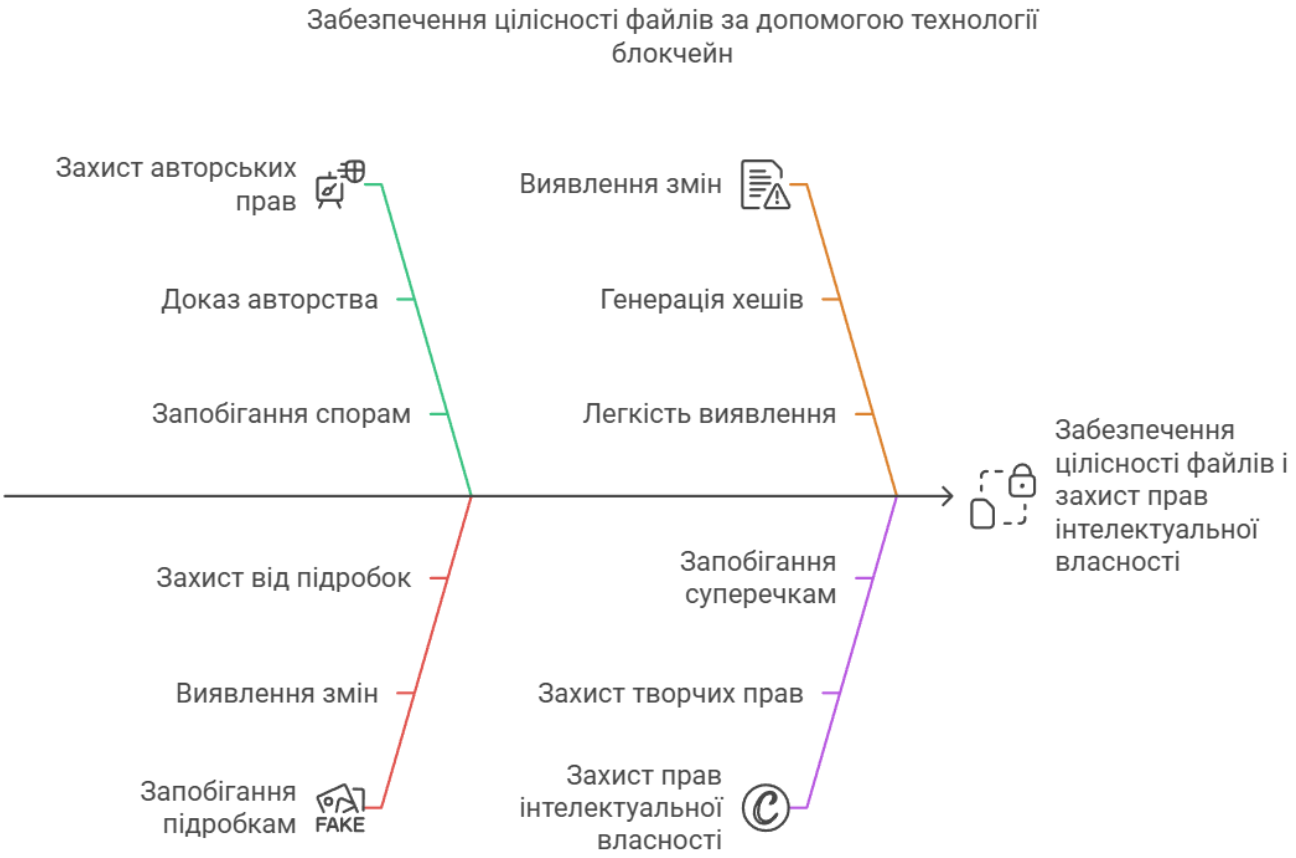

Рисунок 1.2 – Процес забезпечення цілісності файлів

Цей підхід особливо корисний для захисту авторського права. Наприклад, художник може зафіксувати хеш своєї роботи в блокчейні, щоб довести, що він є

автором цієї роботи. Це дозволяє уникнути суперечок щодо авторства та забезпечити захист прав творців .

Переваги блокчейну:

Блокчейн має кілька ключових переваг, які роблять його ефективним інструментом для забезпечення цілісності файлів.

По-перше, незмінність: дані в блокчейні не можуть бути змінені без порушення ланцюжка.

По-друге, децентралізація: відсутність центрального сервера робить систему більш стійкою до атак.

По-третє, прозорість: всі учасники мережі можуть перевіряти цілісність даних, що робить систему більш надійною [12].

Отже відповідно до рисунку 1.3, ми можемо побачити, в чому полягають основні принципи блокчейну.

Рисунок 1.3 – Основні принципи блокчейну

Незмінність може забепечувати легку перевірку змін у даних та контролювати цілісність.

Децентралізація мінімізує ризики втручання в моніторинг цілісності, оскільки ця технологія не є підконтрольна комусь.

При порушенні прав доступу до інформації, блокчейн це впише в хеш який дасть нам зрозуміти, що дані було змінено.

Блокчейн це, по суті, ланцюжок блоків, де кожен блок містить певну інформацію. Уявімо, що кожен блок це як сторінка в великій книзі, де записані всі транзакції або дані. Але ця книга не звичайна, вона має свої особливості, які роблять її надійною та захищеною.

Блоки та їх вміст:

Кожен блок містить три основні речі:

- Це може бути інформація про транзакції (наприклад, хто кому і скільки переказав) або будь-які інші дані, які потрібно зберегти.
- Хеш попереднього блоку: Це унікальний ідентифікатор, який зв’язує поточний блок з попереднім. Якщо хтось спробує змінити дані в попередньому блоці, його хеш зміниться, і це порушить ланцюжок.
- Кожен блок має свій унікальний ідентифікатор, який генерується на основі його вмісту. Якщо дані в блоці змінюються, хеш теж змінюється.

Ланцюжок блоків: Блоки з’єднані між собою через хеші. Наприклад, блок №2 містить хеш блоку №1, блок №3 містить хеш блоку №2 і так далі. Це створює ланцюжок, де кожен блок залежить від попереднього. Якщо хтось спробує змінити дані в одному блоці, це порушить всю структуру, і зміни будуть легко виявлені.

На відміну від звичайних баз даних, які зберігаються на одному сервері, блокчейн розподілений між тисячами комп’ютерів по всьому світу. Це означає, що немає центрального пункту, який можна зламати. Щоб змінити дані, треба змінити їх на всіх комп’ютерах одночасно, що практично неможливо.

Одна з головних переваг блокчейну це його незмінність. Якщо дані записані в блокчейн, їх неможливо змінити без порушення ланцюжка. Це робить блокчейн

ідеальним інструментом для захисту цілісності файлів, особливо в контексті авторського права або фінансових транзакцій.

Уявімо, що ви розробник програмного забезпечення, і ви хочете захистити свою програму від підробки. Ви можете згенерувати хеш файлу з вашою програмою та зберегти його в блокчейні. Якщо хтось спробує змінити файл, хеш зміниться, і це буде легко виявити. Таким чином, ви можете довести, що саме ви є автором цієї програми, і захистити її від несанкціонованих змін.

Блокчейн це не просто технологія для криптовалют. Він може бути використаний для захисту будь-яких даних, які потребують високого рівня цілісності та безпеки. Наприклад, у фінансовій сфері блокчейн дозволяє забезпечити прозорість транзакцій, а в галузі авторського права, тобто захистити права творців.

Отже, блокчейн це потужний інструмент, який дозволяє забезпечити цілісність даних, зробити їх незмінними та захищеними від підробки. І це лише початок його можливостей!

Структура блокчейну зображена на рисунку 1.4.

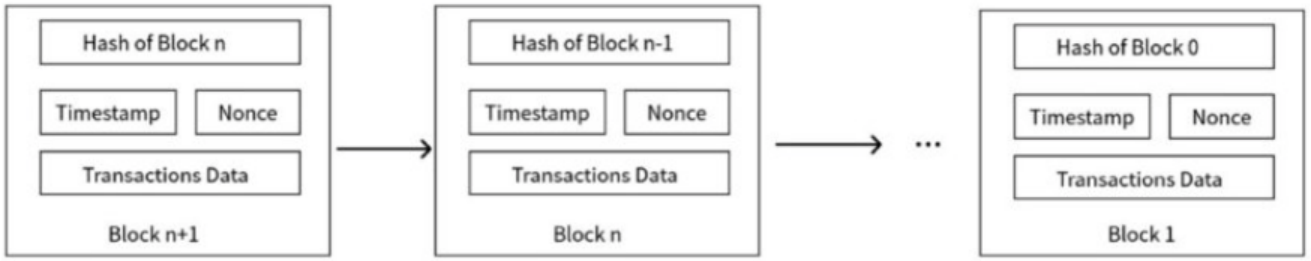

Рисунок 1.4 – Структура блокчейну.

На рисунку 5 показані основні компоненти технології блокчейн, описано, як вона працює та її ключові характеристики.

На зображенні показано, що блокчейн складається з інформації про транзакції, яка може містити дані про перекази, зберігання чи інші операції:

- Кожен блок містить хеш попереднього блоку, який є унікальним ідентифікатором, який зв’язує блоки разом і забезпечує цілісність ланцюжка.

- Кожен блок також має власний унікальний ідентифікатор, який генерується на основі вмісту блоку, що робить його незмінним.

Крім того, блокчейн має розподілену структуру, тобто дані зберігаються на багатьох комп’ютерах у мережі, і немає єдиної точки вразливості, яку можна було б зламати. Це робить систему більш стійкою до атак.

Ще однією важливою властивістю є незмінність даних. Після запису даних у блокчейні їх неможливо змінити без виявлення, що забезпечує високий рівень безпеки та довіри до системи.

Ці компоненти разом роблять блокчейн потужним інструментом для забезпечення цілісності даних, прозорості та безпеки в різних сферах, таких як фінанси, авторське право та багато інших. Компоненти блокчейну зображені на рисунку 1.5.

Рисунок 1.5 – Компоненти технології блокчейн

1.5 Правовий захист програмного забезпечення

У статті Bently L., Johnson P. "The legal protection of software: the case for a specific protection of software by patent law" розглядаються питання правового захисту програмного забезпечення (ПЗ) через патенти. [13]

Бентлі та Джонсон аргументують тим, що традиційні механізми захисту, такі як авторське право, не завжди є достатньо ефективними для захисту програмного

забезпечення, бо вони охоплюють лише вираз від частинки ідеї, а не саму ідею чи її функціональність.

Авторське право захищає вихідний код програми, але, на жаль, не захищає алгоритми або функції, які лежать в основі ПЗ. З цього випливає що є значний ризик для розробників, адже конкуренти можуть легко скопіювати функціональність програмного забезпечення, змінивши лише вихідний код.

Якщо розробник створює новий алгоритм для швидкого сортування даних, патент дозволяє йому захистити цей алгоритм від копіювання конкурентами. Це стимулює інновації, оскільки розробники можуть бути впевнені, що їхні технологічні рішення будуть захищені.

А це означає, що це вплине не тільки на захист ПЗ, а й на мотивацію фахівців та спеціалістів. Нові технології додають інтересу до роботи та чогось нового. Вивчення нових технологій дозволяє створювати власні нові технолгії та покращувати інші, вже винайдені технології. Виходячи з даних, я також звернув увагу на проблеми, пов’язані з патентуванням ПЗ.

По-перше, процес отримання патента може бути довгим і дорогим, що робить його недоступним для малих компаній або стартапів.

По-друге, існує ризик надмірного патентування, коли компанії отримують патенти на очевидні або тривіальні рішення, що може загальмувати інновації.

На рисунку 1.6 зображені основні підходи щодо захисту інтелектуальної власності.

Рисунок 1.6 – Підходи щодо захисту інтелектуальної власності

У висновку статті автори підкреслюють, що спеціальний захист програмного забезпечення через патенти може бути ефективним інструментом для захисту інновацій, але вимагає ретельного балансу між захистом прав розробників та стимулюванням конкуренції та інновацій.

Вони пропонують впровадити чіткі критерії для патентування програмного забезпечення, щоб уникнути зловживань та забезпечити справедливий захист інновацій.

1.6 Криптографія з відкритим ключем

Робота Pradosh Kumar Mohapatra "Public Key Cryptography", представляє собою глибокий аналіз сучасних методів криптографії з відкритим ключем, які є основою для забезпечення безпеки в цифровому світі. Автор розглядає цю технологію не лише як інструмент для шифрування, але й як фундаментальний

механізм, що забезпечує автентифікацію, цілісність даних та конфіденційність у мережевих комунікаціях. Криптографія з відкритим ключем, також відома як асиметрична криптографія, базується на використанні двох ключів: відкритого (публічного) та закритого (приватного). Відкритий ключ доступний для всіх і використовується для шифрування даних, тоді як закритий ключ знаходиться в секреті та використовується для дешифрування. Цей підхід дозволяє вирішити одну з головних проблем симетричної криптографії це безпечний обмін ключами між сторонами [14].

Автор підкреслює, що криптографія з відкритим ключем стала основою для багатьох сучасних технологій, таких як SSL/TLS, які забезпечують безпечний зв’язок в Інтернеті, а також для цифрових підписів, які використовуються для підтвердження автентичності та цілісності документів.

Наприклад, коли ви відвідуєте веб-сайт з HTTPS, ваш браузер використовує відкритий ключ сервера для шифрування даних, які передаються, і лише сервер має закритий ключ для їх розшифрування. Це забезпечує захист від перехоплення даних зловмисниками.

Однією з ключових переваг криптографії з відкритим ключем є її здатність забезпечувати автентифікацію.

Цифрові підписи, які базуються на цій технології, дозволяють перевірити, що повідомлення або документ дійсно надійшли від заявленого відправника і не були змінені під час передачі.

Це особливо важливо в таких сферах, як електронна комерція, де необхідно забезпечити довіру між сторонами.

Також звертається увага на виклики, пов’язані з використанням криптографії з відкритим ключем. Одним із головних є проблема управління ключами.

Оскільки відкриті ключі повинні бути доступні для всіх, важливо забезпечити їхню автентичність.

Для цього використовуються сертифікати, які видаються довіреними центрами сертифікації (CA). Однак система CA не є ідеальною, оскільки вона залежить від

довіри до цих центрів, і будь-який компроміс CA може призвести до серйозних наслідків для безпеки.

Ще одним викликом є обчислювальна складність. Алгоритми асиметричної криптографії, такі як RSA або ECC (Elliptic Curve Cryptography), вимагають значно більше обчислювальних ресурсів порівняно з симетричними алгоритмами.

Це може бути проблемою для пристроїв з обмеженими ресурсами, таких як IoT-пристрої або мобільні телефони.

Автор також обговорює перспективи розвитку криптографії з відкритим ключем. Одним із найперспективніших напрямків є квантова криптографія, яка може забезпечити новий рівень безпеки, зокрема в умовах розвитку квантових комп'ютерів, які можуть зламати традиційні криптографічні алгоритми.

Крім того, Mohapatra звертає увагу на розвиток постквантової криптографії, яка розробляє алгоритми, стійкі до атак з використанням квантових комп'ютерів.

У висновку статті автор підкреслює, що криптографія з відкритим ключем залишається одним із найважливіших інструментів для забезпечення безпеки в цифровому світі.

Незважаючи на виклики, такі як управління ключами та обчислювальна складність, ця технологія продовжує розвиватися, забезпечуючи нові рівні захисту для мережевих комунікацій, електронної комерції та інших сфер.

Висновки до першого розділу

Отже, забезпечення цілісності файлів є критично важливим аспектом кібербезпеки, особливо в контексті захисту програмного забезпечення та фінансових даних. Використання хеш-функцій, цифрових підписів та блокчейн-технологій дозволяє забезпечити високий рівень захисту від несанкціонованих змін та підробки.

Законодавчі вимоги, такі як DMCA, WIPO, GDPR та ISO 27001, встановлюють стандарти для забезпечення цілісності та конфіденційності даних, що робить їх важливими для організацій, які прагнуть забезпечити кібербезпеку. Блокчейн-технології, з їхньою незмінністю та децентралізацією, є перспективним

інструментом для забезпечення цілісності файлів, особливо в контексті захисту авторського права.

РОЗДІЛ 2
АНАЛІЗ МЕТОДІВ ЗАХИСТУ АВТОРСЬКОГО ПРАВА

2.1. Види та методи захисту цілісності ПЗ

У сучасному цифровому середовищі захист програмного забезпечення (ПЗ) від несанкціонованого копіювання, змін і підробок є критично важливим завданням. З поширенням Інтернету та комунікаційних технологій програмне забезпечення стало одним з найцінніших активів ІТ-індустрії. Легкість копіювання цифрового контенту призвела до необхідності розробки нових методів захисту авторських прав, серед яких ключову роль відіграє цифрове маркування (watermarking).

Методи цифрового маркування забезпечують ідентифікацію власника авторських прав, відстеження несанкціонованого розповсюдження та виявлення підробок. На відміну від традиційних механізмів контролю доступу та ліцензування, watermarking дозволяє вбудовувати приховану інформацію безпосередньо у вихідний код ПЗ або його виконувані файли, що робить цей метод ефективним навіть у разі модифікації програмного забезпечення.

Захист ПЗ базується на двох основних підходах: апаратному та програмному. Апаратні методи, такі як використання спеціалізованих чипів або захищених середовищ виконання, забезпечують високий рівень безпеки, проте їх впровадження є дорогим та складним. Програмні методи, зокрема watermarking, дозволяють інтегрувати механізми захисту без необхідності зміни апаратного забезпечення, що робить їх більш гнучкими та доступними.

2.2. Алгоритми водяних знаків для програмного забезпечення

Розробка та впровадження алгоритмів цифрового маркування ПЗ передбачає використання різних технік, які забезпечують надійний захист від піратства. Однією

з основних вимог до watermarking-методів є їхня стійкість до атак та збереження водяного знака навіть після трансформації коду.

Алгоритм перестановки базових блоків

Метод базується на зміні порядку виконання інструкцій програми без порушення її функціональності. Це досягається шляхом перестановки основних елементів коду (basic blocks), що дозволяє вбудувати унікальну цифрову мітку у структуру виконуваного файлу. [15]

Структура зображена на рисунку 2.1.

