Навчально-науковий інститут інформаційних технологій
та інноваційного підприємництва
(повна назва інституту)
Кафедра технічної кібернетики й інформаційних технологій
ім. професора Р.В. Меркта
(повна назва кафедри)

Пояснювальна записка

до кваліфікаційної роботи
бакалавр
(освітньо-кваліфікаційний рівень)
на тему
Захист авторського медіаконтенту
за допомогою асиметричного шифрування

Виконала:
здобувач вищої освіти 4 курсу, групи 2
Спеціальність:
122 «Комп'ютерні науки»
(шифр і назва спеціальності)
(підпис) Ольга РУДЕНКО
(Ім'я, ПРІЗВИЩЕ)
Керівник к. геогр. н., доцент
(вчене звання, посада)
(підпис) Юрій ДАУС
(Ім'я, ПРІЗВИЩЕ)

Одеса – 2026

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ

Інститут Навчально-науковий інститут інформаційних технологій та інноваційного підприємництва

Кафедра технічної кібернетики й інформаційних технологій ім. професора Р.В. Меркта

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки -
(шифр і назва)

Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри КТКйІТ ім. проф. Р.В. Меркта

Павло НОСОВ

«___» _____ 20 26 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Ольга РУДЕНКО
(Ім'я, ПРІЗВИЩЕ)

1. Тема проекту (роботи) Захист авторського медіаконтенту за допомогою асиметричного шифрування

керівник проекту (роботи) Юрій ДАУС к.геогр.н, доцент
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «___»___ 20 26 р. №_____

2. Строк подання проекту (роботи)

3. Вхідні дані до проекту (роботи) Об'єкт захисту: цифрові медіафайли

Алгоритми криптографічного перетворення: симетричний алгоритм AES-256, асиметричний алгоритм RSA-4096

Середовище розробки: фреймворк ASP.NET Core MVC(C#), СУБД SQLite

Технологія візуалізації: HTML5 Canvas

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Теоретично обгрунтувати використання методів гібридного шифрування та криптографічного зношування
2. Розробити математичну модель захисту медіаконтенту із застосуванням блокчейн-реєстру та стеганографії
3. Створити програмну реалізацію вебдодатка "CryptoShield Pro" з механізмами безпечної візуалізації та експертизи.
4. Проаналізувати способи зниження навантаження на серцево-судинну систему та ризики від переробок для ІТ-фахівців.

6. Консультанти розділів проекту (роботи)

Розділ	Ім'я, ПРІЗВИЩЕ та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	доц. Сергій Хотін	07.05.26	11.06.26

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Видача завдання	27.04.2026	
2	Переддипломна практика, залік	27.04.2026 - 16.05.2026	
3	Коригування завдання за результатами практики	17.05.2026	
4	Проміжний звіт на кафедри оцінка готовності	08.06.2026	
5	Попередній захист на кафедрі	15.06.2026	
6	Рецензування	15.06.2026	
7	Захист на засіданні ЕК	25.06.2026	

Здобувач вищої освіти _____ (підпис) Ольга РУДЕНКО (ім'я, ПРІЗВИЩЕ)

Керівник проекту (роботи) _____ (підпис) Юрій ДАУС (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дана кваліфікаційна робота присвячена вирішенню актуальної науково-практичної задачі - розробці та впровадженню інформаційної системи захисту авторського права на цифровий медіаконтент у відкритому вебсередовищі. Захист реалізовано за допомогою гібридного шифрування та технології блокчейн. У роботі детально досліджено сучасні загрози конфіденційності та цілісності графічного контенту, проаналізовано недоліки існуючих централізованих систем управління цифровими правами (DRM).

Науково обґрунтовано впровадження гібридної моделі, яка поєднує високу обчислювальну швидкодію симетричного алгоритму блокового кодування **AES-256** та математичну стійкість асиметричного алгоритму **RSA-4096** (з використанням схем OAEP(Optimal Asymmetric Encryption Padding) та PSS (Probabilistic Signature Scheme)) для інкапсуляції ключів та формування електронних цифрових підписів. Особливу увагу приділено методам просторового хаотичного перемішування пікселів для нівелювання вразливостей, притаманних візуальним даним.

Практична реалізація проєкту представлена у вигляді клієнт-серверного вебдодатка на базі технологічної платформи **ASP.NETCore MVC** (Active Server Pages .NET Core Model-View-Controller.) та оптимізованої архітектури обробки даних (Span/Memory). Реалізовано технології стійкого водяного знака (DWT(Discrete Wavelet Transform) + SVD(Singular Value Decomposition)) та механізми безпечної клієнтської візуалізації (HTML5 Canvas). В систему інтегровано імітаційну модель блокчейн-реєстру з фіксацією транзакцій та часових штампів за алгоритмом **SHA-256**.

Результати роботи підтверджують, що розроблена система надійно захищає медіаконтент та забезпечує ефективне управління правами доступу, можна рекомендувати її для впровадження у сучасні платформи поширення цифрових об'єктів.

ANNOTATION

This qualification work is devoted to solving the actual scientific and practical problem of developing and implementing an information system for protecting copyright on digital media content in an open web environment. The protection is implemented using hybrid encryption and blockchain technology. The paper thoroughly investigates modern threats to the confidentiality and integrity of graphic content and analyzes the shortcomings of existing centralized Digital Rights Management (DRM) systems.

The implementation of a hybrid model is scientifically substantiated, combining the high computational speed of the symmetric block coding algorithm **AES-256** and the mathematical resilience of the asymmetric algorithm **RSA-4096** (using OAEP(Optimal Asymmetric Encryption Padding) and PSS (Probabilistic Signature Scheme)) for key encapsulation and forming electronic digital signatures. Particular attention is paid to methods of spatial chaotic pixel scrambling to neutralize vulnerabilities inherent in visual data.