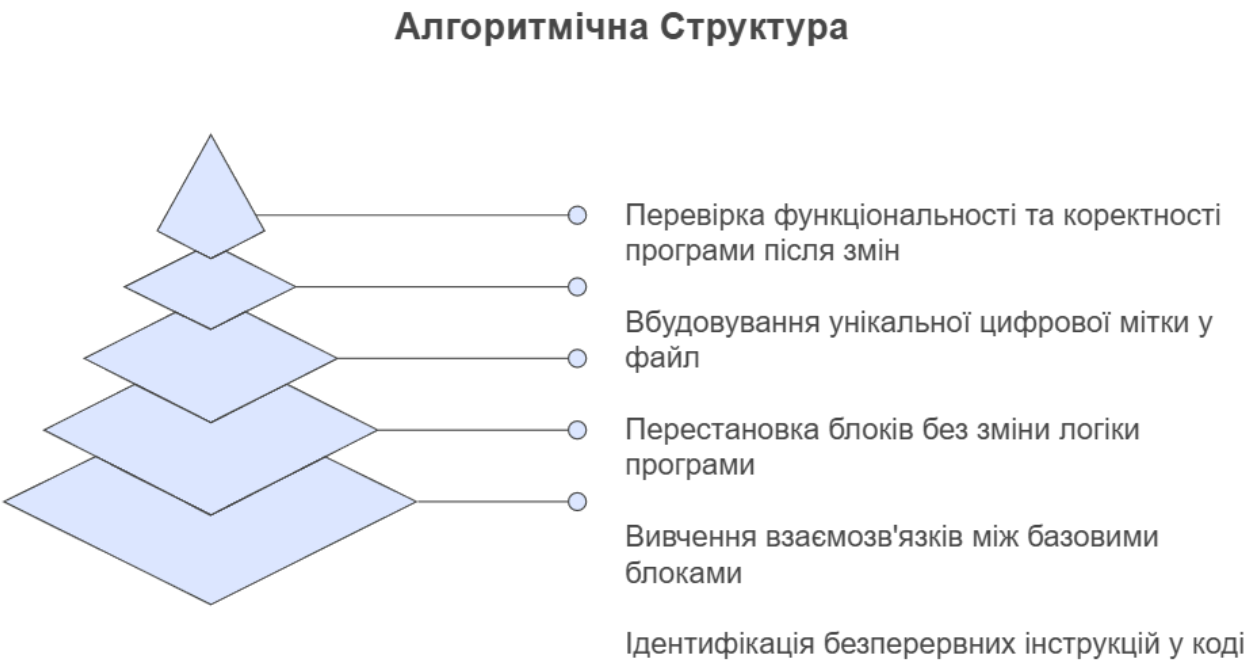

Рисунок 2.1 – Алгоритмічна структура перестановки блоків

Цей підхід використовується у випадках, коли необхідно забезпечити непомітність маркування та його збереження після трансформацій компілятора.

Семантично розподілене маркування (SDSW-алгоритм)

SDSW (Semblance Based Disseminated Software Watermarking) це метод, який використовує модифікацію інструкцій коду шляхом створення спеціальних

шаблонів, що містять приховану інформацію. Алгоритм включає кілька ключових етапів:

- Кодування водяного знака (Watermark Encoding): процес генерації та приховування маркувальних даних у вихідному коді.
- Словникове відображення (Dictionary Mapping): використання спеціальних таблиць відповідності для зберігання інформації про водяний знак.
- Вбудовування інструкцій (Instruction Embedding): інтеграція маркувальних компонентів у виконуваний код програми.
- Розпізнавання водяного знака (Watermark Recognizer): механізм вилучення маркування з виконуваного файлу для перевірки його автентичності. [15]

Рисунок 2.2 – Схема досягнення автентичності ПЗ

Це дозволяє ефективно впроваджувати невидимі водяні знаки, які не впливають на продуктивність програмного забезпечення.

Метод динамічного графічного маркування (DGW)

Один із способів захисту коду від атак полягає у використанні динамічних графічних водяних знаків (Dynamic Graph Watermark, DGW). Метод заснований на модифікації структур даних програми та створенні унікального графа, який містить закодовану інформацію про правовласника. [15]

DGW-метод полягає у тому, що він дозволяє ефективно протидіяти атакам, пов'язаним з аналізом потоку виконання програми. Навіть у разі модифікації коду, його структурна схема залишається незмінною, що дозволяє зберегти автентичність водяного знака.

Захист від несанкціонованих змін.

Головна загроза для watermarking-методів це можливість їх видалення шляхом реверс-інжинірингу. Для запобігання таким атакам застосовуються додаткові механізми тамперостійкості (tamper-proofing), які дозволяють виявляти та блокувати спроби зміни коду.

Ці алгоритми вбудовують у програму спеціальні перевірочні механізми, які контролюють цілісність виконуваного файлу. Якщо програмний код був змінений або модифікований без дозволу власника, система може автоматично блокувати його виконання або видавати попередження.

2.3. Алгоритми цифрового маркування

Цифрове маркування (digital watermarking) є ефективним інструментом захисту програмного забезпечення та цифрових даних. Алгоритми маркування можуть бути реалізовані у різних формах, зокрема через модифікацію вихідного коду, зміни у виконуваному файлі, або вбудовування інформації в структури даних програми.

Метод Least Significant Bit (LSB) Modification використовується для приховування водяного знака шляхом зміни найменш значущих бітів у двійковому представленні даних. Цей метод є одним із найбільш простих і малопомітних способів прихованого кодування інформації. [16]

LSB-модифікація працює за принципом зміни останнього біта числового значення в коді програми, що не впливає на її функціональність. Наприклад, у масивах чисел або рядків символів можна приховуати інформацію про правовласника без видимих змін у роботі ПЗ.

Аналіз даного методу зображений на рисунку 2.3.

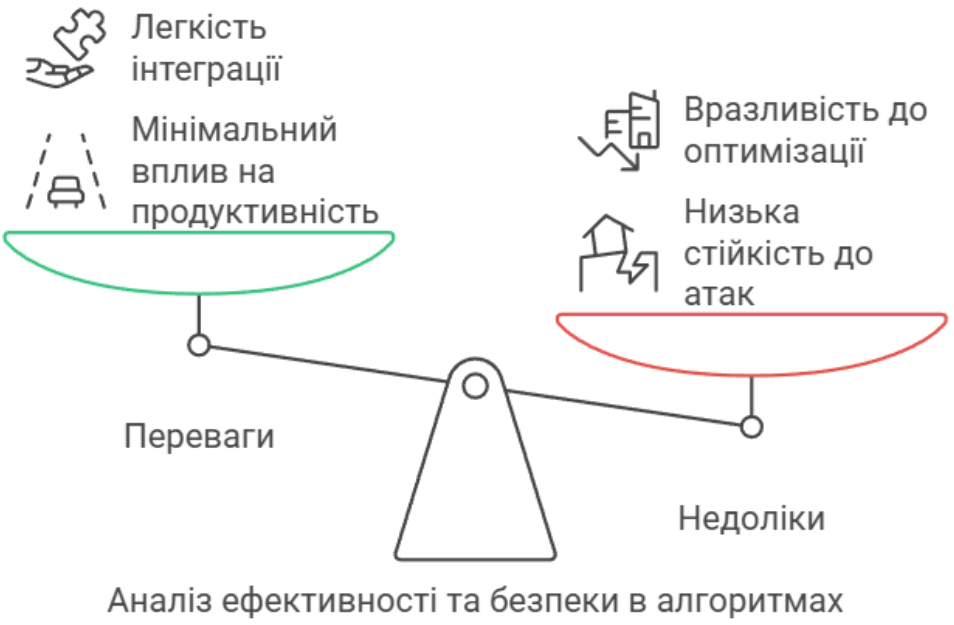

Рисунок 2.3 – Аналіз ефективності та безпеки в алгоритмі

Основні переваги:

- Мінімальний вплив на продуктивність програми.
- Відсутність видимих змін у вихідному коді.
- Висока швидкість виконання алгоритму.

Недоліки:

- Низька стійкість до атак, що змінюють двійковий код.
- Вразливість до методів стиснення та оптимізації коду.

Кореляційний метод маркування (Correlation-Based Techniques). Цей метод базується на додаванні водяного знака у вигляді спеціального коду, який корелює з вихідними даними програми. [16]

Для вилучення маркування використовується кореляційний аналіз, що дозволяє перевірити наявність або відсутність цифрового підпису у виконуваному файлі.

Процес включає:

1. Генерацію унікального водяного знака на основі випадкових значень або криптографічних алгоритмів.

2. Вбудовування коду в програму за допомогою змін у її логічній структурі.

3. Використання спеціального детектора для перевірки автентичності ПЗ.

Процес зображений на рисунку 2.4.

Рисунок 2.4 – Процес захисту ПЗ на основні кореляційного методу маркування

Основні переваги:

- Висока стійкість до випадкових змін коду.

- Можливість перевірки автентичності без повного доступу до вихідного коду.

Недоліки:

- Потребує додаткових ресурсів для перевірки маркування.
- Може впливати на продуктивність програми.

Частотний метод цифрового маркування (Frequency Domain Techniques)

Частотні методи використовують перетворення Фур'є або вейвлет-перетворення для приховування інформації у високочастотних або низькочастотних компонентах програми.

Методи, що використовуються:

- Дискретне косинусне перетворення (DCT) це вбудовування інформації в частотні коефіцієнти.
- Дискретне вейвлет-перетворення (DWT) це модифікація частотних складових коду для приховування водяного знака.

Цей метод широко використовується у цифрових зображеннях, аудіофайлах та відео, але також може бути застосований для захисту ПЗ.

Маркування на основі сингулярного розкладу (SVD-Based Watermarking)

Метод Singular Value Decomposition (SVD) використовує розклад матриць для приховування інформації про власника авторських прав у програмному коді. Основна ідея полягає у розкладі вихідного масиву даних на три матриці, у які додається водяний знак. [16]

Алгоритм реалізації:

1. Вибір частини програмного коду для приховування інформації.
2. Розклад матриці даних на три складові.
3. Вбудовування унікального цифрового підпису у розкладені компоненти.
4. Відновлення програми після змін із можливістю перевірки маркування.

Вейвлет-маркування програмного забезпечення (Wavelet Watermarking Techniques). Вейвлет-методи використовують перетворення з багаторівневою декомпозицією даних. Це дозволяє вставляти водяний знак на кількох рівнях структури програми, що ускладнює його видалення. [16]

Методика включає:

- Декомпозицію даних за допомогою вейвлет-функцій.
- Вбудовування водяного знака у високочастотні компоненти.
- Інверсне перетворення для відновлення вихідного коду із маркуванням.

Таблиця 2.1

Порівняльний аналіз методів цифрового маркування

Метод	**Основні принципи**	**Переваги**	**Недоліки**
Частотний метод цифрового маркування (DCT, DWT)	Використання перетворень Фур'є або вейвлет-перетворень для приховування інформації у частотних складових коду.	Висока стійкість до атак на базі стиснення. Відсутність видимих змін у вихідному коді.	Складність реалізації у програмному забезпеченні. Необхідність складних алгоритмів вилучення водяного знака.
Маркування на основі сингулярного розкладу (SVD)	Використання SVD-розкладу матриць для приховування інформації у програмному коді.	Висока стійкість до модифікації коду. Підходить для складних програмних систем.	Висока обчислювальна складність. Великі вимоги до пам'яті.
Вейвлет-маркування (Wavelet Watermarking)	Використання багаторівневої декомпозиції даних з вбудовуванням знака у	Висока прихованість маркування. Захист від реверс-інжинірингу.	Складність реалізації у виконуваних файлах. Додаткові вимоги до

	високочастотні компоненти		обчислювальних ресурсів.

2.4. Вбудовування водяних знаків у реляційні бази

Реляційні бази даних (RDBMS) часто використовуються у корпоративному секторі для зберігання важливої інформації. Використання watermarking-технік у базах даних дозволяє:

- Захистити дані від модифікації та копіювання.
- Визначити власника інформації у разі витоку.
- Перевіряти автентичність записів. [17]

Один із підходів до маркування баз даних передбачає вбудовування прихованої інформації у зайві поля. Наприклад, можна додати невидимі символи або спеціальні коди у певні атрибути таблиці.

Позапорядкові реляційні дані

Метод передбачає перестановку записів у таблицях без зміни їх значень, що дозволяє додати унікальний порядок записів для кожного клієнта.

Маркування часто оновлюваних записів

Ця методика дозволяє вбудовувати водяні знаки у дані, що регулярно змінюються, що ускладнює їх видалення без руйнування структури бази.

Методи захисту програмного забезпечення постійно еволюціонують у відповідь на зростаючу кількість загроз та спроб несанкціонованого втручання. Одними з ключових механізмів, що застосовуються з метою ускладнення аналізу програмного коду та запобігання його несанкціонованій модифікації, є обфускація (англ. obfuscation) та цифровий підпис коду (code signing). Обидва підходи, хоча й мають різну природу та технічну реалізацію, об’єднані спільною метою забезпечити вищий рівень безпеки, цілісності та довіри до програмного продукту як на рівні розробника, так і з боку кінцевого користувача.

Обфускація це спеціальний процес перетворення програмного коду таким чином, щоб зберегти його функціональність, але значно ускладнити його розуміння для стороннього аналітика чи зловмисника. У практиці програмування це зазвичай включає перейменування змінних і функцій у набори беззмістовних символів, видалення коментарів і налагоджувальної інформації, а також перебудову логіки коду, включаючи вставку зайвих умов, циклів або заплутаних конструкцій. У результаті отримується технічно працездатна програма, яку важко або неможливо проаналізувати традиційними методами, такими як дизасемблювання, декомпіляція або ручне вивчення структури коду. Це суттєво ускладнює реверс-інжиніринг. Це процес, під час якого дослідник намагається відновити логіку роботи програмного забезпечення без доступу до вихідного коду. Таким чином, обфускація виступає як потужний інструмент захисту інтелектуальної власності та конфіденційних алгоритмів, особливо в умовах, коли програма поширюється у відкритому середовищі, наприклад, через інтернет.

Особливо актуальною обфускація є для програм, написаних мовою Java, оскільки програми на цій мові компілюються у байт-код, що порівняно легко піддається декомпіляції. Тому захист Java-додатків зазвичай реалізується як на рівні вихідного коду (до компіляції), так і на рівні байт-коду (після компіляції). Для цього використовуються спеціалізовані інструменти, такі як ProGuard, Allatori, DashO та інші, які дозволяють створити захищену версію програмного забезпечення з мінімальним впливом на його продуктивність.

Іншим важливим елементом у забезпеченні безпеки програмного коду є використання цифрових підписів. Підпис коду це процес, що ґрунтується на криптографії з відкритим ключем, і дозволяє не лише перевірити цілісність програмного забезпечення, а й підтвердити його походження. В основі підпису коду лежить використання хеш-функції, яка генерує унікальний цифровий відбиток (хеш) для конкретної версії програми, та закритого ключа, яким розробник підписує цей хеш. Потім цей цифровий підпис прикріплюється до програмного файлу. Коли користувач завантажує чи запускає програму, система може за допомогою відкритого

ключа розробника перевірити, чи відповідає підписаний хеш фактичному стану файлу. Якщо файл було змінено навіть на один байт, підпис вважається недійсним.

Цифрове підписування не тільки підтверджує, що програмне забезпечення не зазнало змін з моменту його підписання, але й забезпечує перевірку особи автора програми. Це особливо важливо в корпоративному середовищі, де багато компаній не дозволяють встановлювати або запускати непідписані додатки. Таким чином, цифровий підпис стає критично важливим інструментом не лише з точки зору безпеки, а й для побудови довіри до програмного забезпечення, яке поширюється серед широкого кола користувачів. Цей метод активно використовується в екосистемах Windows, macOS, Android та iOS, де підпис є обов'язковим для розгортання програм.

Поєднання обфускації та цифрового підпису дає змогу реалізувати комплексний підхід до захисту програмного забезпечення. З одного боку, обфускація приховує логіку роботи програми, зменшуючи ризик її аналізу або модифікації. З іншого боку, підписування коду гарантує, що навіть у разі спроби зміни програми зловмисником ця модифікація буде одразу виявлена як порушення цілісності. Такий симбіоз методів дозволяє забезпечити більш високий рівень безпеки, особливо в критичних сферах, таких як фінансові додатки, системи електронного документообігу, засоби зв'язку та державні інформаційні ресурси.

Отже, впровадження методів обфускації та цифрового підпису коду є не просто рекомендованим, а необхідним етапом у процесі створення сучасного програмного забезпечення. Вони дозволяють запобігти несанкціонованому доступу до логіки програми, захищають від модифікацій і поширення шкідливих версій, а також формують довіру з боку кінцевих користувачів та партнерів. У контексті кібербезпеки, ці методи слід розглядати як обов'язкову складову стратегії захисту програмного продукту на всіх етапах його життєвого циклу.

Приклади застосування обфускації та підпису коду в сучасній практиці

1. Обфускація як механізм захисту бізнес-логіки

Уявімо, що компанія розробляє програму для автоматичного трейдингу на фондових ринках. Основна цінність такого ПЗ це унікальні алгоритми ухвалення

торговельних рішень, які є результатом років досліджень і тестувань. Якщо вихідний код буде легко зчитуваним після компіляції, зловмисник зможе скопіювати або змінити ці алгоритми для створення конкурентного продукту. Використання обфускації у цьому випадку дозволяє заплутати логіку програми настільки, щоб навіть у разі доступу до коду без прямої авторизації, зловмиснику знадобилися б значні зусилля та ресурси для розуміння його суті. У Java-програмах, які легко декомпілюються до читаємого вигляду, цей захід є критично необхідним.

2. Підпис коду як елемент довіри в цифрових екосистемах

Підпис коду є необхідною умовою для розміщення додатків на більшості сучасних платформ. Наприклад, операційні системи Windows та macOS використовують цифрові підписи для перевірки автентичності програм. Якщо файл не має дійсного підпису, система може заблокувати його запуск або попередити користувача про потенційну небезпеку. Це особливо важливо для антивірусного або банківського програмного забезпечення, де довіра до розробника є критичною. Крім того, на Android та iOS платформах додатки без підпису просто неможливо встановити, оскільки цифровий підпис слугує також ідентифікатором розробника, що забезпечує захист від підміни чи підробки програм.

3. Інтегроване використання в корпоративному середовищі

У великих компаніях, особливо в банківському секторі, застосування обох методів одночасно обфускації та цифрового підпису є стандартом безпеки. Наприклад, якщо розробляється внутрішня система електронного документообігу, програму перед розповсюдженням серед співробітників спочатку обфускують для приховування логіки обробки конфіденційних даних, а потім підписують із використанням сертифіката, виданого внутрішнім центром сертифікації. Це гарантує як конфіденційність, так і верифікацію цілісності коду. Користувачі, своєю чергою, бачать, що програма є автентичною, і можуть довіряти її джерелу.

4. Захист від модифікації та шкідливих втручань

Ще один практичний приклад це захист від втручання у відкриті клієнти ігор. У багатьох онлайн-іграх, де гравці можуть спробувати змінити клієнт для здобуття переваги, розробники використовують обфускацію, щоб ускладнити модифікацію

логіки гри, а також підпис коду, щоб переконатися, що запущений клієнт не був змінений. Сервер перевіряє підпис під час підключення, і у разі невідповідності блокує доступ.

Таким чином, ці приклади наочно демонструють, що як обфускація, так і цифрове підписування не є лише теоретичними концепціями. Вони мають практичне значення та широке застосування в реальному світі від мобільних додатків і фінансових систем до ігрових клієнтів і внутрішнього корпоративного ПЗ. Розробники, які ігнорують ці засоби захисту, наражають свій продукт на серйозні ризики, пов’язані з компрометацією, незаконним використанням або поширенням модифікованих версій. У контексті зростаючого значення кібербезпеки та цифрової довіри, застосування таких методів вже не є опцією це необхідність.

2.5. Використання хеш-функцій та цифрового підпису для захисту ПЗ

Використання хеш-функцій та цифрового підпису є ключовими елементами у забезпеченні захисту програмного забезпечення (ПЗ). Хеш-функції забезпечують контроль цілісності даних, тоді як цифровий підпис гарантує автентичність та неспростовність походження ПЗ.

Хеш-функції в захисті ПЗ

Хеш-функції є математичними алгоритмами, які перетворюють вхідні дані будь-якої довжини у фіксовану довжину хеш-значення. Це значення є унікальним для кожного набору вхідних даних, що дозволяє виявляти будь-які зміни у файлах. У контексті захисту ПЗ, хеш-функції використовуються для:

- Контролю цілісності: Обчислення хеш-значення для кожного файлу ПЗ дозволяє перевіряти його цілісність. Якщо хеш-значення змінюється, це свідчить про модифікацію файлу, що може бути результатом несанкціонованого втручання або пошкодження.
- Захисту паролів: Замість зберігання паролів у відкритому вигляді, системи зберігають їх хеш-значення. Це підвищує безпеку, оскільки навіть у разі компрометації бази даних, відновити оригінальні паролі з хеш-значень є надзвичайно складно. [18]

Цифровий підпис у захисті ПЗ

Цифровий підпис є криптографічним механізмом, який забезпечує автентичність та цілісність даних. Він гарантує, що ПЗ походить від надійного джерела і не було змінено після підписання. Процес створення цифрового підпису включає:

1. Хешування: Обчислення хеш-значення від вихідного повідомлення або файлу.

2. Шифрування хеш-значення: Отримане хеш-значення шифрується за допомогою закритого ключа відправника, створюючи цифровий підпис.

При перевірці підпису отримувач розшифровує його за допомогою відкритого ключа відправника та порівнює отримане хеш-значення з самостійно обчисленим хеш-значенням від отриманого файлу. Якщо вони співпадають, це підтверджує автентичність та цілісність ПЗ. [17]

Переваги використання хеш-функцій та цифрового підпису

- Ефективність: Хеш-функції забезпечують фіксовану довжину вихідного значення незалежно від розміру вхідних даних, що спрощує обробку та зберігання цифрових підписів. [17]
- Безпека: Односторонність хеш-функцій ускладнює відновлення вихідних даних з хеш-значення, що підвищує безпеку зберігання та передачі інформації.
- Неспростовність: Цифровий підпис забезпечує юридичну значущість електронних документів, оскільки підтверджує авторство та цілісність даних.

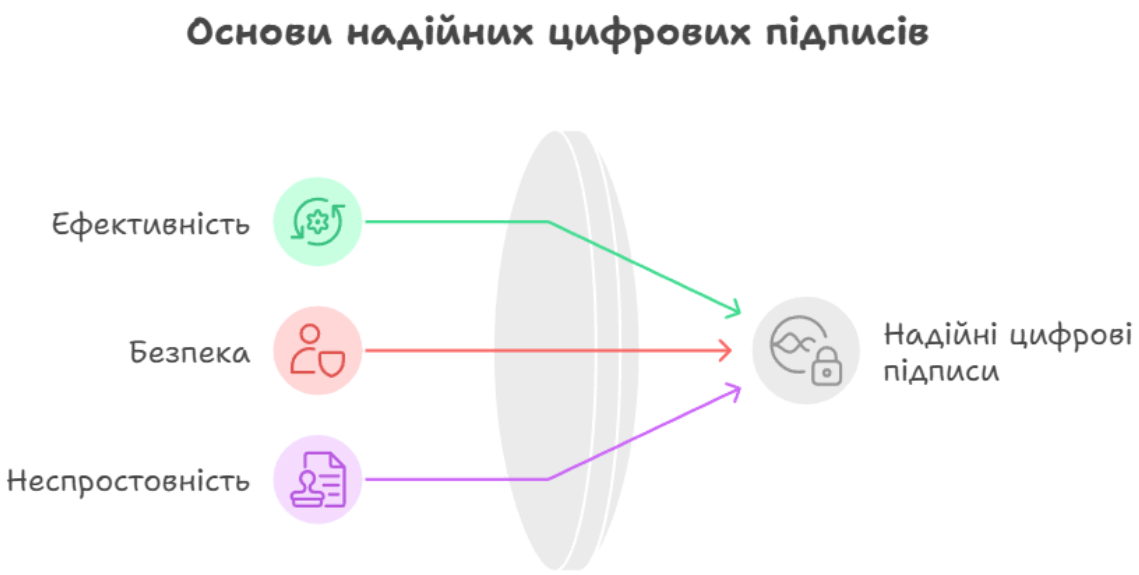

Рисунок 2.5 – Основи надійних цифрових процесів

Таким чином, інтеграція хеш-функцій та цифрового підпису є невід'ємною частиною сучасних систем захисту ПЗ, забезпечуючи його цілісність, автентичність та захист від несанкціонованих змін.

2.6 Методи приховання змін у коді (Obfuscation, Code Signing)

Методи маскування змін у програмному коді, зокрема обфускація та цифровий підпис, є ключовими інструментами для підвищення рівня безпеки програмного забезпечення.

Обфускація передбачає навмисне ускладнення структури коду з метою ускладнення його аналізу та розуміння. Це реалізується шляхом зміни ідентифікаторів (імен змінних, функцій), видалення інформації для налагодження, а також модифікації логіки виконання програми. Основною метою такого підходу є протидія реверс-інжинірингу, що зменшує ризик несанкціонованого доступу до вихідного коду та його подальшого використання чи модифікації. Зокрема, у випадку з Java-додатками обфускація може здійснюватися як на рівні вихідного коду, так і на рівні байт-коду, що забезпечує багаторівневий захист від аналізу.

Цифровий підпис коду є криптографічною технологією, яка гарантує цілісність і достовірність програмного продукту. Суть методу полягає у створенні цифрового підпису, який дозволяє переконатися, що програмний код не зазнав змін після моменту підписання, а також підтвердити авторство розробника. Цей підхід дозволяє кінцевим користувачам упевнитися в автентичності програмного забезпечення й уникнути використання скомпрометованих або шкідливих версій. Основу процесу становить використання хеш-функцій та криптографічних алгоритмів підписування, які дозволяють точно виявляти навіть незначні зміни в коді.

Таким чином, впровадження обфускації та цифрового підпису виступає важливим елементом системного підходу до захисту програмного забезпечення. Вони сприяють збереженню конфіденційності алгоритмів, захисту від несанкціонованого втручання та підтриманню високого рівня довіри до ПЗ з боку користувачів.Методи приховування змін у програмному коді, такі як обфускація та підпис коду, відіграють важливу роль у забезпеченні безпеки програмного забезпечення.

Обфускація полягає у навмисному ускладненні структури програмного коду, щоб зробити його важчим для розуміння та аналізу. Це досягається шляхом зміни імен змінних та функцій, видалення налагоджувальної інформації та спотворення логіки програми. Основна мета обфускації це ускладнити процес реверс-інжинірингу, що знижує ймовірність несанкціонованого доступу до вихідного коду та його модифікації. Зокрема, для Java-програм обфускація може здійснюватися як на рівні вихідного коду, так і на рівні байт-коду, що забезпечує додатковий рівень захисту від несанкціонованого аналізу та модифікації. [19]

Підпис коду є криптографічним методом, який забезпечує цілісність та автентичність програмного забезпечення. Цей процес передбачає використання цифрових підписів для підтвердження того, що код не був змінений після його підписання, а також для верифікації особи розробника. Це дозволяє користувачам бути впевненими в тому, що програмне забезпечення є оригінальним і не містить шкідливих модифікацій. Використання хеш-функцій та цифрових підписів є

ключовими елементами цього процесу, оскільки вони забезпечують перевірку цілісності та автентичності програмного коду. [20]

Застосування обфускації та підпису коду є важливими складовими стратегії захисту програмного забезпечення від несанкціонованого доступу та модифікації. Ці методи допомагають зберегти конфіденційність алгоритмів, забезпечити цілісність коду та підвищити загальний рівень безпеки програмних продуктів.

2.7 Блокчейн у використанні контролю цілісності

Хмарні обчислення (ХО) це середовище, яке забезпечує доступ на вимогу через мережу до обчислювальних ресурсів, таких як сховища, сервери, додатки та інші сервіси, які клієнт може ефективно агрегувати або вивільняти [21].

Клієнти можуть швидко створювати хмарні сервіси в розподілених хмарних дата-центрах, де вони можуть зберігати і обробляти свої дані, а також розгортати і ефективно запускати свої додатки [22].

Послуги ХЦ працюють за бізнес-моделлю, яка передбачає стягнення з клієнта плати за використання обчислювальних ресурсів, які клієнт швидко контрактує і управляє ними за допомогою стандартизованих веб-сервісів без бюрократичних перешкод.

Серед переваг можна виділити масштабованість, доступність і практично необмежену ємність сховища. З огляду на популяризацію послуг, головним чином через їхню нижчу вартість порівняно з традиційними технологіями та постійну появу нових провайдерів послуг зберігання даних у хмарі, багато компаній обирають ці сервіси для зберігання своєї інформації [23].

Звільняє замовника від необхідності турбуватися про складність управління інфраструктурою зберігання даних. Однак конфіденційність, доступність і цілісність даних, що зберігаються, залежать від якості наданих послуг. Через структурні або людські недоліки дані можуть бути пошкоджені, витоку або навмисно видалені.

Оскільки включає в себе комплексну технологію, яка постійно розвивається протягом багатьох років, вона все ще пропонує багато можливостей для вивчення у

відкритих дослідницьких сферах. Багато дослідницьких робіт присвячено інтеграції різних методів і технологій як для вирішення проблем, так і для покращення результатів рішень, зокрема такі приклади:

Yuan [24] пропонують алгоритм, заснований на методах машинного навчання, для покращення графіку розподілу завдань з послуг CC між географічно розподіленими центрами обробки даних, що працюють на зеленій енергії, з метою оптимізації прибутку та зменшення втрат завдань;

Ві пропонують алгоритм, заснований на методах машинного навчання, для покращення графіку розподілу завдань з послуг CC між географічно розподіленими центрами обробки даних, що працюють на зеленій енергії;

Ві та інші пропонують нову структуру, засновану на методах математичного моделювання, для забезпечення динамічного надання ресурсів у віртуалізованих хмарних дата-центрах, мінімізуючи витрати на електроенергію за рахунок вимкнення непотрібних віртуальних машин при дотриманні рівня обслуговування, узгодженого між клієнтом і постачальником хмарних послуг. Wilczyński та Kołodziej пропонують нову загальну модель безпечного хмарного планувальника, засновану на технології Blockchain.

Тобто кожен сусідній вузол у хмарному кластері затверджує запропоновані розклади завдань за допомогою нового алгоритму консенсусу «доказ розкладу», отримуючи внутрішню згоду перед тим, як запропонувати розклад кінцевим користувачам.

Ще одним прикладом цієї технології є блокчейн, який дозволяє створити децентралізовану базу даних, в якій агенти та установи можуть здійснювати транзакції, що піддаються перевірці, при цьому жодна зі сторін не може контролювати або нав'язувати ринкову владу.

Однією з особливостей блокчейну є те, що він дозволяє ефективно підтримувати консенсус щодо хронологічного порядку подій та стану справ [25].

Реалізація блокчейну дозволяє створювати мережі, що складаються із зацікавлених учасників, серед яких немає вимоги взаємної довіри. Транзакції, здійснені цими учасниками, записуються невеликими блоками, утворюючи

ланцюжок послідовних блоків (Blockchain), який реплікується всім учасникам відповідної мережі.

Транзакції здійснюються безпосередньо між сторонами без необхідності залучення довіреної третьої сторони. Після того, як новий блок вставляється в блокчейн, зареєстровані транзакції не можуть бути змінені або скасовані [26].

Смарт-контракт (СК) це цифровий контракт, захищений від несанкціонованого втручання, який часто забезпечується за допомогою автоматизованого виконання [26]. Кожен СК це невелика комп'ютерна програма, що зберігається і виконується в блокчейні для виконання умов угоди між сторонами, які не зобов'язані довіряти одна одній

Висновки до другого розділу

Отже, було виконано глибокий аналіз сучасних методів і систем контролю цілісності програмного забезпечення, який засвідчив широкий спектр підходів із власними сильними та слабкими сторонами. По-перше, методи водяних знаків (наприклад, SDSW та DGW) демонструють високу ефективність у визначенні походження коду та виявленні несанкціонованих модифікацій: унікальні патерни, вбудовані в бінарний або вихідний код, дозволяють відстежувати навіть незначні зміни, проте такі рішення залишаються вразливими до цілеспрямованого реверс-інжинірингу та видалення маркерів досвідченим атакуючим. Хеш-функції, зокрема SHA-256, забезпечують миттєве порівняння контрольних сум і спрощують автоматизацію перевірок, але не запобігають цілком заміні оригінального файлу на новий зі зміненими даними. Для цього потрібні додаткові механізми автентифікації джерела. Цифрові підписи додають рівень криптографічного захисту, підтверджуючи автентичність і немодифікованість об’єкта, проте вимагають розгортання й підтримки складної інфраструктури управління ключами (PKI), що збільшує витрати на впровадження й обслуговування. Інтеграція з блокчейном відкриває можливість незмінного і прозорого зберігання записів про стан контрольних сум і підписів,

проте обмежена низькою пропускною здатністю та високими транзакційними витратами, особливо в публічних мережах з підтвердженням PoW.

У ході порівняння практичних інструментів виявилося, що такі рішення, як VirusTotal, Git та подібні платформи, орієнтовані передусім на антивірусний аналіз та контроль версій, але не пропонують цілісного поєднання всіх вищезгаданих методів. VirusTotal швидко ідентифікує відомі сигнатури шкідливого ПЗ, але не надає можливостей для вбудованого маркування або зберігання даних у блокчейні; Git зручний для відстеження змін у вихідному коді, проте не захищає компільовані артефакти і не інтегрується зі складними криптографічними схемами.

Оцінка недоліків існуючих підходів показала три ключові проблеми: по-перше, відсутність повноцінної автоматизації перевірок у режимі реального часу, що знижує оперативність реагування на спроби втручання; по-друге, обмежена сумісність і взаємодія між різними технологіями (наприклад, комбінування водяних знаків із блокчейном часто реалізується фрагментарно або взагалі вимагає ручного втручання); по-третє, високі обчислювальні й інфраструктурні вимоги для впровадження складніших методів, зокрема SVD-маркування та схем динамічної обфускації.

Цей всебічний аналіз став теоретичною основою для розробки в третьому розділі нового гібридного методу, який поєднує швидкість перевірки хеш-функцій із невразливістю блокчейну та додатковою валідацією через API VirusTotal, що дозволяє усунути виявлені недоліки та забезпечити оперативний, масштабований і стійкий до маніпуляцій контроль цілісності програмного забезпечення.

РОЗДІЛ 3
ПРОГРАМНИЙ МЕТОД МОНІТОРИНГУ ЦІЛІСНОСТІ ФАЙЛІВ

3.1 Інтерфейс GUI (Tkinter)

Інтерфейс графічного користувача Tkinter, який входить до базового розподілу Python та ґрунтується на Tcl/Tk, забезпечує розробнику готову платформу для створення віконних застосунків без необхідності інсталювати додаткові бібліотеки[27]

Така «вбудованість» гарантує, що середовище для прототипування та розробки графічних інтерфейсів доступне відразу після інсталяції Python, що суттєво спрощує налаштування робочого місця та знижує часові витрати на підготовчий етап.

API Tkinter вирізняється чіткою структурою та помірною кількістю базових віджетів.

Розробник працює з текстовими мітками, кнопками, полями введення, фреймами, меню та іншими елементами, що дозволяє сконцентруватися на логіці застосунку, а не на освоєнні надлишкових компонентів[29].

Для розташування елементів на екрані пропонуються три вбудовані менеджери геометрії pack, grid і place які охоплюють найпоширеніші сценарії: від простого шарування до точної прив’язки координат[29].