The practical implementation of the project is presented as a client-server web application based on the **ASP.NET Core MVC** (Active Server Pages .NET Core Model-View-Controller.) technology platform and an optimized data processing architecture (Span/Memory). Technologies for robust watermarking (DWT(Discrete Wavelet Transform) + SVD(Singular Value Decomposition)) and secure client visualization mechanisms (HTML5 Canvas) are implemented. An imitation model of a blockchain register with transaction and timestamp fixation using the **SHA-256** algorithm is integrated into the system.

The results confirm that the developed system reliably protects media content and ensures effective access rights management, making it suitable for implementation in modern digital object distribution platforms.

ЗМІСТ

ВСТУП

Активний розвиток інтернет-технологій зробив цифрові зображення та відео важливими активами для медіабізнесу, електронної комерції та державних комунікаційних систем. Зважаючи на широку доступність такого контенту, питання захисту інтелектуальної власності набуло особливої актуальності. Незаконне копіювання, несанкціоноване поширення та модифікація медіафайлів завдають значних збитків авторам та власникам прав, що вимагає впровадження ефективних технічних засобів захисту.

Використання традиційних методів криптографії у сучасному вебсередовищі має низку обмежень, пов’язаних як з великим обсягом мультимедійних даних, так і з особливостями їхньої структури. Хоча деякі асиметричні алгоритми забезпечують надійний обмін ключами, їхнє пряме застосування для шифрування великих файлів спричиняє надмірне навантаження на обчислювальні ресурси серверів. Тому доцільним є використання гібридної архітектури: поєднання високої продуктивності симетричних шифрів (для захисту контенту) та математичної надійності асиметричних алгоритмів (для безпечного розподілу ключів). Такий підхід забезпечує конфіденційність та цілісність інтелектуальної власності без відчутного зниження швидкодії системи.

Мета роботи полягає у модифікації та поєднанні методів захисту медіаконтенту шляхом розробки спеціалізованого гібридного рішення, що поєднує швидкість симетричних алгоритмів та надійність асиметричних протоколів обміну ключами.

Об’єктом дослідження є процеси технічного захисту авторського права на медіаконтент у цифровому середовищі.

Предметом дослідження є криптографічні методи та алгоритми гібридного шифрування, засоби хешування та механізми візуалізації, що забезпечують підтвердження авторства та цілісність даних.

Для досягнення мети сформовано наступні завдання:

а) проаналізувати вразливості цифрових об'єктів та сучасні загрози їхній цілісності у відкритих мережах;

б) дослідити математичні основи асиметричного шифрування RSA та алгоритмів хешування для обґрунтування параметрів безпеки;

в) спроектувати архітектуру гібридної системи для захисту даних та безпечного розподілу ключів доступу;

г) розробити програмний комплекс із застосуванням технологій блокчейн-реєстру та фронтенд-захисту від несанкціонованого копіювання.

Теоретична значущість результатів полягає в удосконаленні моделі захисту великих масивів даних шляхом інтеграції стеганографічного зв'язування з імітаційним блокчейн-реєстром.

Практичне значення роботи підтверджується створенням вебдодатка "CryptoShield Pro", який автоматизує процеси шифрування, зберігання, дешифрування та верифікації цифрових активів.

1 КРИПТОГРАФІЧНІ ОСНОВИ ТА ЕВОЛЮЦІЯ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

1.1 Еволюція та передумови виникнення асиметричної криптографії

Історичний розвиток методів криптографічного захисту інформації тривалий час ґрунтувався на використанні симетричних алгоритмів. У цій архітектурі як відправник, так і одержувач повідомлення застосовують один спільний секретний ключ для виконання операцій перетворення відкритого тексту на шифротекст та його відновлення. Хоча симетричні алгоритми характеризуються високою обчислювальною ефективністю та здатністю обробляти великі масиви даних у режимі реального часу, вони мають суттєвий недолік, а саме - проблему безпечного розподілу ключів. У закритих інфраструктурах це завдання вирішувалося шляхом фізичної передачі ключів довіреними особами, проте з початком масового розгортання відкритих комп'ютерних мереж використання симетричної моделі стало складнішим, оскільки виникла потреба встановлення захищеного каналу зв'язку між віддаленими вузлами без ризику перехоплення секретного ключа під час його передачі.

Важливим кроком у розвитку інформаційної безпеки стала публікація американських вчених Вітфілда Діффі та Мартіна Геллмана.У 1976 році Вітфілд Діффі та Мартін Геллман запропонували концепцію криптографії з відкритим ключем, яка передбачає використання двох взаємопов'язаних ключів — відкритого та закритого. У тій самій роботі було представлено протокол обміну ключами Діффі—Геллмана, що дає двом сторонам сформувати спільний секретний ключ через відкритий канал зв'язку без попередньої передачі секретної інформації. Запропоновані ними ідеї стали основою розвитку сучасних асиметричних криптографічних систем та засобів захисту інформації.[1]

Вже у 1977 році дослідники Рональд Рівест, Аді Шамір та Леонард Адлеман розробили алгоритм RSA, який став першою практичною реалізацією асиметричної криптосистеми.[2] На відміну від протоколу обміну ключами Діффі-

Геллмана, RSA дозволяв як шифрувати дані, так і створювати електронні цифрові підписи. Математичним підґрунтям RSA стала задача факторизації великих цілих чисел – це операція розкладання великого складеного числа на його прості множники, що є обчислювально складною задачею.

Попри математичну ефективність, застосування алгоритму RSA для безпосереднього захисту мультимедійного контенту (цифрових зображень, графічних об'єктів, відеопотоків) виявилося недоцільним з технічного погляду. Мультимедійні файли характеризуються значними обсягами даних та високою просторовою кореляцією пікселів. Процес асиметричного шифрування потребує виконання операцій модульного піднесення до степеня над числами розміром у тисячі бітів. Робота з такими математичними об'єктами суттєво навантажує центральні процесори серверів і знижує швидкість обробки інформації, крім того, сучасні методи криптоаналізу вимагають використання ключів RSA довжиною 4096 бітів для забезпечення стійкості, що додатково уповільнює роботу системи. Ці технічні обмеження зумовлюють потребу в переході до гібридних криптосистем, де асиметричні алгоритми використовуються переважно для безпечної інкапсуляції сеансових ключів та підтвердження авторства, тоді як потокова обробка великих масивів даних делегується більш швидким симетричним стандартам.[3]

1.2 Основні методи асиметричного шифрування та математичний апарат RSA

Незважаючи на те, що Діффі та Геллман розробили теоретичний фундамент, їхній початковий протокол дозволяв лише безпечно обмінюватися ключами, але не підтримував пряме шифрування довільних масивів даних чи створення цифрових підписів. Практична реалізація повноцінної асиметричної криптосистеми відбулася у 1978 році, коли науковці Рональд Рівест, Аді Шамір та Леонард Адлеман втілили цю концепцію в математичний алгоритм, який отримав назву RSA за першими літерами їхніх прізвищ.[2]

Стійкість алгоритму RSA не базується на складності операцій перестановки чи підстановки бітів, як у симетричних шифрах. Вона ґрунтується виключно на обчислювальній складності розв'язання відомої алгебраїчної задачі факторизації, тобто на складності розкладання надзвичайно великих цілих складених чисел на їхні первинні прості множники. Перемножити два великі прості числа комп'ютер здатен за частки мікросекунди, проте виконання зворотної операції вимагає застосування субекспоненційних алгоритмів, що перетворює розв'язання цієї задачі на процес, який може тривати тисячоліттями за належно обраних параметрів ключів. На сьогодні для забезпечення довгострокової надійності використовуються ключі довжиною 4096 бітів (стандарт RSA-4096). Алгоритм генерації ключів RSA складається з наступних етапів:

На першому етапі за допомогою криптографічно стійкого генератора псевдовипадкових чисел обираються два великі прості числа p та q [2]. Для досягнення рівня безпеки RSA-4096 довжина кожного з цих чисел у двійковому поданні становить рівно 2048 бітів. Використання апаратних джерел ентропії на цьому етапі є критичним, оскільки компрометація генератора псевдовипадкових чисел дозволить зловмиснику передбачити обрані прості числа.

Далі обчислюється загальний модуль системи N, який утворюється шляхом алгебраїчного перемноження двох знайдених простих чисел:

$$N = p \cdot q \tag{1.1}$$

де p є першим згенерованим простим числом; q є другим згенерованим простим числом.

Цей модуль N стає центральним елементом як відкритого, так і закритого ключів. Його довжина визначає розрядність криптосистеми.

Наступним кроком є обчислення значення функції Ейлера. У теорії чисел ця функція визначає точну кількість цілих додатних чисел, які менші за N і є взаємно простими з ним. Оскільки числа p та q є простими, функція Ейлера для їхнього добутку розраховується за спрощеною формулою:

$$\varphi(N) = (p-1) \cdot (q-1) \tag{1.2}$$

Після визначення функції Ейлера система обирає відкриту експоненту e. Математичною вимогою до цього параметра є те, що число e має бути взаємно простим зі значенням функції Ейлера. Стандартно в сучасній інженерній практиці застосовується просте число Ферма:

$$e = 65537 \tag{1.3}$$

Вибір саме цього числа зумовлений тим, що його двійкове подання містить лише дві одиниці, що суттєво прискорює операцію піднесення до степеня під час шифрування даних. На завершальному етапі генерується закритий ключ. За допомогою розширеного алгоритму Евкліда обчислюється закрита експонента d, яка виступає мультиплікативним оберненим числом до e за модулем функції Ейлера. Математично це виражається у вигляді лінійного діофантового рівняння:

$$e \cdot d \equiv 1 \left(mod\ \varphi(N)\right) \tag{1.4}$$

У результаті описаних операцій формуються дві структури. Публічним ключем стає пара чисел e та N, а приватним ключем виступає пара d та N. Процес шифрування відкритого повідомлення M, яке попередньо конвертується у велике ціле число, здійснюється за допомогою операції модульного піднесення до степеня:

$$C \equiv M^e (mod\ N) \tag{1.5}$$

Дешифрування отриманого шифротексту C виконується легітимним одержувачем шляхом застосування секретної експоненти d:

$$M \equiv C^d (mod\ N) \tag{1.6}$$

У сучасних системах інформаційної безпеки використання чистого базового алгоритму RSA категорично забороняється. Базовий алгоритм є жорстко детермінованим, що означає утворення абсолютно ідентичного шифротексту при багаторазовому шифруванні одного й того ж повідомлення. Для нівелювання цих математичних загроз застосовуються спеціальні схеми криптографічного доповнення. У розробленому проєкті імплементовано стандарт форматування RSA-OAEP. Алгоритм OAEP(Optimal Asymmetric Encryption Padding) доповнює повідомлення випадковим шумом та обробляє його за допомогою маскувальних

функцій на базі криптографічного хешування. Це робить процес шифрування ймовірнісним та блокує можливості частотного криптоаналізу.[4]

1.3 Алгоритми криптографічного хешування (SHA-256) та стандарти ДСТУ

Для контролю цілісності даних, формування цифрових доказів авторства та побудови розподілених блокчейн-реєстрів у межах даної роботи застосовано алгоритми криптографічного хешування. Хеш-функція виконує перетворення вхідного масиву даних довільного розміру на унікальний двійковий рядок (хеш-дайджест) фіксованої довжини, це гарантує надійну ідентифікацію кожного цифрового об’єкта та доводить його незмінність протягом усього циклу зберігання.