Такий підхід дозволяє розробникам із різним рівнем підготовки швидко досягати функціональних результатів, уникаючи «перевантаження» інтерфейсу зайвими опціями.

Низький поріг входження до Tkinter відзначають у порівняльних оглядах як одну з його ключових переваг.

У порівнянні з більш потужними фреймворками (наприклад, PyQt чи Kivy), де необхідно опановувати специфічні дизайнерські інструменти та патерни проєктування, Tkinter дозволяє сфокусуватися на основах роботи з подіями та

віджетами, що робить його оптимальним для освітніх курсів і внутрішніх прототипів[30].

Кількість навчальних ресурсів, офіційна документація та численні спільноти (форумі, блоги, відеоуроки) полегшують перехід від теорії до практики навіть початківцям.

Кросплатформеність Tkinter дозволяє створеним у ньому застосункам без змін запускатися на Windows, Linux і macOS, що істотно розширює їхню переносимість та знижує витрати на підтримку окремих версій під різні ОС[31].

При цьому для упаковки у самостійні виконувані файли часто використовують утиліту PyInstaller, яка створює архівні контейнери, сумісні з цільовими платформами, що спрощує доставку кінцевого продукту користувачу.

Втім, стандартний вигляд віджетів у Tcl/Tk іноді сприймають як застарілий у порівнянні з нативним оформленням сучасних операційних систем.

Роль «рецептора» таких зауважень відіграють як обмежений набір тем оформлення за замовчуванням, так і специфічна стилістика «Motif», що не завжди відповідає очікуванням користувача[32].

До того ж, у стандартній поставці немає готових компонентів для роботи з мультимедіа, вкладками чи побудови наукових діаграм, що потребує або створення власних віджетів, або підключення сторонніх розширень[33].

Натомість модуль ttk, інтегрований починаючи з Python 3.1, відкриває можливості тематизації та покращеного оформлення. Застосування стилів (Style) та готових тем (themes), а також сторонніх наборів, таких як ttkthemes, дозволяє привести інтерфейс до єдиного дизайну сучасних ОС, що підвищує привабливість застосунку для кінцевого користувача[34].

Щодо продуктивності, Tkinter демонструє високу швидкість ініціалізації інтерфейсу та обробки базових подій, проте в сценаріях із сотнями і тисячами одночасно активних віджетів можуть виникати затримки в рендерингу та оновленні екрану[35].

Водночас у більшості прикладних завдань це про створення форм вводів, кнопкових панелей чи простих інформаційних панелей, ці затримки малопомітні, а

оптимізовані алгоритми оновлення дозволяють підтримувати потрібну швидкодію[36].

Легкість розгортання без необхідності інсталювати додаткові пакети знижує ризики конфліктів версій та полегшує налаштування середовища на різних робочих станціях.

Універсальність і простота API сприяють тому, що студенти швидко пишуть перші програми та одразу відчувають результат своєї праці.

Разом із тим, для комерційних десктопних застосунків із високими вимогами до сучасного дизайну, мультимедійного супроводу чи складних наукових візуалізацій варто розглядати PyQt, Kivy або Electron, які надають ширший набір готових компонентів та інструментів дизайнерів[37].

Однак у тих випадках, коли пріоритетом є швидкість розробки, легкість підтримки та гарантована кросплатформеність із мінімумом налаштувань, Tkinter залишається незамінним засобом у арсеналі розробника.

3.2 Використання блокчейну в інформаційній безпеці

Блокчейн реалізує децентралізовану модель зберігання даних, де кожен блок містить криптографічно захищений хеш попереднього елемента, що унеможливлює заднім числом змінювати історію транзакцій без консенсусу всіх учасників мережі, суттєво підвищуючи цілісність й достовірність інформації в IT-системах організацій [38].

Консенсусні алгоритми блокчейну, зокрема Proof of Work та Proof of Stake, забезпечують захист від атак типу «подвійна витрата» і розподілене формування єдиного джерела істини, що знижує залежність від централізованих довірчих вузлів та посилює стійкість інформаційних систем до зовнішніх загроз [39].

Застосування блокчейну в контролі доступу дозволяє реалізувати прозорий аудит змін прав користувачів через незмінний журнал подій, у якому кожна операція запису чи змінення прав записується у новий блок із часовою міткою, що мінімізує ризики внутрішніх зловживань доступом [40].

Імплементація смарт-контрактів на основі блокчейну автоматизує виконання безпекових політик, оскільки умови доступу й валідації операцій кодуються безпосередньо в контракті, що виконується одночасно всіма вузлами, забезпечуючи непрозорий та незмінний алгоритм авторизації [41].

Дослідження інтеграції блокчейну в архітектуру Інтернету речей демонструють покращення виявлення аномалій й захисту від несанкціонованого моніторингу, адже кожен пристрій у мережі стає учасником розподіленого реєстру з фіксацією кожної взаємодії [42].

Використання блокчейну для захисту цілісності програмного забезпечення дозволяє контролювати походження та версії коду: кожне оновлення підписується цифровим ключем розробника й фіксується в мережі, що унеможливлює впровадження неперевірених змін [43].

Блокчейн-підхід до керування ідентичностями (“self-sovereign identity”) надає користувачам контроль над власними атрибутами, зберігаючи їх у зашифрованому вигляді в розподіленому реєстрі й зменшуючи ризик централізованих витоків персональних даних [44].

Застосування гібридних блокчейн-мереж це поєднання публічних і приватних блокчейнів, що дозволяє організаціям забезпечувати публічний аудит найважливіших записів при водночас високому рівні конфіденційності комерційних операцій у закритих сегментах мережі [45].

Надійність блокчейну як платформи для проведення транзакцій підтверджується використанням криптографічних цифрових підписів та механізмів узгодження, що забезпечують незаперечність походження даних і відсутність можливості відмови від виконаних операцій [46].

Забезпечення конфіденційності в блокчейн-системах реалізується через технології нульових доказів (Zero-Knowledge Proofs), які дозволяють верифікувати факт володіння інформацією без її розкриття, що особливо актуально для фінансових транзакцій та медичних записів [47].

Гетерогенність блокчейн-рішень для різних сценаріїв безпеки вимагає адаптивних протоколів консенсусу, що підлаштовуються під рівень довіри між

учасниками та обсяги оброблюваних транзакцій, забезпечуючи оптимальний баланс між продуктивністю та безпекою [48].

Інтеграція смарт-контрактів із традиційними системами безпеки передбачає створення гібридних API-шарів, які контролюють виклики до блокчейн-мережі й забезпечують взаємодію з існуючими SIEM та DLP-системами підприємств [49].

Використання блокчейну у ланцюгах постачання створює прозорі та стійкі до фальсифікації логи подій, що дозволяє оперативно виявляти втручання у виробничі процеси та гарантувати достовірність походження продукції [50].

Моделі приватних блокчейн-мереж із керованим переліком учасників дозволяють впроваджувати регуляторні вимоги й політики конфіденційності галузей (наприклад, GDPR), контролюючи, хто саме може бачити або змінювати записи в реєстрі [51].

Застосування блокчейну в охороні здоров’я демонструє ефективність у забезпеченні цілісності та конфіденційності медичних даних пацієнтів через розподілене шифрування й прозору систему доступу, що дозволяє лікарям підтверджувати автентичність записів, а пацієнтам керувати правами доступу [52].

3.3 Моніторинг подій через модуль Watchdog

Моніторинг подій (Watchdog)

Менеджер використовує бібліотеку Watchdog, яка надає кросплатформний API для відстеження змін у директоріях та файлах без необхідності ручного опитування диску

Ключовим компонентом є клас Observer, який запускає цикл спостереження за вказаними шляхами і направляє події до спеціального обробника

Обробник FileSystemEventHandler перехоплює такі події, як on_created, on_modified, on_deleted та on_moved, передаючи інформацію далі для запису в базу даних.

Сховище подій (SQLite).

У якості легковагової реляційної СУБД обрано SQLite, яка вбудована в стандартну бібліотеку Python і не потребує окремого серверного процесу.

Модуль sqlite3 реалізує інтерфейс DB-API 2.0, що дозволяє виконувати транзакції, складати запити SQL та гарантувати атомарність записів.

Основні компоненти та їх принципи зображені на Рисунку 3.1.

Observer (спостерігач): запускає нескінченний цикл, моніторить директорії та передає події до обробника.

EventHandler (обробник): інтерпретує події, формує структуру даних (наприклад, кортеж) і викликає методи запису в базу.

DatabaseManager: встановлює з’єднання з SQLite, готує схему таблиць і керує транзакціями для вставки, оновлення та вибірок.

Configuration: дозволяє налаштувати шляхи обробки, формат журналів, періодичність перевірки та інші параметри.

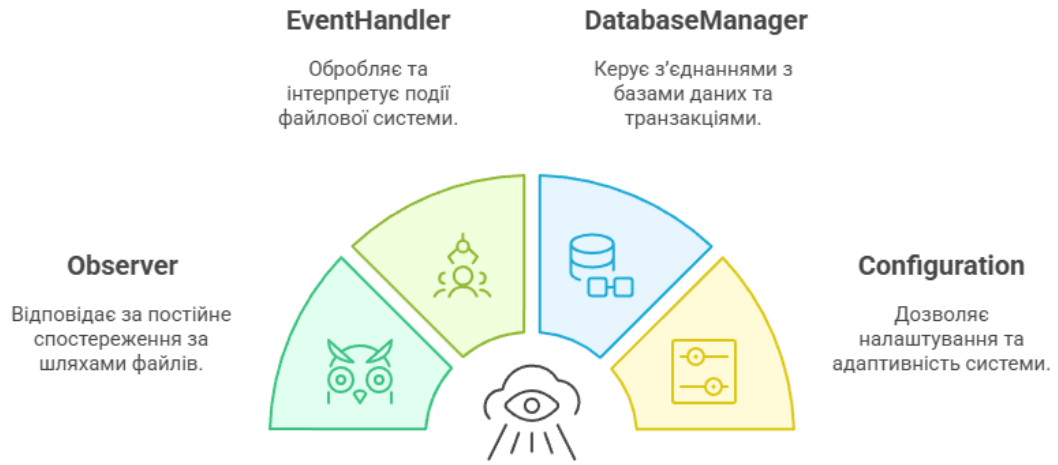

Рисунок 3.1 – Структура системи моніторингу файлів

Менеджер шляху файлів на основі Watchdog та SQLite це просте, надійне та ефективне рішення для відстеження та збереження подій у файловій системі. Воно ідеально підходить як для невеликих внутрішніх утиліт та прототипів, так і для

освітніх проєктів, де важливі швидкість розробки та мінімум зовнішніх залежностей. За умов помірного навантаження архітектура демонструє високу стабільність і забезпечує прозорий аудит змін у режимі реального часу.

Таблиця 3.1

Аналіз недоліків та переваг використання Watchdog та SQLite

Переваги	**Недоліки**
Мінімальні залежності: Watchdog легко встановити, а SQLite вже є у Python.	Обмеження одночасних записів: SQLite підтримує лише одну записувальну транзакцію одночасно.
Кросплатформеність: працює на Windows, Linux, macOS без зміни коду.	Можливі втрати подій: при великій частоті змін Watchdog може не встигати обробити всі події.
Низькі ресурси: немає потреби в окремому сервері баз даних.	Відсутність вбудованої аналітики: потрібне підключення сторонніх бібліотек.
Структурованість: усі події фіксуються у вигляді записів з часом і типом.	Погане масштабування: при дуже великій кількості файлів зростає навантаження.
Швидке прототипування: ідеально підходить для освітніх проєктів і внутрішніх інструментів.	Мінімалістичність інтерфейсу: для складних функцій потрібна додаткова розробка.

3.4 Архітектура програми контролю цілісності файлів

При першому знайомстві з архітектурою розробленої системи відкривається уявний простір делікатно налагодженої взаємодії, немов складний механізм годинника, у якому жодна шестерня не може обійтися без іншої. У цьому тонкому балансі кожен компонент: від спостережувача за змінами до модуля верифікації, далі отримує власну роль, проте тільки в єдності вони творять справжню симфонію

безпеки. Ініціалізація розгортається плавно, крок за кроком: спочатку «прокидається» сховище даних, що покликане берегти історію файлів, далі по черзі вмикаються адаптери зовнішніх сервісів, несучи світлі промені довіри від відомих аналітичних платформ, а завершальним акордом стають механізми спостереження, готові миттєво зафіксувати будь-яку зміну. Такий поетапний сценарій дозволяє уникнути раптових «стрибків» навантаження та дарує кожній складовій можливість увійти в роботу зосереджено й упевнено.

У вирішенні питання, яким інструментом користуватися для моніторингу, ми схилилися до Watchdog і ця прихильність аж ніяк не є випадковою. Поєднання миттєвості сповіщень про створення чи модифікацію файлів і тонкого механізму дебаунсингу, що фільтрує «шум» дрібних та одночасних подій, створює відчуття, ніби система «чує» кожен шорох у файловій системі, але не піддається паніці при справжньому штурмі змін. Це рішення гармонійно поєднує свободу реакції з обережністю, не допускаючи розгубленості навіть у надзвичайних ситуаціях.

Не менш важливою є концептуальна двоступенева верифікація: спочатку локальне сховище фіксує «моментальний знімок» файлу, збережений у вигляді хешу, а вже потім цей унікальний відбиток нісенітно передається до зовнішніх джерел довіри VirusTotal і блокчейну. Сам факт наявності такого поділу на локальне та віддалене дає змогу досягти одночасно високої оперативності й гарантії незмінності у майбутньому. Локальна база виступає швидким кешем, а зовнішні сервіси виступають вищою інстанцією, котра засвідчує справжність і чистоту файлу, наче незалежні експерти, що переглядають докази.

У центрі цієї взаємодії знаходиться IntegrityEngine, подібно до складального конвеєра в сучасному цеху: він приймає «сировину» подій, переробляє її в структуровані запити, дбайливо використовує кеш минулих результатів і скеровує запити одразу до кількох напрямків перевірки. Коли система опиняється під високим навантаженням, цей модуль виявляє гнучкість, розділяючи роботу на паралельні потоки та зберігаючи гроючу легкість інтерфейсу. Саме такий підхід іменують справжньою архітектурною вишуканістю: розмежування обов’язків не лише прискорює виконання, а й додає стійкості до несподіваних навантажень.

Поступовий рух даних завершується у графічному модулі на базі Tkinter, який стає для користувача інтерфейсом тонкої точки дотику з програми. Лаконічні вікна та інтуїтивні елементи дозволяють заглибитись у подробиці перевірок, переглядати історію або вносити налаштування, не відволікаючись на технічні дрібниці. Відокремлення UI від бізнес-логіки дозволяє дизайнерам поступово вдосконалювати вигляд і відчуття програми, залишаючи основну пісню контролю цілісності недоторканою.

На рисунку 3.2 зображено процес двоступеневої верифікації з використанням функції IntegrityEngine.

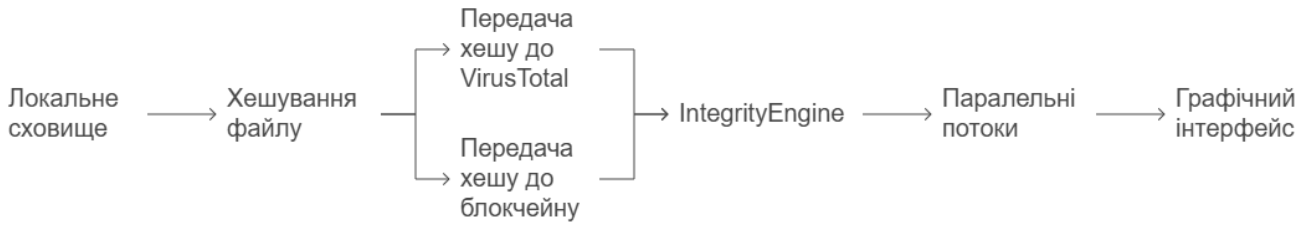

Рисунок 3.2 – Процес двоступеневої верифікації та функції IntegrityEngine

У кінцевому рахунку ця архітектура не просто сукупність модулів, а вдумливий синтез перевірених практик, що організують процес перевірки файлів як ретельно сплановану подорож від моменту виявлення зміни до остаточного вердикту про її безпечність.

Такий підхід гарантує, що в будь-який момент можна відтворити хронологію подій, довести достовірність даних і зберегти бездоганну ланцюжок довіри від локального кешу до блокчейн-реєстру та глобального антивірусного аналізу.

У цій гармонії технологічного ансамблю й народжується справжня опора кібернадійності.

3.5 Опис програми моніторингу цілісності файлів з використанням блокчейну

Представлена система програмного забезпечення є інструментом нового покоління, створеним для автоматизованого моніторингу та перевірки цілісності файлів у цифровому середовищі. Її ключове призначення полягає в тому, щоб забезпечити можливість своєчасної та достовірної верифікації автентичності файлів, а також своєчасного виявлення будь-яких несанкціонованих змін у їхній структурі чи вмісті. Така функціональність є особливо актуальною в умовах зростання масштабів використання програмного забезпечення в критичних галузях, а також в епоху посиленої загрози кібератак і поширення цифрового піратства.

Одним із центральних викликів, що стоять перед розробниками та користувачами сучасних інформаційних систем, є гарантування достовірності походження та незмінності цифрових об’єктів у часі. У відповідь на це завдання запропонована система використовує технологію блокчейн. Завдяки своїм властивостям, блокчейн виступає своєрідним «незаперечним нотаріусом» у цифровому середовищі: усі дані, що в ньому зберігаються, є криптографічно захищеними, незмінними та доступними для подальшої перевірки. У межах роботи системи при додаванні нового файлу його криптографічне хеш-значення обчислюється відповідно до заданого алгоритму (наприклад, SHA-256) і фіксується у відповідному блоці. Це хеш-значення є унікальним для кожного стану файлу, тому будь-які зміни вмісту миттєво порушують відповідність. Завдяки цьому зберігається не просто інформація про сам файл, а й цифровий слід, що дозволяє з високою точністю підтвердити, що файл залишався незмінним з моменту його реєстрації.

Окремої уваги заслуговує модуль, що відповідає за інтеграцію системи із зовнішнім сервісом VirusTotal. Цей сервіс відомий у сфері кібербезпеки як авторитетне джерело аналітики з виявлення шкідливого програмного забезпечення. Сервіс базується на колективному аналізі, який здійснюється десятками антивірусних двигунів з усього світу. Вбудована функціональність перевірки файлів за допомогою VirusTotal дозволяє не лише зафіксувати факт незмінності, а й

отримати додаткову оцінку репутації конкретного файлу. Таким чином, система не просто технічно перевіряє, чи змінено файл, а й надає змогу оцінити, чи не є сам файл потенційно небезпечним або підозрілим з точки зору міжнародних антивірусних експертів.

У своїй сукупності, всі ці компоненти це хешування, фіксація у блокчейні та перевірка репутації, тобто формують цілісну архітектуру, спрямовану на підвищення довіри до цифрових об'єктів. Завдяки автоматизації ключових етапів перевірки система може бути інтегрована як у корпоративну IT-інфраструктуру, так і в процеси забезпечення захисту авторських прав, цифрової відповідальності розробників, або юридичної верифікації даних у рамках електронного документообігу.

Система складається з кількох ключових компонентів, кожен з яких виконує певну функцію:

- Клас IntegrityRecord

У системах, орієнтованих на захист та перевірку цілісності файлів, необхідно не лише виявляти зміни у даних, а й належним чином фіксувати інформацію про кожен факт перевірки. Саме з цією метою до архітектури системи було включено клас IntegrityRecord, який відіграє роль універсального контейнера для структурованого зберігання результатів перевірки цілісності.

По суті, IntegrityRecord це об'єкт, що акумулює ключову метаінформацію про стан файлу в певний момент часу. Він включає такі основні параметри, як шлях до файлу у файловій системі (атрибут file_path) та його криптографічне хеш-значення (hash_value), що є унікальним відбитком вмісту файлу. Саме хеш виступає об'єктивним критерієм визначення змін у файлі: навіть незначне редагування призведе до повністю іншого значення хешу.

Ще одним важливим елементом запису є мітка часу (timestamp). Вона вказує, коли саме було здійснено обчислення хешу або сформовано запис. Якщо під час створення об'єкта IntegrityRecord час не передається явно, система автоматично фіксує поточний момент, що гарантує точність і хронологічну впорядкованість усіх записів.

Додатково передбачено можливість зберігати інформацію про цифрового підписанта. Атрибут signer може містити адресу чи ідентифікатор того суб’єкта (наприклад, користувача, програми або сервісу), який здійснив підписування файлу або підтвердив його достовірність. Це особливо актуально в системах, де перевірка цілісності інтегрується з механізмами довіри, такими як блокчейн, PGP чи корпоративна PKI-інфраструктура. Атрибут signature, у свою чергу, містить безпосередньо криптографічний підпис, якщо такий було згенеровано.

Окремої уваги заслуговує метод to_json(). Він забезпечує серіалізацію об'єкта, тобто перетворення всієї структури в стандартний формат JSON. Це дозволяє зручно зберігати або передавати інформацію про перевірку іншим компонентам системи або зовнішнім сервісам. Завдяки цьому, IntegrityRecord можна безпосередньо зберігати в базі даних, надсилати через мережу або логувати для подальшого аудиту.

Таким чином, IntegrityRecord є не просто допоміжною структурою, а критичним будівельним блоком системи, який забезпечує прозорість, простежуваність і юридичну значущість результатів перевірки цілісності. Його використання дозволяє формалізувати процес фіксації стану файлів і забезпечити належний рівень доказовості у випадках виникнення інцидентів безпеки чи розслідувань.

Фрагмент коду:

```
class IntegrityRecord:
    """JSON object holding file hash, timestamp, signer address, optional PGP signature."""

    def __init__(self, file_path, hash_value, timestamp=None, signer=None, signature=None):
        self.file_path = file_path
        self.hash_value = hash_value
        self.timestamp = timestamp or datetime.now().isoformat()
        self.signer = signer or "0x0000000000000000000000000000000000000000"
        self.signature = signature
```

```
def to_json(self):
    return {
        "file_path": self.file_path,
        "hash_value": self.hash_value,
        "timestamp": self.timestamp,
        "signer": self.signer,
        "signature": self.signature
    }
```

- Клас Database

У контексті інформаційної системи контролю цілісності файлів, клас Database виконує фундаментальну роль зберігача станів, історії перевірок та результатів взаємодії з іншими компонентами. Саме ця складова відповідає за збереження інформації про всі ключові події, пов’язані з файлами: коли вони були проаналізовані, які хеші було обчислено, чи виявлено зміни, а також які відповіді були отримані від зовнішніх систем, таких як VirusTotal чи блокчейн.

Упровадження окремого класу для взаємодії з базою даних дозволяє централізувати доступ до всіх даних, що накопичуються під час роботи системи. Це спрощує обробку інформації, забезпечує структурованість записів та дозволяє реалізувати більш складну логіку.

Наприклад, історичне відстеження змін по кожному окремому файлу або аналітику на рівні системи загалом.

Під час ініціалізації об’єкта Database встановлюється з’єднання з конкретною СКБД (системою керування базами даних). Це може бути як легковажна SQLite для тестового середовища, так і більш потужна PostgreSQL чи MySQL для продуктивного використання. Залежно від вибраної технології, адаптер забезпечує коректну взаємодію, з урахуванням підключення, захисту даних і оптимізації запитів.

Однією з ключових функцій цього класу є збереження хеш-значень файлів. У реальній експлуатації це дозволяє системі перевірити, чи змінився файл з моменту останньої перевірки, порівнюючи актуальний хеш із тим, що вже зберігається в базі. Таким чином реалізується базовий рівень моніторингу цілісності.

Крім того, через клас Database зберігаються результати перевірок із зовнішніх джерел. Наприклад, якщо для певного файлу було отримано відповідь від VirusTotal, вона також може бути збережена, що дозволяє уникнути дублювання запитів та прискорює подальші перевірки. Те саме стосується і відповідей з блокчейн-мережі: зафіксовані результати можуть служити як додатковий етап звірки або навіть як доказ у разі виявлення інциденту.

Ще однією важливою функцією є ведення журналу подій (логів). Такий журнал дозволяє в режимі реального часу або постфактум аналізувати поведінку системи, виявляти нетипові ситуації або реагувати на підозрілі зміни. У цьому сенсі Database служить не лише сховищем, а й основою для аудиту, контролю відповідності та підзвітності.

У сукупності, клас Database є критичним елементом системної архітектури, що забезпечує постійність, надійність і відтворюваність усіх дій, пов’язаних із перевіркою цілісності даних. Без надійного сховища система втрачає здатність до довгострокового аналізу та контролю, а отже, і свою ефективність у забезпеченні безпеки.

Фрагмент коду:

```
class Database:
    """SQLite database for storing file hashes and verification status"""

    def __init__(self, db_path='acim.db'):
        self.db_path = db_path
        self.conn = sqlite3.connect(db_path)
        self.create_tables()

    def create_tables(self):
```

```
cursor = self.conn.cursor()

# Create watched_paths table
cursor.execute('''
        CREATE TABLE IF NOT EXISTS watched_paths
        (
            id
            INTEGER
            PRIMARY
            KEY,
            path
            TEXT
            UNIQUE,
            recursive
            BOOLEAN
        )''')

# Create hash_history table
cursor.execute('''
        CREATE TABLE IF NOT EXISTS hash_history
        (
            id
            INTEGER
            PRIMARY
            KEY,
            path_id
            INTEGER,
            algo
            TEXT,
            hash
```

```
            TEXT,
            ts
            DATETIME,
            FOREIGN
            KEY
        (
            path_id
        ) REFERENCES watched_paths
        (
            id
        )
            )''')

    # Create verification table
    cursor.execute('''
            CREATE TABLE IF NOT EXISTS verification
            (
                hash
                TEXT
                PRIMARY
                KEY,
                vt_result
                TEXT,
                chain_status
                TEXT,
                last_checked
                DATETIME
            )''')
```

- Клас BlockchainAdapter

Клас BlockchainAdapter відіграє роль зв’язуючої ланки між внутрішньою системою забезпечення цілісності даних та блокчейн-мережею, яка використовується як надійне середовище для зберігання контрольних відбитків файлів. Сам по собі блокчейн це розподілений реєстр, де інформація фіксується в незмінному вигляді. Тож інтеграція з такою технологією дозволяє гарантувати, що одного разу записані дані (наприклад, хеш файлу) залишаються недоторканими й можуть бути перевірені будь-коли в майбутньому, без ризику фальсифікації.

У рамках даної архітектури адаптер виконує низку важливих функцій, головною з яких є можливість реєстрації (тобто публікації) цифрових відбитків файлів у блокчейні. Така публікація означає, що система фіксує певний хеш як еталон і це доказ того, що файл існував у такому вигляді на момент запису. Надалі це дозволяє точно встановити, чи зазнавав файл будь-яких змін, оскільки навіть найменша модифікація змінить його хеш-значення.

Окрім цього, BlockchainAdapter також забезпечує механізм зворотної перевірки. Тобто система має змогу звернутися до блокчейн-мережі для того, щоб звірити поточне хеш-значення файлу з тим, що було раніше збережено. Якщо значення співпадають, то файл визнається автентичним, якщо ж ні, то це свідчить про потенційне порушення цілісності, можливо, викликане навмисною модифікацією, технічною помилкою або зовнішнім втручанням.

У процесі взаємодії з блокчейном адаптер дотримується певних стандартів з'єднання, автентифікації та шифрування. Такі заходи необхідні для того, щоб убезпечити саму комунікацію із мережею, а також запобігти несанкціонованому доступу до інтерфейсів, через які здійснюється запис або перевірка даних.

Таким чином, BlockchainAdapter це не просто допоміжний компонент, а ключовий елемент архітектури, що додає системі додатковий рівень довіри. Використання блокчейну як опорного джерела істини дозволяє будувати серйозні механізми цифрової відповідальності: кожна зафіксована дія має свій слід, який не можна стерти або підробити. Саме тому такий підхід дедалі частіше використовується у сферах, де цілісність інформації є критично важливою, а зокрема

в банківській справі, охороні здоров’я, електронному документообігу та державному управлінні.

Фрагмент коду:

```
class BlockchainAdapter:
    """Simulates blockchain interaction for hashes"""

    def __init__(self):
        # For demo purposes, we'll simulate blockchain with a dictionary
        self.published_hashes = {}

    def exists(self, file_hash):
        """Check if hash exists on chain"""
        return file_hash in self.published_hashes
    def publish(self, file_hash, metadata):
        """Simulate publishing hash to blockchain"""
        self.published_hashes[file_hash] = {
            'metadata': metadata,
            'timestamp': datetime.now().isoformat(),
            'tx_hash': f"0x{os.urandom(32).hex()}"
        }
        logger.info(f"Published hash {file_hash[:10]}... to blockchain")
        return self.published_hashes[file_hash]['tx_hash']

    def get_transaction_receipt(self, tx_hash):
        """Simulate getting transaction receipt"""
        for hash_val, data in self.published_hashes.items():
            if data['tx_hash'] == tx_hash:
                return {
                    'status': 1,  # Success
                    'blockNumber': 12345678,
```

```
                'timestamp': data['timestamp']
            }
        return None
```

- Клас VirusTotalAdapter

Клас VirusTotalAdapter виконує важливу роль посередника між системою контролю цілісності файлів та зовнішнім сервісом аналізу загроз, а саме VirusTotal. У сучасному цифровому середовищі, де кількість нових зразків шкідливого програмного забезпечення зростає щодня, ефективна інтеграція з такими платформами стає ключовою умовою оперативної оцінки потенційних загроз. Саме через цей адаптер система може вийти за межі локального аналізу й скористатися колективним досвідом світової спільноти кібербезпеки.

Початкове налаштування адаптера відбувається в момент створення об'єкта, коли задається унікальний API-ключ користувача. Цей ключ необхідний для автентифікації кожного запиту до сервісу VirusTotal. Окрім ключа, одразу встановлюються базова адреса для доступу до API та необхідні заголовки запиту, які відповідають технічним вимогам зовнішнього сервера. Усе це дозволяє налаштувати стабільний канал взаємодії з платформою.

Основним функціональним елементом адаптера є метод, що дозволяє перевірити файл за його хеш-значенням. Ідея проста: замість того, щоб передавати сам файл (що було б ресурсозатратним і потенційно небезпечним), система обчислює його хеш і звертається до VirusTotal із запитом щодо репутації саме цього цифрового ідентифікатора. У відповідь повертається агрегована оцінка на основі результатів аналізу різноманітними антивірусними рушіями.

Для підвищення ефективності та зменшення навантаження на API, у класі реалізовано елементарний механізм кешування результатів. Якщо той самий файл уже перевірявся протягом останніх 24 годин, результат попередньої перевірки зберігається й повторно використовується без нового звернення до сервісу. Такий

підхід дозволяє оптимізувати використання обмеженого ресурсу (кількості дозволених запитів) і водночас пришвидшити роботу системи.

З технічного погляду, метод обробляє кілька можливих сценаріїв: файл може бути визнаний «чистим», «підозрілим» або «шкідливим», залежно від кількості спрацювань антивірусних рушіїв. Якщо файл ще не відомий сервісу, результат повертається як «невідомий». У разі виникнення проблем із з'єднанням або помилок у відповіді API, передбачено обробку винятків із відповідним логуванням події.

Таким чином, VirusTotalAdapter це не просто технічна надбудова, а важливий міст між внутрішнім середовищем системи захисту та глобальним інформаційним полем. Його наявність дозволяє оперативно отримувати об'єктивну оцінку потенційних ризиків, базуючись на даних з багатьох антивірусних лабораторій світу.

Фрагмент коду:

```
class VirusTotalAdapter:
    """Interface with VirusTotal API"""

    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://www.virustotal.com/api/v3"
        self.headers = {
            "x-apikey": self.api_key,
            "Accept": "application/json"
        }
        self.cache = {}  # Simple cache to avoid repeated requests

    def check_file_hash(self, file_hash):
        """Query VirusTotal for file reputation by hash"""
        # Check cache first
        if file_hash in self.cache:
            cache_time, result = self.cache[file_hash]
            if datetime.now() - cache_time < timedelta(hours=24):
```

```
            logger.info(f"Using cached VirusTotal result for {file_hash[:10]}...")
            return result

        try:
            url = f"{self.base_url}/files/{file_hash}"
            logger.info(f"Querying VirusTotal for {file_hash[:10]}...")

            response = requests.get(url, headers=self.headers)

            if response.status_code == 200:
                data = response.json()
                stats = data.get('data', {}).get('attributes', {}).get('last_analysis_stats', {})

                # Determine result based on malicious detections
                malicious = stats.get('malicious', 0)
                suspicious = stats.get('suspicious', 0)

                if malicious > 0:
                    result = "malicious"
                elif suspicious > 0:
                    result = "suspicious"
                else:
                    result = "clean"

                # Cache the result
                self.cache[file_hash] = (datetime.now(), result)
                return result
            elif response.status_code == 404:
                # File not found in VirusTotal
                self.cache[file_hash] = (datetime.now(), "unknown")
```

```
            return "unknown"
        else:
            logger.error(f"VirusTotal API error: {response.status_code}")
            return "error"
    except Exception as e:
        logger.error(f"Error checking VirusTotal: {str(e)}")
        return "error"
```

- Клас IntegrityEngine

Клас `IntegrityEngine` виконує роль своєрідного "мозкового центру" системи контролю цілісності файлів. Саме він відповідає за те, щоб усе працювало узгоджено: обчислюються хеші файлів, перевіряється їхня цілісність, а також забезпечується взаємодія з зовнішніми сервісами та внутрішніми модулями.

Коли створюється об'єкт цього класу (через метод `__init__()`), йому передаються посилання на інші важливі частини системи: базу даних (`Database`), адаптер для взаємодії з сервісом VirusTotal (`VirusTotalAdapter`) і адаптер для роботи з блокчейном (`BlockchainAdapter`). Таким чином, `IntegrityEngine` одразу має доступ до всіх необхідних ресурсів для виконання своїх завдань.

Метод `compute_file_hash()` відповідає за створення хешу певного файлу. Це своєрідний цифровий відбиток, який однозначно ідентифікує вміст файлу. Метод дозволяє використовувати різні алгоритми хешування.

Наприклад, SHA-256 або інші, залежно від потреби.

Окрему увагу заслуговує метод `verify_file()`. Він реалізує повну перевірку файлу на цілісність і потенційні загрози. Спочатку він обчислює хеш, далі звіряє його з інформацією, що вже зберігається в блокчейні (як джерелі незмінних записів), і додатково звертається до сервісу VirusTotal, щоб перевірити файл на відомі шкідливі сигнатури. Це дозволяє комплексно оцінити стан файлу з погляду безпеки.

Метод `publish_to_blockchain()` дозволяє додати новий хеш у блокчейн. Це корисно, коли потрібно зафіксувати поточний стан файлу, аби потім мати можливість

перевірити, чи не зазнав він змін. Таке збереження в блокчейні виступає надійним доказом автентичності файлу в майбутньому.

Фрагмент коду:

```
class IntegrityEngine:
    """Coordinates hash calculation, verification and adapters"""

    def __init__(self, db, vt_adapter, blockchain_adapter):
        self.db = db
        self.vt_adapter = vt_adapter
        self.blockchain_adapter = blockchain_adapter

    def compute_file_hash(self, file_path, algorithm='sha256'):
        """Compute hash of a file"""
        try:
            hash_func = getattr(hashlib, algorithm)()

            with open(file_path, 'rb') as f:
                # Read file in chunks to handle large files
                for chunk in iter(lambda: f.read(4096), b''):
                    hash_func.update(chunk)

            return hash_func.hexdigest()
        except Exception as e:
            logger.error(f"Error computing hash for {file_path}: {str(e)}")
            return None

    def verify_file(self, file_path, path_id=None):
        """Verify file integrity against blockchain and VirusTotal"""
        if not os.path.isfile(file_path):
            return {
```

```
            'status': 'error',
            'message': 'File not found'
        }

    # Compute hash
    file_hash = self.compute_file_hash(file_path)
    if not file_hash:
        return {
            'status': 'error',
            'message': 'Could not compute hash'
        }

    # If path_id not provided, get it or create it
    if not path_id:
        path_id = self.db.add_watched_path(file_path, False)

    # Store hash in history
    self.db.add_hash_record(path_id, 'sha256', file_hash)

    # Check verification status
    verification = self.db.get_verification(file_hash)
```

3.6 Розробка методу автоматичного моніторингу цілісності файлів для захисту авторського права з використанням технології блокчейн.