Основним інструментом для верифікації медіаданих у розробленій системі обрано алгоритм SHA-256 (Secure Hash Algorithm 256-bit). [5]

Вибір цього стандарту зумовлений його широкою підтримкою у криптографічних бібліотеках, високою обчислювальною продуктивністю та відповідністю сучасним вимогам до безпеки даних. Алгоритм базується на ітеративній обробці вхідного потоку даних блоками фіксованого розміру (512 біт), де шляхом послідовних логічних операцій (побітові зсуви, додавання, функції вибору) формується 256-бітний дайджест.

Криптографічна стійкість SHA-256 базується на виконанні трьох основних вимог:

1. Незворотність (Pre-image resistance): за відомим значенням хешу обчислювально неможливо відновити початковий вхідний файл, це унеможливлює спроби реконструювання медіаконтенту на основі його ідентифікатора.

2. Стійкість до знаходження другого прообразу (Second pre-image resistance): неможливо підібрати інший файл, який би мав аналогічне значення хешу, це гарантує, що жоден модифікований медіаоб’єкт не зможе видати себе за оригінальний файл.

3. Стійкість до колізій (Collision resistance): ймовірність утворення однакових хеш-дайджестів для двох різних файлів малоймовірною, що забезпечує унікальність кожного об’єкта у блокчейн-реєстрі.

Окремої уваги заслуговує «лавинний ефект», притаманний SHA-256: будь-яка мінімальна зміна у вхідному файлі - від редагування одного байта метаданих до зміни значення одного пікселя призводить до повної, непередбачуваної зміни хеш-відбиткая, ця властивість дає системі можливість миттєво фіксувати факт несанкціонованого втручання у структуру файлу під час проведення аудиту цілісності.

На рівні національних стандартів України функцію хешування виконує стандарт ДСТУ 7564:2014 («Купина»)[6], даний стандарт дає високий рівень стійкості до сучасних методів лінійного та диференціального криптоаналізу. Використання стандартизованого хешування є обов'язковою умовою для юридичного підтвердження автентичності та цілісності даних у вітчизняних системах електронного документообігу. Не дивлячись на те, що в роботі інтегровано SHA-256 через його широку сумісність із блокчейн-технологіями, стандарт «Купина» є повноцінною альтернативою, яка відповідає вимогам вітчизняного законодавства щодо криптографічного захисту інформації.

1.4 Принципи побудови гібридних криптосистем

Алгоритм RSA забезпечує високий рівень криптографічної стійкості, проте його застосування для безпосереднього захисту мультимедійного контенту є технічно недоцільним через високу обчислювальну складність. Операції модульного піднесення до степеня над числами розміром у 4096 бітів при обробці файлів обсягом у десятки чи сотні мегабайтів призводять до надмірного навантаження на центральні процесори та суттєвого зниження швидкодії системи. Для забезпечення захисту даних без погіршення продуктивності інформаційного середовища в криптографії застосовується архітектура гібридних систем.[3]

Гібридний підхід базується на поєднанні переваг симетричного та асиметричного шифрування. Така архітектура реалізує модель «цифрового конверта», яка розподіляє функції захисту між двома типами алгоритмів. Потокова обробка та шифрування основного масиву даних делегується симетричним алгоритмам, тоді як асиметричні алгоритми використовуються для безпечної передачі (інкапсуляції) сеансових ключів доступу.

На першому етапі роботи системи генерується одноразовий симетричний ключ для алгоритму AES-256. Даний алгоритм є блочним шифром, який використовує складну послідовність побітових перетворень (підстановок та перестановок) у межах скінченних полів Galois [23]. Завдяки реалізації апаратної підтримки (інструкції AES-NI) у сучасних процесорах, симетричне шифрування прискорює швидкість обробки даних, що робить можливим шифрування гігабайтних обсягів інформації в режимі, близькому до реального часу. Оригінальний масив пікселів зображення повністю перетворюється на шифротекст із використанням згенерованого сеансового ключа у режимі зчеплення блоків (CBC). Оскільки безпечне розповсюдження симетричних ключів через відкриті мережі є неможливим, реалізується механізм їх захищеної передачі за допомогою асиметричної криптографії. Згенерований ключ AES-256 зашифровується публічним ключем RSA, що належить одержувачу. Отриманий зашифрований масив даних зберігається як окремий криптографічний конверт.

Результатом застосування гібридної архітектури є формування двох взаємопов'язаних об'єктів: зашифрованого контейнера з медіаданими та криптографічного конверта з ключем доступу. Дешифрування інформації можливе лише за наявності закритого приватного ключа RSA, який математично пов'язаний з використаним публічним ключем. Такий розподіл обов'язків реалізує поєднання двох ключових переваг: швидкодію симетричних алгоритмів при роботі з великими обсягами даних та високий рівень безпеки асиметричних систем при управлінні ключами доступу.

1.5 Аналіз існуючих систем управління цифровими правами (DRM) та їхні архітектурні недоліки

Перехід медіа індустрії до цифрових форматів розповсюдження контенту зумовив необхідність створення спеціалізованих програмно-апаратних комплексів для захисту інтелектуальної власності. У сучасному інформаційному середовищі цю функцію виконують системи управління цифровими правами, які в науковій літературі позначаються абревіатурою DRM (Digital Rights Management) . Головне завдання таких комплексів полягає у технічному обмеженні дій кінцевого користувача виключно тими повноваженнями, які були попередньо делеговані правовласником, система DRM повинна гарантувати неможливість копіювання, модифікації або несанкціонованої передачі придбаного цифрового об'єкта третім особам без проходження відповідної авторизації [7].