У процесі виконання завдань магістерської роботи було здійснено розробку спеціалізованого програмного забезпечення, що поєднує сучасні технології захисту інформації з доступним для кінцевого користувача інтерфейсом. Програма створена з використанням мови програмування Python та оснащена графічним інтерфейсом, що забезпечує зручність взаємодії з користувачем незалежно від його технічної

підготовки. Основна мета розробки полягає у підвищенні рівня захисту авторських прав шляхом інтеграції традиційних засобів контролю цілісності файлів із новітніми блокчейн-рішеннями.

Унікальність запропонованого підходу полягає в поєднанні трьох незалежних механізмів перевірки, кожен із яких спрямований на виконання окремої задачі. Зокрема, реалізовано:

1. Локальну перевірку хеш-сум файлів для виявлення будь-яких змін у структурі даних;

2. Відправку хешів на онлайн-сервіс VirusTotal, що дозволяє ідентифікувати потенційно шкідливе або змінене програмне забезпечення;

3. Публікацію отриманих хешів у децентралізованій блокчейн-мережі для створення постійного, незмінного запису, який може слугувати доказом авторства та цілісності цифрового продукту.

Такий підхід набуває особливої актуальності в контексті захисту авторських прав на цифрові продукти, зокрема програмне забезпечення, цифрові твори мистецтва, музичні композиції, 3D-моделі, тексти, сценарії, презентації тощо. Наприклад, незалежний розробник програмного забезпечення, який не має ресурсу для патентування своєї розробки або реєстрації авторських прав у державних органах, може використати запропоновану програму для створення цифрового сліду, тобто унікального хешу, який буде опублікований у блокчейн. Оскільки блокчейн є незмінним і публічно доступним реєстром, така публікація може виступати у якості доказу того, що саме дана особа першою створила і оприлюднила відповідний цифровий продукт.

Крім того, це рішення може бути корисним для дизайнерів, які створюють цифрові шаблони, UI/UX-елементи, логотипи чи візуальні айдентики. Після завершення роботи над певним елементом, дизайнер може зафіксувати його цифрову ідентичність шляхом створення хешу та публікації у блокчейні. У разі виникнення спірних ситуацій щодо авторства, наявність такого запису стане аргументованим підтвердженням права власності.

Ще одним прикладом є створення навчальних матеріалів або електронних підручників. Освітяни, викладачі чи незалежні автори, які займаються розробкою оригінального контенту, можуть використовувати цей програмний продукт для збереження цілісності створених документів і їх захисту від несанкціонованого копіювання чи модифікації. Опублікувавши хеш документа в блокчейні, автор має надійний механізм фіксації факту створення, який є доступним, прозорим та юридично значущим.

Таким чином, сфера застосування розробленого інструменту є надзвичайно широкою, охоплюючи як індивідуальних розробників та авторів, так і малі команди або стартапи, яким важливо швидко та надійно захистити результати своєї праці без потреби у складних та витратних юридичних процедурах. У результаті користувач отримує інструмент, що не лише забезпечує автоматизований моніторинг змін у файлах, але й слугує важливою складовою загальної системи захисту інтелектуальної власності.

Основна сторінка розробленої програми виконує роль центрального елемента взаємодії користувача з програмним забезпеченням та надає широкий спектр функціональних можливостей для здійснення моніторингу цілісності файлів і папок. Інтерфейс основного вікна розроблений таким чином, щоб забезпечити зручність, логічність і швидкий доступ до ключових операцій, що виконуються у процесі роботи системи.

На даній сторінці користувач має змогу виконувати низку основних дій, кожна з яких відповідає за окремий етап у процесі налаштування та реалізації моніторингу:

1. Додавання папки або директорії до моніторингу за допомогою відповідної кнопки або пункту меню користувач може обрати потрібну директорію, яка підлягатиме постійному або періодичному контролю. Це дозволяє здійснювати моніторинг не окремих файлів, а цілих структур, що особливо актуально у випадках, коли мова йде про проєкти з великою кількістю пов’язаних між собою файлів (наприклад, програмні репозиторії, пакети документації, графічні ресурси тощо).

2. Додавання окремого файлу до моніторингу, ця функція забезпечує можливість працювати точково, контролюючи лише конкретні, найбільш критичні

об’єкти. Зазвичай це застосовується до виконуваних файлів програм, конфігураційних документів або іншого роду цифрових активів, зміни в яких можуть мати значний вплив на систему в цілому.

3. Видалення папки або файлу з переліку об’єктів моніторингу, користувач має змогу у будь-який момент видалити той чи інший елемент із переліку об’єктів, які підлягають перевірці. Це дозволяє динамічно адаптувати налаштування під конкретні потреби, наприклад, у разі зміни структури даних чи завершення робіт над певним проєктом.

4. Сканування обраного файлу або директорії, одним із ключових інструментів є функція негайної перевірки. Вона дозволяє вручну ініціювати процес хешування та звірки поточного стану об’єкта із попередньо зафіксованим значенням. Це є надзвичайно корисним при проведенні періодичних аудитів або перевірок за запитом адміністратора чи користувача.

5. Публікація хешу файлу в блокчейн, після отримання хешу будь-якого об’єкта (файлу чи каталогу), користувач може здійснити його публікацію у блокчейн-мережі. Ця функція є ключовою в контексті захисту авторського права, оскільки створює незмінний запис про існування об’єкта у певний момент часу, що може бути використано як доказ у разі виникнення спору щодо авторства чи права власності на цифровий контент.

Крім основних функцій керування моніторингом, інтерфейс сторінки також містить важливу інформацію, яка відображається в реальному часі. Зокрема:

- Місце розташування файлу або папки, що перебуває під контролем системи, виводиться у вигляді абсолютного шляху на файловій системі користувача. Це дозволяє легко ідентифікувати об’єкт, з яким проводиться робота.
- Статус моніторингу, який сигналізує про поточний стан об’єкта: чи зазнав він змін, чи є дані цілісними, або ж було зафіксовано порушення. Це надає користувачу оперативну інформацію про безпеку цифрового середовища.
- Кількість файлів, що моніторяться у вибраній директорії. У випадку, якщо до контролю додано цілу папку, система автоматично підраховує кількість

вкладених об’єктів, включаючи підкаталоги. Це дозволяє оцінити масштаб моніторингу та, при потребі, оптимізувати налаштування.

- Вікно сповіщень, це окрема область інтерфейсу, у якій відображаються всі важливі події, пов’язані з роботою програми.

Сюди входять повідомлення про додавання або видалення об’єктів, зміни у хешах, результати сканування, а також технічна інформація, корисна для адміністратора або відповідальної особи. Завдяки цій панелі користувач має змогу відстежувати повну історію дій, що виконувались у системі, і швидко реагувати на будь-які інциденти.

Таким чином, основна сторінка програми виступає не лише точкою входу до системи, але й забезпечує повний функціональний контроль над усіма ключовими процесами моніторингу, управління даними та захисту цифрової власності.

Логічно структурований інтерфейс дозволяє легко орієнтуватися у функціоналі, а високий рівень автоматизації гарантує ефективність і точність кожної операції, що виконується.

На рисунку 3.3 зображено початковий екран програми.

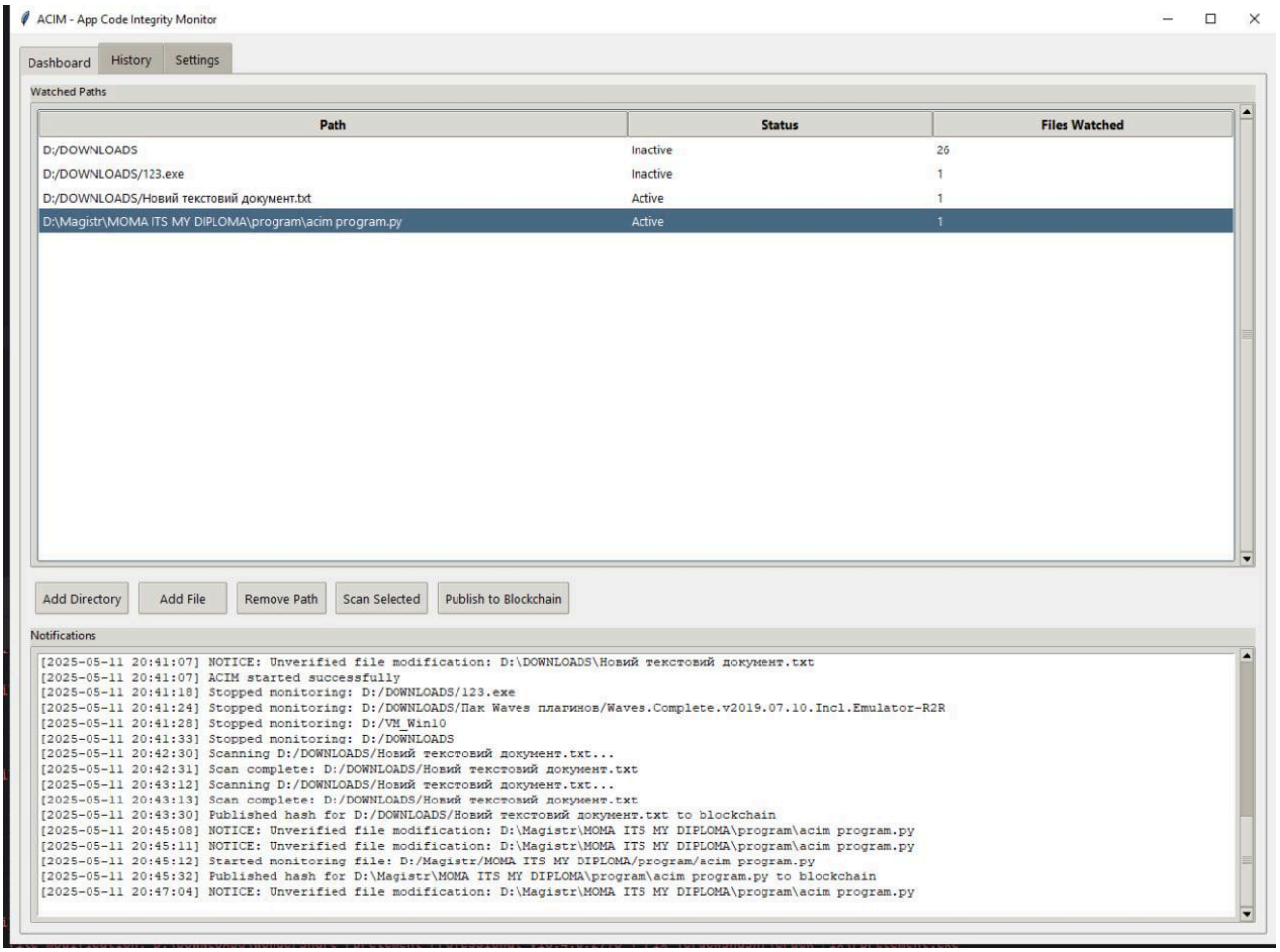

Рисунок 3.3 – Вікно dashboard

Інтуїтивно зрозумілий графічний інтерфейс, реалізований у програмі, дозволяє швидко здійснювати необхідні налаштування, додавати файли чи директорії до моніторингу, переглядати статуси перевірок та публікувати хеші безпосередньо з середовища програми. Це значно спрощує процес експлуатації системи та відкриває можливості її використання у повсякденній практиці незалежно від рівня технічної компетенції користувача.

Однією з ключових функціональних складових програмного забезпечення, розробленого у межах магістерської роботи, є вкладка «Історія», яка забезпечує зручний доступ до детального журналу перевірок та подій, пов’язаних з моніторингом обраних файлів та директорій.

Даний інтерфейсний елемент реалізовано з метою надання користувачу повної, структурованої інформації про усі дії, що відбувались у процесі контролю цілісності,

а також для аналітичного огляду змін, результатів перевірок і взаємодії з блокчейн-мережею.

У вкладці «Історія» відображаються наступні ключові параметри для кожного файлу, що був доданий до системи моніторингу:

1. Час зняття хешу, тобто фіксується точна дата та час, коли для конкретного файлу було згенеровано його хеш-суму. Це дозволяє точно визначити момент останньої перевірки та встановити хронологічну послідовність змін у контрольованих об’єктах.

2. Значення хешу, тобто відображається у вигляді унікального цифрового ідентифікатора, отриманого за допомогою алгоритму хешування (наприклад, SHA-256). Хеш виконує роль цифрового відбитка файлу, який дозволяє однозначно ідентифікувати стан даного об'єкта у конкретний момент часу. У разі навіть найменшої зміни у вмісті файлу його хеш-сума змінюється, що забезпечує високу чутливість до модифікацій.

3. Статус за результатами перевірки на платформі VirusTotal, після отримання хешу система автоматично надсилає його до сервісу VirusTotal для аналізу. У вкладці історії відображається результат цієї перевірки з вказанням одного з трьох можливих статусів:

- Чистий, тобто файл не містить шкідливих компонентів згідно з аналізом антивірусних двигунів платформи;
- Зловмисний, тобто хеш файлу відповідає зразку, що визнаний шкідливим або небезпечним;
- Невідомий, тобто хеш ще не внесено до бази VirusTotal, що може свідчити про новизну файлу або обмежену розповсюдженість.

4. Статус щодо блокчейн-мережі, тобто додатково зазначено, чи був даний хеш опублікований у блокчейні. Якщо так, то статус відповідно відображає факт реєстрації у публічному децентралізованому реєстрі. Це дозволяє швидко перевірити, чи має файл юридично значущий цифровий слід, який можна використати у випадку виникнення спорів щодо авторства або змін у файлі.

Крім перегляду інформації по окремих файлах, вкладка «Історія» також надає можливість вибору директорії, після чого здійснюється автоматичне відображення всіх файлів, які належать до цієї папки, разом з відповідною історичною інформацією по кожному з них. Такий підхід дозволяє зручно аналізувати великі масиви даних та отримувати повну картину щодо змін у структурі проєктів, що контролюються.

Візуально вкладка реалізована у вигляді таблиці або списку, що містить згадані параметри у вигляді колонок. Користувач має змогу фільтрувати, сортувати та шукати записи за певними критеріями, що значно підвищує ефективність взаємодії з великим обсягом інформації.

Наприклад, у випадку активного моніторингу великої кількості файлів, які зберігаються у різних директоріях або підлягають частим оновленням, користувачу може бути необхідно швидко виявити останні зміни, що відбулись у системі. Для цього у вкладці «Історія» реалізовано можливість сортування записів за часовою міткою перевірки.

Завдяки цьому функціоналу користувач має змогу, наприклад, в один клік відсортувати усі файли у порядку спадання або зростання часу останнього хешування.

Це дозволяє оперативно ідентифікувати найсвіжіші або найдавніші зміни у контрольованих об’єктах, що є особливо корисним у ситуаціях, коли потрібно виявити, який саме файл зазнав змін протягом певного проміжку часу (наприклад, упродовж останньої доби, тижня чи після розгортання оновлення програмного забезпечення).

Отже, згадані інструменти сортування та пошуку не є лише допоміжними, а виступають важливими елементами функціонального забезпечення системи, які підвищують її ефективність, масштабованість та придатність до використання в умовах реальних задач цифрової безпеки та захисту авторського права.

Таким чином, вкладка «Історія» є потужним аналітичним інструментом, який доповнює основний функціонал програми, забезпечуючи прозорість,

контрольованість та доказову базу для підтвердження авторства, аналізу змін або розслідування інцидентів інформаційної безпеки.

На рисунку 3.4 зображено вікно «Історія».

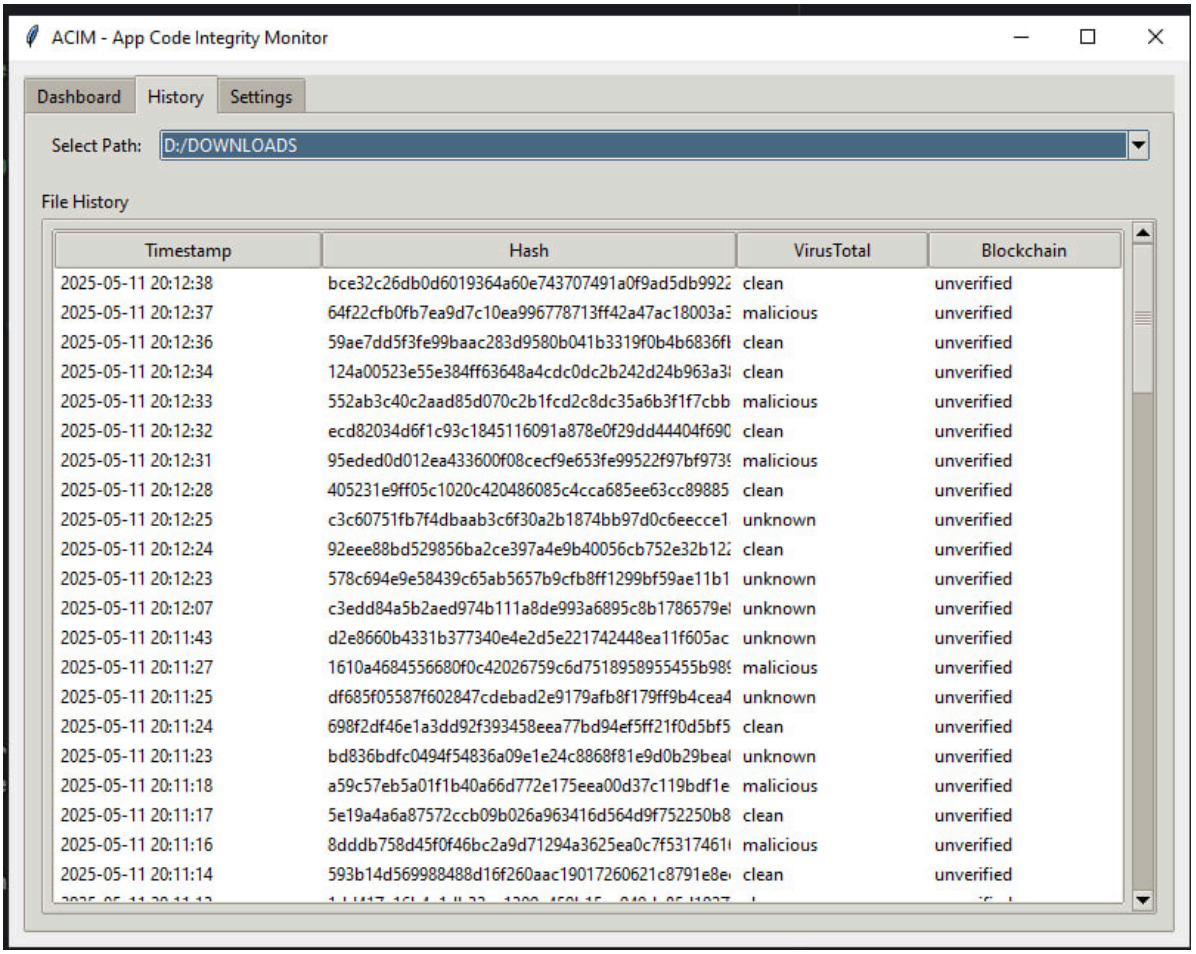

Рисунок 3.4 – Вкладка «Історія»

Після запуску програми відбувається ініціалізація всіх основних компонентів та сервісів, необхідних для коректного функціонування системи моніторингу. Зокрема, одним із ключових етапів стартової фази є автоматичне зчитування даних з локальної бази даних, у якій зберігаються шляхи до файлів та директорій, попередньо доданих користувачем до системи контролю цілісності.

На основі цієї інформації програма автоматично активує процес моніторингу усіх відповідних об'єктів. Це означає, що одразу після запуску здійснюється перевірка наявності вказаних шляхів у файловій системі, ініціюється оновлення

хешів (якщо необхідно), а також активується фонове спостереження за потенційними змінами у вмісті зазначених файлів або папок.

На рисунку 3.5 зображено вікно початку програми та приклад, як виглядає зупинка моніторингу вручну.

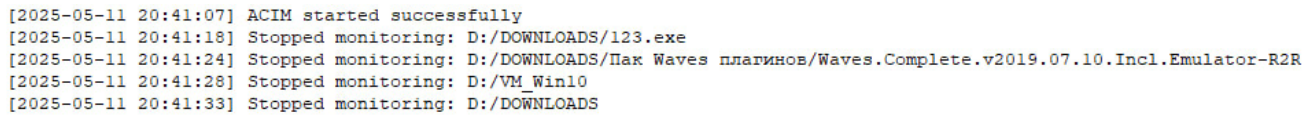

Рисунок 3.5 – Запуск програми

3.7 Вікно налаштувань програми

На рисунку 3.6 представлено інтерфейс вікна налаштувань програми, яке виконує функцію конфігурації основних параметрів роботи програмного забезпечення. Це вікно дозволяє користувачу зручно налаштовувати поведінку програми відповідно до власних потреб або особливостей середовища, в якому програма використовується.

Однією з ключових функцій, що реалізовані в цьому інтерфейсі, є можливість активації автоматичного запуску програми синхронно з операційною системою Windows. У разі ввімкнення цієї опції, програма автоматично стартує разом із запуском ОС, переходячи у фоновий режим роботи та одразу ініціалізуючи моніторинг цілісності усіх файлів і директорій, які були попередньо додані до бази даних. Така можливість дозволяє забезпечити постійний захист цифрових об’єктів без додаткових дій з боку користувача, що особливо важливо у випадках, коли система функціонує у безперервному середовищі, наприклад, на робочих станціях, серверах або в корпоративних системах.

Іншою важливою функцією налаштувань є вибір алгоритму хешування, що використовується для обчислення контрольних сум файлів. За замовчуванням встановлено SHA-256. Проте у разі необхідності користувач може обрати інший алгоритм.

Вікно налаштувань також містить функцію збереження обраних параметрів, що дозволяє зафіксувати конфігурацію і використовувати її при наступних запусках програми без необхідності повторного введення. Це підвищує зручність та економить час користувача, особливо в умовах частого перезапуску або роботи на різних пристроях.

У нижній частині інтерфейсу налаштувань відображено коротку інформацію про саму програму. Тут користувач може знайти опис функціональних можливостей програмного забезпечення, дізнатися поточну версію, ознайомитися з інформацією про використані технології або переглянути контактні дані розробника.

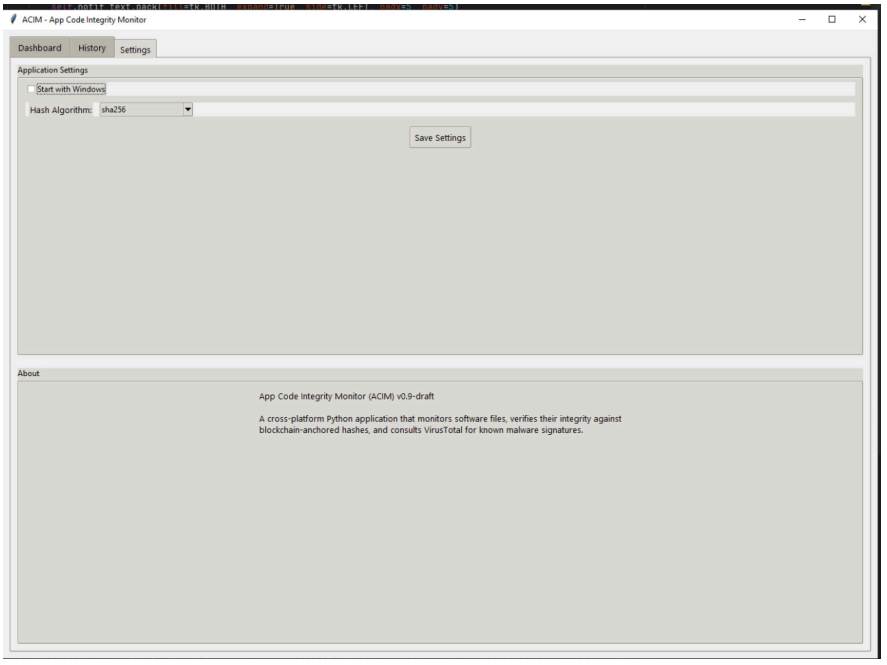

Рисунок 3.6 – Вікно налаштувань програми

Таким чином, вікно налаштувань відіграє роль важливого елементу інтерфейсу програми, що дозволяє здійснювати персоналізацію, оптимізувати запуск та забезпечити стабільну роботу механізмів моніторингу у відповідності до індивідуальних або корпоративних вимог.

3.8 Тестування методу автоматичного моніторингу цілісності файлів

Розроблена програма передбачає функціонал системи сповіщень (нотифікацій), що є важливою складовою з точки зору оперативного інформування користувача або адміністратора про події, які потребують уваги, реагування або подальшого аналізу.

Усі нотифікації виводяться у спеціальному вікні повідомлень на головній сторінці програми, а також можуть бути додатково дубльовані через журнали подій або у вигляді спливаючих системних вікон (за відповідних налаштувань).

Зокрема, система передбачає три основні типи нотифікацій, кожен з яких виконує специфічну функцію в межах загального механізму моніторингу цілісності та безпеки цифрових об’єктів:

1. Порушення цілісності файлу.

Процес інформування адміністратора під час порушення цілісності файлів зображено на рисунку 3.7

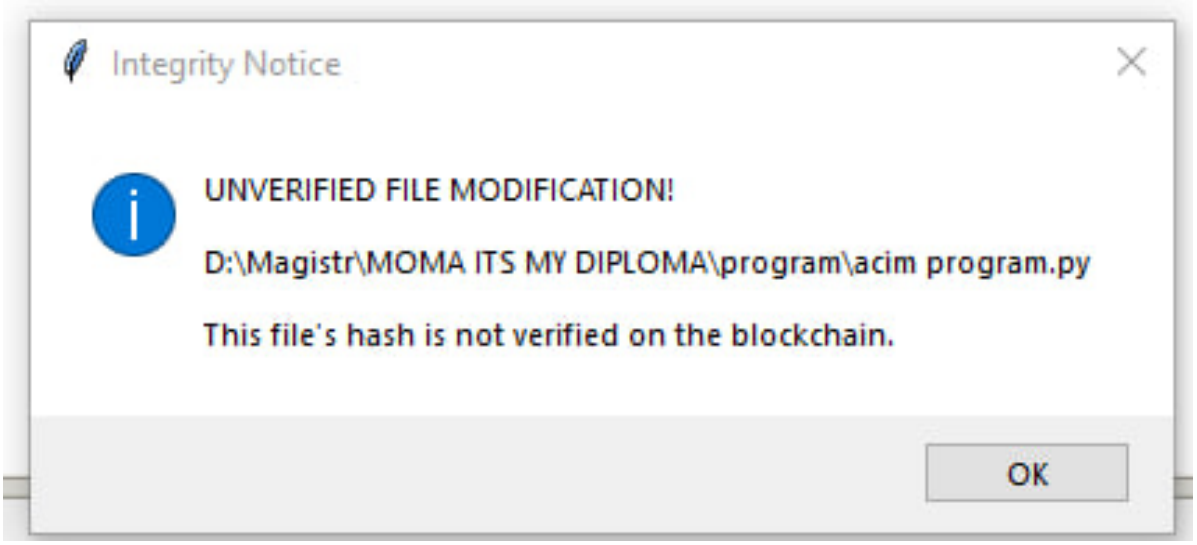

Рисунок 3.7 – Сповіщення про модифікацію файлу

2. Порушення цілісності файлу, що був опублікований у блокчейн.

На рисунку 3.8 зображено, що отримує користувач коли публікує в блокчейн хеш певного файлу.

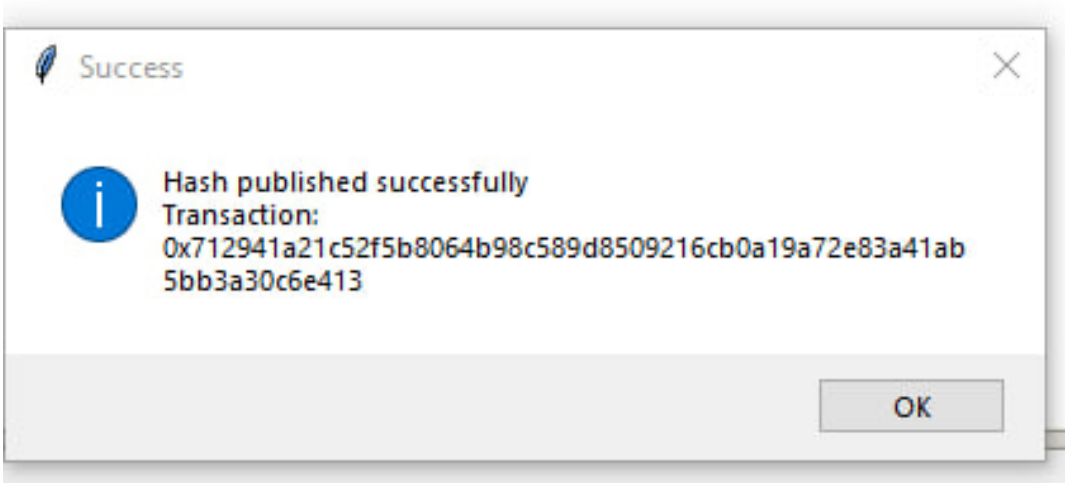

Рисунок 3.8 – Публікація в блокчейн

При порушенні цілісності такого файлу, він отримає статус неверифікованого та вискочить повідомлення, яке зображено на рисунку 3.9.

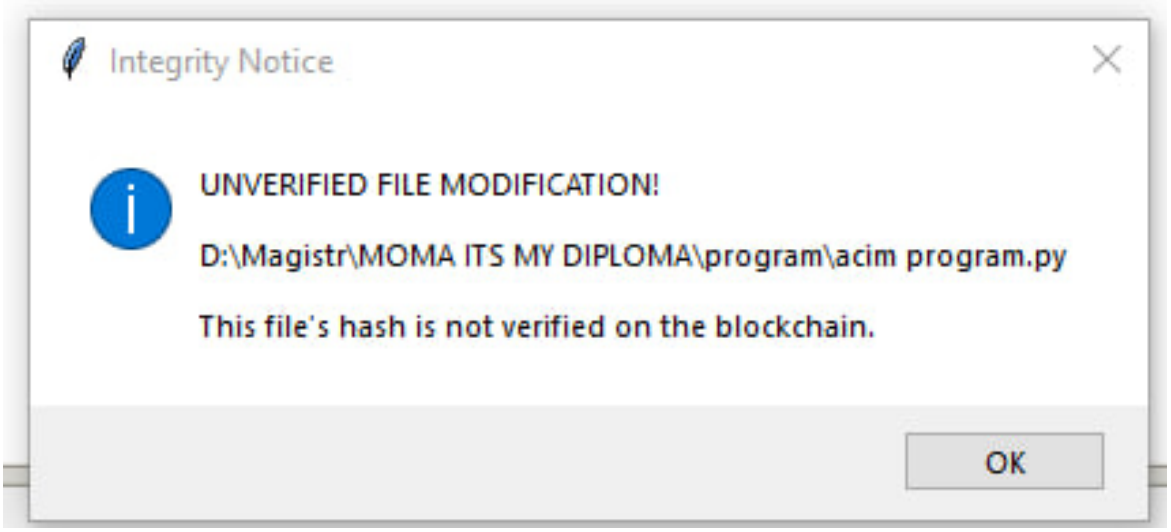

Рисунок 3.9 – Сповіщення про неверифікований файл

3. Виявлення зловмисного файлу у директорії (за даними VirusTotal)

Програма також видає в консоль IDE логи щодо роботи програми.

На рисунках 3.10-3.11 представлено інтерфейс журналу подій програми, який фіксує усі важливі дії, що виконуються під час моніторингу. Зокрема, тут відображаються зміни у контрольованих файлах, включаючи час модифікації, старі

та нові хеш-значення, а також шлях до відповідного об’єкта. Це дозволяє оперативно виявляти втручання або втрату цілісності даних.

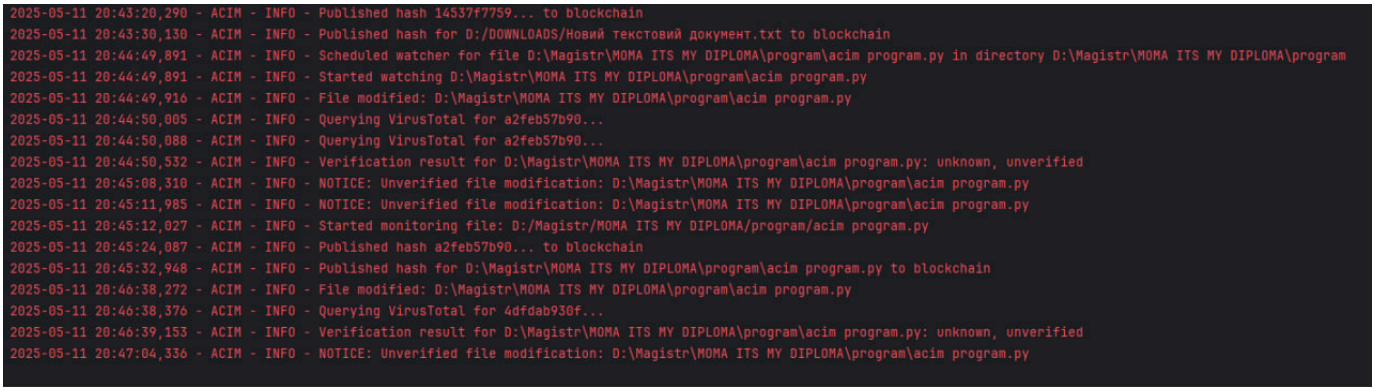

Рисунок 3.10 – Події програми, що логуються в консоль IDE

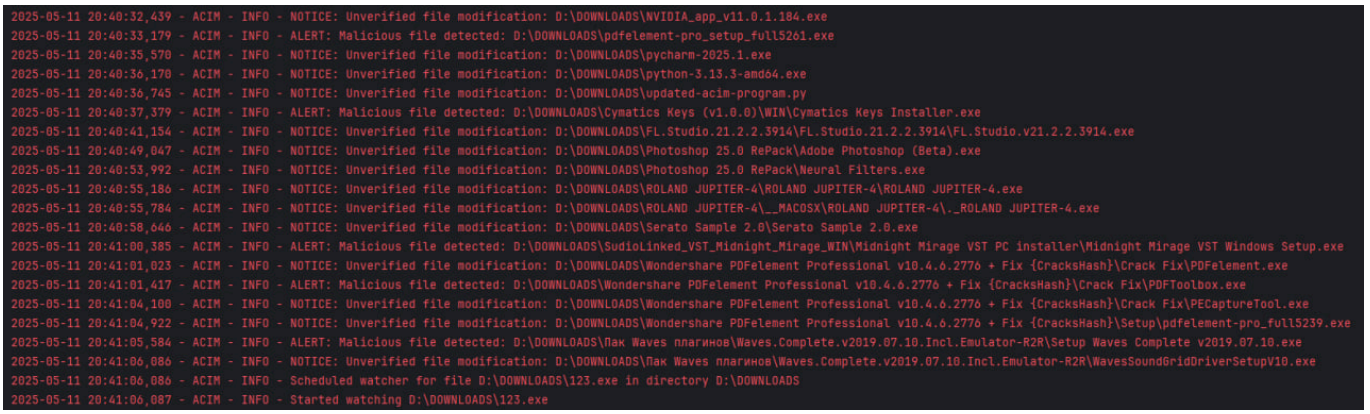

Рисунок 3.11 – Логи, що виводяться в консолі IDE

Окрім змін у файлах, журнал також реєструє звернення до сервісу VirusTotal через API. Для кожного звернення зазначається час, об'єкт перевірки та результат, а саме чи є файл безпечним, зловмисним або невідомим.