Формування ринку систем управління цифровими правами (DRM) супроводжувалося стійкою тенденцією до централізації через те, що більшість великих компаній розробили власні закриті платформи, робота яких жорстко залежить від виділеного сервера ліцензій. Стандартна процедура передбачає, що після завантаження зашифрованого медіаконтейнера клієнтський застосунок має звернутися до центрального вузла для ідентифікації та отримання ключа. Попри зручність адміністрування для правовласників, така архітектура містить критичну вразливість, а саме - єдину точку відмови (Single Point of Failure), тобто у разі апаратного збою або цілеспрямованої мережевої атаки (наприклад, DDoS) користувачі залишаються без доступу до придбаного контенту, оскільки локальне дешифрування без зв'язку із сервером стає неможливим. Ще одним недоліком традиційних систем DRM виступає їхня тотальна залежність від постійного підключення до глобальної мережі, для верифікації прав доступу програма повинна регулярно синхронізуватися з віддаленим сервером, це робить неможливим повноцінне використання медіаконтенту в умовах автономної роботи або за відсутності стабільного інтернет з'єднання. Крім того, подібна архітектура породжує серйозні проблеми у сфері інформаційної приватності бо сервери

авторизації безперервно збирають та акумулюють деталізовані телеметричні дані про поведінку користувачів, фіксуючи точний час, геолокацію та тривалість перегляду кожного захищеного файлу. З точки зору сучасних стандартів захисту персональних даних такий підхід розцінюється як надмірне втручання у приватний простір особи. Класичні системи DRM мають суттєві обмеження щодо захисту контенту на етапі його візуалізації,не дивлячись на те, що більшість алгоритмів забезпечують безпеку даних під час їхнього транспортування мережею та ізолюють файли на накопичувачах, у момент розшифрування та передачі інформації у відеопам'ять для виведення на екран рівень захисту знижується, через це зловмисники можуть застосовувати програмне забезпечення для захоплення екрана або апаратні відеограбери для створення високоякісних копій в обхід наявних ліцензійних механізмів. З огляду на такі архітектурні недоліки, класична централізована модель потребує оновлення. Вирішення проблеми лежить у площині децентралізованих рішень, де доказова база авторства інтегрується безпосередньо у структуру медіафайлу. У такій системі контроль доступу доцільно реалізовувати через гібридне шифрування, а верифікацію — за допомогою вузлів блокчейн-реєстру. Запропонована архітектура нівелює вразливість єдиної точки відмови та дає правовласникам зберігати технічний контроль над власними матеріалами.

1.6 Вектори атак на сучасні криптосистеми та обґрунтування вибору довжини ключів

Проектування ефективної архітектури для захисту авторського медіаконтенту потребує всебічного врахування потенційних векторів атак та ретельного вибору криптографічних параметрів. Алгоритми та розміри ключів, які забезпечували достатній рівень конфіденційності ще десятиліття тому, сьогодні вважаються надзвичайно вразливими, це зумовлено стрімким розвитком обчислювальної техніки, появою спеціалізованих апаратних прискорювачів та масовим поширенням технологій розподілених хмарних обчислень.

Симетричні алгоритми шифрування, які застосовуються для безпосереднього кодування масивів пікселів, найчастіше піддаються атакам методом повного або тотального перебору. Ефективність цього методу, відомого як атака грубої сили, залежить виключно від довжини секретного сесійного ключа. Кожен додатковий біт у ключі збільшує обсяг простору можливих комбінацій у два рази, експоненційно збільшуючи час для алгоритмічного зламу. У минулому використання ключів довжиною 128 бітів вважалося індустріальним стандартом. Проте сучасні кластери, побудовані на базі високопродуктивних графічних процесорів, здатні генерувати мільярди варіантів ключів за секунду. Для забезпечення беззаперечного запасу криптографічної міцності на найближчі десятиліття розроблена система використовує алгоритм AES із максимально можливою довжиною ключа у 256 бітів. Простір комбінацій для такого ключа унеможливлює проведення успішної атаки повним перебором навіть за умови об'єднання всіх існуючих обчислювальних потужностей планети.

Асиметричні криптосистеми мають абсолютно іншу математичну природу, що зумовлює відмінність векторів атак на них. Стійкість алгоритму RSA, який використовується у системі для інкапсуляції симетричних ключів та підтвердження авторства, базується на складності розв'язання задачі факторизації великих складених чисел. Зловмисник із доступом до публічного ключа намагається розкласти базовий криптографічний модуль на два початкові прості множники. Найшвидшим відомим методом для факторизації модулів RSA є алгоритм загального решета числового поля, головна небезпека цього методу полягає у його здатності до розпаралелювання, тобто етап збору алгебраїчних відношень легко розподіляється між сотнями тисяч незалежних комп'ютерів через глобальну мережу.

Успішні експерименти криптоаналітиків із розкладання чисел довжиною 768 бітів остаточно довели неспроможність стандартних ключів розміром 1024 біти гарантувати надійний захист.[7] Провідні міжнародні агенції з кібербезпеки вже зараз наполягають на відмові від використання модулів довжиною 2048 бітів для обробки інформації, яка потребує тривалого збереження конфіденційності.

Додатковою глобальною загрозою виступає інтенсивна розробка квантових комп'ютерів. Квантові обчислювальні пристрої здатні застосувати алгоритм Шора, який розв'язує задачу факторизації за поліноміальний час.[8] Це може призвести до миттєвого зламу класичних систем із недостатньою довжиною ключів.