Також в логах фіксується запуск сканування, його параметри та отримані результати.

Таким чином, журнал подій виконує роль засобу аудиту та контролю, дозволяючи користувачу чи адміністратору отримати повну хронологію ключових подій системи моніторингу.

На рисунку 3.12 зображено результат тестувань. На рисунку ми бачимо, що ми маємо 4 хеші (файли) з яких 3 невідомі та 1 чистий файл виходячи з даних VirusTotal.

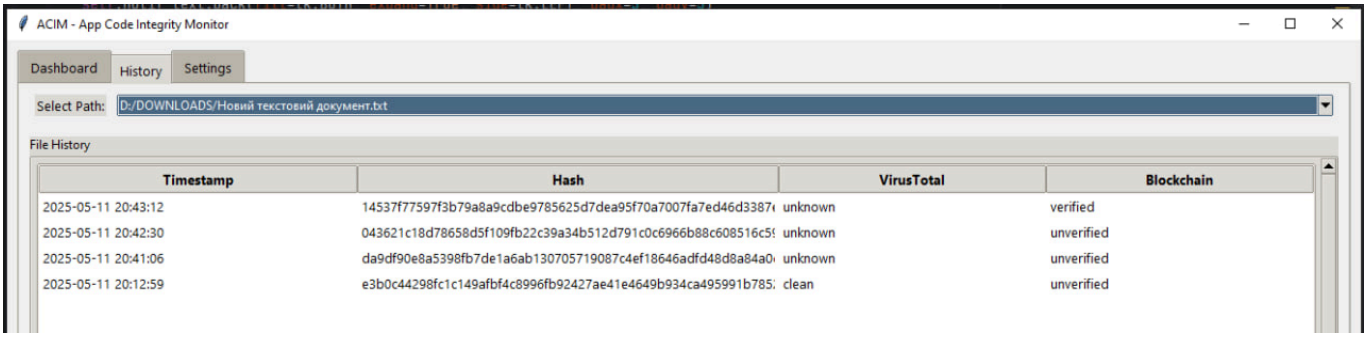

Рисунок 3.12 – Результати тестування

Також оскільки ми опублікували файл в блокчейн ми бачимо статус по блокчейну – верифіковано, що означає, що файл не змінювався з моменту публікації в блокчейн.

Висновки до третього розділу

У третьому розділі було здійснено розробку та практичну реалізацію методу автоматичного моніторингу цілісності файлів, який поєднує кілька технологічних підходів для досягнення високого рівня надійності, безпеки та зручності використання. Основна ідея реалізованого методу полягає у створенні програмного забезпечення, яке здатне в автоматичному режимі виявляти зміни у вмісті файлів, порівнюючи їх контрольні суми з раніше зафіксованими значеннями, а також аналізувати потенційні загрози шляхом інтеграції з зовнішніми сервісами безпеки.

Однією з ключових інновацій є застосування технології блокчейн, яка використовується для публікації хеш-сум файлів. Це дозволяє не лише зберігати інформацію про цілісність у захищеному середовищі, але й забезпечує можливість незалежної верифікації даних будь-яким користувачем. Таким чином, у разі порушення цілісності цифрового об’єкта можна достовірно довести факт втручання, що є особливо важливим у контексті захисту авторських і суміжних прав, а також у середовищах, де критичною є автентичність файлів.

Додатково, в межах програмного рішення реалізовано взаємодію з API сервісу VirusTotal. Це дозволяє автоматично перевіряти хеші файлів на наявність відповідностей до відомих шкідливих об'єктів у базах даних антивірусних компаній. Така перевірка значно підвищує рівень захисту кінцевого користувача, оскільки дає змогу виявляти не лише змінені, а й потенційно небезпечні файли у реальному часі.

Розроблений інтерфейс програми дозволяє легко керувати процесом моніторингу, додавати або видаляти об’єкти, переглядати історію перевірок, результати аналізу та отримувати сповіщення про будь-які порушення. Завдяки цьому реалізований метод може бути ефективно застосований як у побутовому, так і у корпоративному середовищі, де є потреба у постійному контролі за збереженням та автентичністю цифрових даних.

У результаті розробки було створено комплексне рішення, що об’єднує традиційні підходи до контролю контрольних сум із сучасними можливостями блокчейн-технологій та сервісів кібербезпеки, забезпечуючи надійний, масштабований та універсальний інструмент для моніторингу цілісності файлів.

.

ВИСНОВОК

Отже, у роботі запропоновано метод автоматичного моніторингу цілісності файлів, що об’єднує сучасні технології для ефективного захисту авторських прав. Теоретична частина роботи показала, що цифрове піратство дедалі активніше використовує слабкі місця у традиційних підходах, а аналіз ключових законодавчих норм (DMCA, GDPR) засвідчив необхідність гарантування незмінності та прозорості даних без втручання в права користувачів. Саме це обґрунтувало ідею повної автоматизації спостереження за станом файлів та швидкого реагування на будь-які спроби їхнього підмінювання чи копіювання.

Під час виконання роботи виконані усі поставлені завдання. Було досліджено існуючі методи забезпечення цілісності файлів та їхні обмеження. Було розроблено алгоритм автоматичної перевірки цілісності файлів з використанням хеш-функцій. Було створено базу даних для зберігання контрольних сум файлів. Було інтегровано технологію блокчейн до алгоритму перевірки цілісності файлів. Було проведено тестування розробленої системи на реальних даних.

Детальний аналіз існуючих систем виявив, що класичні інструменти контролю цілісності або не дають оперативних сповіщень про зміну файлу, або ж представляють собою окремі модулі без центрального зв’язку між собою. Це призводить до фрагментації процесу захисту: хеш-функції забезпечують лише базову перевірку, платформи на кшталт VirusTotal, забезпечення віддаленого аналізу загроз, а блокчейн-рішення це прозоре, але повільне збереження записів. Така роздробленість суттєво знижує швидкість виявлення загроз і створює прогалини в системі безпеки.

Практична реалізація в третьому розділі демонструє робочу програму з графічним інтерфейсом на базі Tkinter, що моніторить зміни в обраних директоріях у реальному часі за допомогою Watchdog. Кожна подія створення, видалення чи модифікації файлу автоматично тригерить обчислення контрольної суми SHA-256 та порівняння з попереднім значенням. У разі невідповідності хешів інформація

негайно заноситься в блокчейн, тобто це гарантує незмінність запису навіть у разі компрометації локального сховища.

Додатково програма звертається до API VirusTotal для глибшого аналізу підозрілих файлів, об’єднуючи переваги локальної криптографічної верифікації та зовнішнього антивірусного сканування. Завдяки двоступеневій верифікації, спершу локальній перевірці цілісності, а потім зовнішній валідації через блокчейн і VirusTotal це система забезпечує високу оперативність сповіщень і стійкість до спроб обходу або знищення слідів втручання. Застосоване рішення дозволяє організаціям мінімізувати ризики несанкціонованого поширення ПЗ та захистити авторські права в умовах сучасних загроз.

Отже, всі поставлені завдання для досягнення мети магістерської роботи досягнуто, цілі виконані. Тобто розроблено метод автоматичного моніторингу цілісності файлів для захисту авторського права.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stallings W. Cryptography and Network Security: Principles and Practice. 7 : Pearson, 2017.

2. Diffie W., Hellman M. New directions in cryptography. IEEE Transactions on Information Theory. 1976. Vol. 22, no. 6. P. 644–654. URL: https://doi.org/10.1109/tit.1976.1055638.

3. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System: Independent, 2008.

4. Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM. 1978. Vol. 21, no. 2. P. 120–126. URL: https://doi.org/10.1145/359340.359342.

5. INFORMATION SECURITY CONTROLS. ISO/IEC 27001:2022. 2022. P. 28–36. URL: https://doi.org/10.2307/j.ctv30qq13d.8.

6. Digital Millennium Copyright Act (DMCA), 1998.

7. World Intellectual Property Organization (WIPO). Berne Convention for the Protection of Literary and Artistic Works.

8. General Data Protection Regulation (GDPR), 2016.

9. National Institute of Standards and Technology (NIST). FIPS PUB 180-4: Secure Hash Standard (SHS).

10. Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform: 2014.

11. Swan M. Blockchain: Blueprint for a New Economy. O'Reilly Media, Incorporated, 2015.

12. Radziwill N. Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World. Quality Management Journal. 2018. Vol. 25, no. 1. P. 64–65. URL: https://doi.org/10.1080/10686967.2018.1404373.

13. Bently, L., Johnson, P. The legal protection of software: the case for a specific protection of software by patent law: Queen Mary Journal of Intellectual Property, 5(1), 1-25, 2015.

14. Mohapatra P. K. Public key cryptography. XRDS: Crossroads, The ACM Magazine for Students. 2000. Vol. 7, no. 1. P. 14–22. URL: https://doi.org/10.1145/351092.351098.

15. Dynamic path-based software watermarking / C. Collberg et al. ACM SIGPLAN Notices. 2004. Vol. 39, no. 6. P. 107–118. URL: https://doi.org/10.1145/996893.996856.

16. Steganography / I. J. Cox et al. Digital Watermarking and Steganography. 2008. P. 425–467. URL: https://doi.org/10.1016/b978-012372585-1.50015-2.

17. Kiernan J., Agrawal R., Haas P. J. Watermarking relational data: framework, algorithms and analysis. The VLDB Journal The International Journal on Very Large Data Bases. 2003. Vol. 12, no. 2. P. 157–169. URL: https://doi.org/10.1007/s00778-003-0097-x.

18. APPLIED AND METHODICAL ASPECTS OF USING HASH FUNCTIONS FOR INFORMATION SECURITY / Y. Zhdanova et al. Cybersecurity: Education, Science, Technique. 2020. Vol. 4, no. 8. P. 85–96. URL: https://doi.org/10.28925/2663-4023.2020.8.8596.

19.EITCA Academy. Як хеш-функції використовуються в цифрових підписах і перевірці цілісності даних: EITCA Academy. URL: https://uk.eitca.org/.

20. Проблеми застосування електронного цифрового підпису : thesis / В. І. Лисиця та ін. 2008. URL: http://essuir.sumdu.edu.ua/handle/123456789/18294.

21. Дослідження методів захисту Java-додатків та їх програмна реалізація: Харківський національний університет радіоелектронікиб 2021. URL: https://openarchive.nure.ua/bitstreams/d3a5459a-0c63-4d22-ad8f-158f69acf7b9/download
.

22. Negi A., Goyal A. Optimizing Fully Homomorphic Encryption Algorithm using RSA and Diffie- Hellman Approach in Cloud Computing. International Journal of Computer Sciences and Engineering. 2018. Vol. 6, no. 5. P. 215–220. URL: https://doi.org/10.26438/ijcse/v6i5.215220.

23. Bi J., Yuan H., Zhou M. Temporal Prediction of Multiapplication Consolidated Workloads in Distributed Clouds. IEEE Transactions on Automation Science and Engineering. 2019. Vol. 16, no. 4. P. 1763–1773. URL: https://doi.org/10.1109/tase.2019.2895801.

24. Biobjective Task Scheduling for Distributed Green Data Centers / H. Yuan et al. IEEE Transactions on Automation Science and Engineering. 2020. P. 1–12. URL: https://doi.org/10.1109/tase.2019.2958979.

25. Wilczyński A., Kołodziej J. Modelling and simulation of security-aware task scheduling in cloud computing based on Blockchain technology. Simulation Modelling Practice and Theory. 2020. Vol. 99. P. 102038. URL: https://doi.org/10.1016/j.simpat.2019.102038.

26. Cong L. W., He Z., Zheng J. Blockchain Disruption and Smart Contracts. SSRN Electronic Journal. 2017. URL: https://doi.org/10.2139/ssrn.2985764.

27. Tkinter GUI. Python by Example. 2019. P. 110–123. URL: https://doi.org/10.1017/9781108591942.019 (date of access: 20.05.2025).

28. Kaswan K. S., Dhatterwal J. S., Balamurugan B. GUI Programming Using Tkinter. Python for Beginners. Boca Raton, 2023. P. 313–347. URL: https://doi.org/10.1201/9781003202035-12 (date of access: 20.05.2025).

29. GeeksforGeeks. Python Tkinter Tutorial. URL: https://www.geeksforgeeks.org/python-tkinter-tutorial.

30. GeeksforGeeks. Python GUI – PyQt VS TKinter. URL: https://www.geeksforgeeks.org/python-gui-pyqt-vs-tkinter/.

31. Medium.com. Building Cross-Platform Applications with Tkinter and PyInstaller / Tom Talks Python. URL: https://medium.com/tomtalkspython/building-cross-platform-applications-with-tkinter-and-pyinstaller-d7a10163c550.

32. PyGuis.com. PyQt vs. Tkinter: Which Should You Choose for Your Next Python GUI? URL: https://www.pythonguis.com/faq/pyqt-vs-tkinter/.

33. GeeksforGeeks. Disadvantages of using Tkinter. URL: https://www.geeksforgeeks.org/python-gui-pyqt-vs-tkinter/.

34. Medium. Medium. URL: https://medium.com/@fareedkhandev/themes-for-tkinter-232c17813e3a.

35. Styles and Themes. TkDocs. URL: https://tkdocs.com/tutorial/styles.html.

36. Tkinter Performance Issues - Follow Up. Stack Overflow. URL: https://stackoverflow.com/questions/61938729/tkinter-performance-issues-follow-up.

37. GeeksforGeeks. Difference Between The Widgets Of Tkinter And Tkinter.Ttk In Python - GeeksforGeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/difference-between-the-widgets-of-tkinter-and-tkinter-ttk-in-python/.

38. Beyond Bitcoin: Understanding Legal Boundaries of Blockchain Technology. European Journal of Privacy Law & Technologies. 2022. No. 1. P. 110–131. URL: https://doi.org/10.57230/ejplt221adavo.

39. Where Is Current Research on Blockchain Technology?–A Systematic Review / J. Yli-Huumo et al. PLOS ONE. 2016. Vol. 11, no. 10. P. e0163477. URL: https://doi.org/10.1371/journal.pone.0163477.

40. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends / Z. Zheng et al. 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017. 2017. URL: https://doi.org/10.1109/bigdatacongress.2017.85.

41. Blockchain for IoT security and privacy: The case study of a smart home / A. Dorri et al. 2017 IEEE International Conference on Pervasive Computing and Communications: Workshops (PerCom Workshops), Kona, HI, 13–17 March 2017. 2017. URL: https://doi.org/10.1109/percomw.2017.7917634.

42. Christidis K., Devetsikiotis M. Blockchains and Smart Contracts for the Internet of Things. IEEE Access. 2016. Vol. 4. P. 2292–2303. URL: https://doi.org/10.1109/access.2016.2566339.

43. Atzei N., Bartoletti M., Cimoli T. A Survey of Attacks on Ethereum Smart Contracts (SoK). Lecture Notes in Computer Science. Berlin, Heidelberg, 2017. P. 164–186. URL: https://doi.org/10.1007/978-3-662-54455-6_8.

44. A survey on the security of blockchain systems / X. Li et al. Future Generation Computer Systems. 2020. Vol. 107. P. 841–853. URL: https://doi.org/10.1016/j.future.2017.08.020.

45. Casino F., Dasaklis T. K., Patsakis C. A systematic literature review of blockchain-based applications: Current status, classification and open issues. Telematics and Informatics. 2019. Vol. 36. P. 55–81. URL: https://doi.org/10.1016/j.tele.2018.11.006.

46. Munira M. S. K. SYSTEMATIC REVIEW OF BLOCKCHAIN TECHNOLOGY IN TRADE FINANCE AND BANKING SECURITY. SSRN Electronic Journal. 2025. URL: https://doi.org/10.2139/ssrn.5161366.

47. Introduction to Blockchain and Smart Contract – Principles, Applications, and Security / A. Singh et al. Blockchain Technology in Healthcare Applications. Boca Raton, 2022. P. 175–197. URL: https://doi.org/10.1201/9781003224075-9.

48. A survey on blockchain-based Recommender Systems: Integration architecture and taxonomy / L. Mekouar et al. Computer Communications. 2022. Vol. 187. P. 1–19. URL: https://doi.org/10.1016/j.comcom.2022.01.020.

49. Smart Contract-Based Access Control for the Internet of Things / Y. Zhang et al. IEEE Internet of Things Journal. 2019. Vol. 6, no. 2. P. 1594–1605. URL: https://doi.org/10.1109/jiot.2018.2847705.

50. Zhang R., Xue R., Liu L. Security and Privacy on Blockchain. ACM Computing Surveys. 2019. Vol. 52, no. 3. P. 1–34. URL: https://doi.org/10.1145/3316481.

51. A Survey on Consensus Algorithms used in Blockchain Platforms. International Journal of Advanced Trends in Computer Science and Engineering. 2020. Vol. 9, no. 5. P. 9155–9162. URL: https://doi.org/10.30534/ijatcse/2020/323952020.

52. MedRec: Using Blockchain for Medical Data Access and Permission Management / A. Azaria et al. 2016 2nd International Conference on Open and Big Data (OBD), Vienna, Austria, 22–24 August 2016. 2016. URL: https://doi.org/10.1109/obd.2016.11.

ДОДАТОК А

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ РОБОТИ

Тези наукових доповідей:

1. Гаврищук М., Барановська Л. Метод автоматичного моніторингу цільності файлів для захисту авторського права. VIII Міжнародна науково-практична конференція «Проблеми кібербезпеки інформаційно-комунікаційних систем» (PCSITS), 11 квітня 2025, Київ, Україна, С. 127-128.