З огляду на ці тенденції, архітектура розробленого програмного комплексу передбачає обов'язкове використання ключів RSA із довжиною модуля у 4096 бітів. Збільшення розрядності зумовлює відчутне зростання обчислювального навантаження під час виконання операцій модульного піднесення до ступеня, однак у застосованій гібридній моделі цей недолік повністю нівелюється. Математичний апарат факторизації з ключем 4096 бітів застосовується лише один раз для шифрування короткого сесійного ключа AES, а не для прямої обробки багато мегабайтного медіафайлу. Таке інженерне рішення мінімізує навантаження на обчислювальні ресурси сервера, забезпечуючи при цьому стійкість системи до відомих методів криптоаналізу та перспективних квантових обчислень.

2 МЕТОДИ ЗАХИСТУ ЦИФРОВИХ ОБ'ЄКТІВ ТА БЛОКЧЕЙН-ТЕХНОЛОГІЇ

2.1 Види цифрових об'єктів та необхідність їхнього захисту

Цифрові медіаоб'єкти у сучасному інформаційному середовищі часто сприймаються кінцевим користувачем як прості статичні файли, що зберігаються на фізичних носіях пам'яті або у хмарних сховищах. Проте їхня внутрішня архітектурна та технічна природа є значно складнішою. З технічної точки зору будь-яке цифрове зображення чи відеопотік являє собою колосальну числову матрицю, яка утворюється у момент апаратного перетворення безперервного оптичного сигналу на дискретну сітку з окремих пікселів.

На апаратному рівні цифровий актив завжди складається з двох жорстко пов'язаних сегментів.

Перший сегмент містить безпосередньо закодовані візуальні або аудіальні дані, що є корисним навантаженням.

Другий сегмент є службовим блоком метаданих, який виконує функцію цифрового паспорта об'єкта, зберігаючи лише важливі параметри форматування, інформацію про пристрій створення, геолокацію та ідентифікаційні дані правовласника. Архітектура та методи алгоритмічного стиснення дозволяють виокремити кілька базових класів цифрових активів, кожен з яких має унікальні технічні вразливості:

– Растрові зображення (JPEG, PNG). Технічно растрове зображення являє собою двовимірну матрицю пікселів. Для зменшення кінцевого обсягу файлу алгоритми стиснення з втратами видаляють ту частину надлишкової візуальної інформації, яка найменш помітна для людського фізіологічного сприйняття. Критична вразливість растрових форматів лежить у структурі службового сегмента, бо для несанкційного привласнення контенту порушнику достатньо виконати автоматизовану процедуру видалення метаданих, що назавжди знищує вбудовану ідентифікаційну інформацію про справжнього автора.

– Векторна графіка (SVG). Такі формати побудовані на основі математичних рівнянь, а не фіксованої сітки пікселів. Файл містить структурований набір інструкцій, просторових координат та геометричних примітивів у вигляді структурованого текстового коду з розміткою XML. Оскільки код файлу є відкритим для текстового редагування, зловмисники можуть легко змінити числові коефіцієнти в коді, щоб непомітно зруйнувати вбудовані водяні знаки правовласника.
– Динамічні потоки та відеодані. Відеофайл являє собою строгу послідовність часово синхронізованих кадрів, доповнену аудіо-доріжками. Критичною загрозою для відеоданих є процес транскодування, бо будь-яке переформатування файлу, зміна роздільної здатності або бітрейту вимагає повного математичного перерахунку кожного пікселя, що безповоротно руйнує класичні приховані цифрові водяні знаки.

З огляду на ці архітектурні недоліки, для надійного збереження авторських прав необхідно здійснювати жорстке шифрування безпосередньо корисного навантаження файлу, унеможливлюючи його несанкціоновану обробку до моменту легітимного дешифрування.

2.2 Технологія блокчейн

Незважаючи на високу ефективність симетричних та асиметричних криптографічних протоколів у забезпеченні конфіденційності медіафайлу, класичні архітектури мають критичний концептуальний недолік у сфері управління дозволами. Традиційні централізовані моделі управління цифровими правами тотально залежать від працездатності та чесності єдиного центрального сервера компанії-розробника. У випадку виходу сервера з ладу через кібератаку або технічний збій, усі законно придбані ключі доступу стають недійсними. Математично обґрунтованим вирішенням цієї архітектурної проблеми виступає перехід до децентралізованих мереж на основі технології блокчейну.

Концептуальні наукові засади криптографічно пов'язаних ланцюгів блоків були закладені у 1991 році американськими криптографами Стюартом Хабером та Скоттом Сторнеттою.[9] Їхньою первинною метою було створення криптографічно надійної системи маркування часу для цифрових документів. Дослідники прагнули розробити математичний механізм, який унеможливив би зміну дати створення файлу або підробку послідовності подій. У 1992 році систему було вдосконалено інтеграцією дерев Меркла, що дозволило збирати велику кількість цифрових документів в один криптографічний блок.[10]

Блокчейн являє собою безперервну послідовну базу даних, що підтримується та верифікується одночасно тисячами незалежних вузлів по всьому світу. Інформація в такій мережі жорстко групується у структуровані блоки. Кожен новий блок має заголовок, що містить записи про нові підтверджені транзакції, точну мітку часу та криптографічний хеш безпосередньо попереднього блоку. Третій пункт виступає математичним гарантом незмінності. Як тільки блок створюється і приймається більшістю вузлів мережі, змінити хоча б один біт всередині нього стає алгоритмічно неможливо. Завдяки лавинному ефекту криптографічної хеш-функції SHA-256, найменше редагування старого запису призведе до радикальної зміни його результуючого хешу. Децентралізована система миттєво виявить розбіжність і відхилить таку спробу підробки як невалідне відгалуження.

Застосування блокчейну повністю змінює філософію захисту авторських прав. Архітектура реалізує механізм доведення авторства, за якого система обчислює відбиток SHA-256 оригінального медіафайлу та відправляє його як транзакцію у блокчейн. Зберігати великі графічні або відеофайли безпосередньо у блокчейн-мережі технічно складно через жорсткі вимоги до пам'яті, тому застосовується гібридний підхід розділеного зберігання. Фізичний зашифрований файл зберігається локально на диску користувача, а в блокчейн записується виключно криптографічний хеш цього файлу та метадані транзакції завдяки чому й досягається баланс між продуктивністю системи та надійністю аудиту.

2.3 Математична модель гібридного шифрування та стеганографічного зв'язування даних

Практична реалізація розробленої архітектури вимагає побудови чіткої математичної моделі, здатної описати трансформацію оригінального відкритого медіафайлу у захищений криптографічний об'єкт, стійкий до сучасних векторів атак. [3]

Нехай змінна P позначає відкритий текст у вигляді двійкового масиву пікселів оригінального зображення, а змінна $C_{payload}$ позначає згенерований симетричний шифротекст.

Математичний апарат гібридного кодування ініціюється генерацією сесійних параметрів. Криптопровайдер генерує криптографічно стійкий симетричний сесійний ключ K_{aes} та вектор ініціалізації IV. Вектор ініціалізації є необхідним параметром для забезпечення семантичної безпеки. Формалізовано генерацію можна подати у вигляді виразу:

$$K_{\{aes\}} \in \{0,1\}^{\{256\}}, \backslash quad\ IV\ \in \{0,1\}^{\{128\}} \tag{2.1}$$

де K_{aes} є псевдовипадковим ключем довжиною 256 біт; IV є вектором ініціалізації довжиною 128 біт.

Перед початком симетричного шифрування система зчитує з бази даних хеш попереднього блоку в ланцюгу H_{prev}. Проводиться операція бінарної конкатенації, за якої хеш інтегрується у кінець файлу. Це утворює модифікований масив $P_{\{embedded\}}$ що фізично прив'язує контент до історії блокчейну:

$$P_{embedded} = P \parallel \left[H_{prev}\right] \tag{2.2}$$

де $P_{\{embedded\}}$ є модифікованим масивом даних; P є оригінальним двійковим потоком медіафайлу; $\backslash parallel$ виступає оператором двійкової конкатенації; H_{prev} є криптографічним відбитком попереднього блоку.

Модифікований масив даних зображення шифрується алгоритмом AES у режимі зчеплення блоків. Режим використовує попередній зашифрований блок для маскування наступного відкритого блоку за допомогою операції виключного АБО.

Це повністю руйнує візуальні патерни у матриці пікселів. Рівняння шифрування для ітеративного блоку набуває вигляду:

$$C_i = E_{K_{aes}}(P_i \oplus C_{i-1}) \quad (2.3)$$

де C_i є поточним зашифрованим блоком; E виступає функцією симетричного шифрування алгоритму AES; P_i є відкритим блоком даних зображення; $\backslash oplus$ побітовою логічною операцією виключного АБО; C_{i-1} є попереднім зашифрованим блоком даних.

Для безпечного транспортування симетричний ключ шифрується публічною експонентою сервера. Операція асиметричного захисту ключа описується модульним рівнянням:

$$C_{key} = (K_{aes})^e \backslash pmodn \quad (2.4)$$

де C_{key} є зашифрованим криптографічним конвертом; e виступає відкритою експонентою асиметричного ключа RSA; n є криптографічним модулем системи RSA.

Результатом процесу є формування масивного контейнера з контентом та криптографічного конверта. Спроба зламати конверт вимагатиме факторизації 4096-бітного модуля RSA, що наразі вважається обчислювально нездійсненним завданням у сучасних умовах.

2.4 Захист від несанкційного копіювання: HTML5 Canvas, водяні знаки та блокування інтерфейсу

Окрім надійного криптографічного захисту медіафайлу під час його передачі та зберігання на сервері, повноцінна система управління правами потребує ефективних засобів запобігання несанкційованому копіюванню контенту безпосередньо під час його легітимного відображення у браузері користувача. Для розв'язання цього завдання у розробленому додатку інтегровано комплекс фронтенд-технологій HTML5 Canvas Rendering.[11]

У традиційних вебзастосунках зображення вбудовуються у структуру сторінки за допомогою стандартного тегу зображення, тому будь-який користувач може безперешкодно зберегти файл через контекстне меню браузера. Для усунення цієї загрози розроблена система обробляє бінарні дані прев'ю-файлу JavaScript-скриптом на стороні клієнта і динамічно відмальовує їх на віртуальному полотні. Оскільки елемент інтерпретується рушієм браузера як програмно згенерована растрова область, стандартні інструменти збереження контенту для нього апаратно відсутні.

Для посилення превентивного захисту імплементовано додаткові фронтенд-бар'єри. Поверх відмальованого полотна з фотографією накладається абсолютно прозорий блок з вищим індексом перекриття. При спробі зловмисника скопіювати елемент через інспектор коду браузер виділяє і зберігає лише цей порожній прозорий шар, захищаючи оригінальний графічний контекст. Додатково за допомогою інтеграції спеціальних інструкцій жорстко заблоковано виклик контекстного меню на всій робочій області галереї та забороняє перетягування зображення на робочий стіл.

У процесі рендерингу зображення на полотні скрипт програмно наносить напівпрозорий діагональний напис для захисту авторського права, одночасно алгоритм автоматично виводить точне значення поточного криптографічного хешу та хешу попереднього блоку бази даних, це візуально підтверджує структуру блокчейн-ланцюга і також знижує комерційну цінність контенту у разі спроби створення піратського знімка екрана.

2.5 Теоретичне обґрунтування вибору методу стеганографічної ін'єкції у маркер кінця файлу порівняно з класичними алгоритмами

Традиційні підходи до прихованого маркування мультимедійних файлів здебільшого спираються на класичні стеганографічні алгоритми. Серед таких методів найвідомішим є просторова заміна найменш значущих бітів (LSB) та частотні перетворення у просторі дискретного косинусного перетворення.

Зазначені алгоритми модифікують безпосередньо матрицю пікселів зображення або аудіопотік. Таке алгоритмічне втручання неминуче призводить до незворотної деградації вихідної якості цифрового об'єкта.[12] Хоча візуальні зміни часто залишаються непомітними для людського фізіологічного сприйняття, будь-яке повторне стиснення файлу або його конвертація в інший формат гарантовано руйнує прихований криптографічний маркер.

Для усунення проблеми деградації контенту в архітектурі розробленої системи інтегровано метод стеганографічної ін'єкції в маркер кінця файлу (End of File, EOF). Цей інженерний підхід базується на специфіці зчитування бінарних даних операційними системами та стандартними програмними плеєрами. Кожен графічний або відеофайл має жорстко визначену внутрішню структуру із заголовком та спеціальним байтовим маркером завершення корисного навантаження. Більшість програм для перегляду медіа ігнорують будь-яку додаткову інформацію, записану після цього фінального маркера.

Відповідно до цієї технічної особливості програмний комплекс бере згенерований хеш-відбиток блокчейн-транзакції та алгоритмічно приєднує його в кінець бінарного масиву вже сформованого файлу. Цей механізм гарантує збереження абсолютної побітової ідентичності оригінальної матриці пікселів. Окрім гарантії збереження початкової якості графічного контенту, метод ін'єкції в маркер кінця файлу демонструє виняткову стійкість проти атак, спрямованих на видалення метаданих. Якщо зловмисник використає спеціалізовані утиліти для автоматичного очищення EXIF-тегів, прихований блокчейн-хеш залишиться неушкодженим. Подібна стійкість пояснюється тим, що доданий хеш фізично не належить до стандартного ідентифікаційного блоку метаданих зображення, а виступає додатковим бінарним причепом, який ігнорується стандартними очищувачами контенту.

2.6 Архітектурна модель нульового розголошення (Zero-Knowledge)

Забезпечення абсолютної конфіденційності в сучасних клієнт-серверних екосистемах вимагає впровадження нульового розголошення. Ця концепція інформаційної безпеки гарантує неможливість доступу адміністраторів сервера або сторонніх осіб до приватних файлів користувача навіть у випадку повного фізичного захоплення апаратного забезпечення центру обробки даних. Класичні хмарні платформи зберігають ключі дешифрування у власних реляційних базах даних, що створює критичну структурну вразливість перед внутрішніми загрозами та хакерськими атаками.

У розробленій системі управління правами модель нульового розголошення реалізується через суворий алгоритмічний контроль життєвого циклу криптографічного матеріалу в оперативній пам'яті. Сервер виступає виключно у ролі транзитного середовища та надійного сховища повністю зашифрованих контейнерів. Під час завантаження користувачем медіафайлу система генерує симетричний сесійний ключ AES та пару асиметричних ключів RSA локально в оперативній пам'яті під час виконання єдиного ізольованого HTTP-запиту. Після шифрування контенту симетричним ключем і подальшого захисту самого симетричного ключа відкритою експонентою RSA система формує фінальний криптографічний конверт.

Критичним етапом цієї безпекової архітектури є процес експорту ключів. Згенерований приватний ключ RSA та зашифрований симетричний ключ негайно передаються на пристрій легітимного власника через захищене мережеве з'єднання у вигляді стисненого архіву. Одразу після підтвердження успішної передачі серверний контролер ініціює безумовне очищення відповідних сегментів оперативної пам'яті за допомогою механізмів примусового збирання сміття середовища розробки. У результаті на жорстких дисках сервера залишаються лише математично незламні шифротексти без жодних інструментів для їхнього зворотного перетворення. Для відновлення доступу до власного цифрового архіву користувач зобов'язаний самостійно надати свій збережений приватний ключ. Цей

криптографічний маркер обробляється виключно в межах його поточної активної сесії та ніколи не зберігається на сервері у постійному вигляді, що повністю задовольняє найвищі міжнародні вимоги архітектури нульового розголошення.[13]

3 РЕАЛІЗАЦІЯ ВЕБДОДАТКУ "CRYPTOSHIELD PRO"

3.1 Архітектура серверного вебдодатка

Практична реалізація гібридної системи захисту медіаконтенту потребує надійного інженерного фундаменту, тому в якості базової технологічної платформи для серверної частини було обрано кросплатформний фреймворк ASP.NET Core версії 8.0.[14]. Це середовище функціонує на основі об'єктно-орієнтованої мови програмування C# та використовує кероване середовище виконання. Вибір саме цієї платформи обґрунтований наявністю оптимізованого механізму керування оперативною пам'яттю, під час шифрування масивних мультимедійних файлів вкрай важливо ефективно контролювати виділення ресурсів і в даному випадку платформа автоматично розділяє об'єкти на малі та великі, двійкові потоки графічних зображень потрапляють до спеціальної купи великих об'єктів, а алгоритм збирання сміття ізолює ці ділянки пам'яті та примусово очищує їх після завершення криптографічних операцій, така архітектура захищає від витоку конфіденційної інформації та запобігає переповненню серверних ресурсів під час пікових навантажень.

Ефективність роботи з оперативною пам'яттю у C# додатково підсилюється підтримкою структур Span<T> та Memory<T>[16]. Ці типи даних дозволяють виконувати низькорівневі операції з масивами байтів — наприклад, потокове хешування та блокове перетворення пікселів зображення — без додаткового копіювання даних у системі (парадигма zero-allocation). Робота безпосередньо з безперервними фрагментами пам'яті знижує загальне навантаження на сервер та збільшує пропускну здатність криптографічного ядра при обробці багатомегабайтних файлів.

Окрім оптимізації пам'яті, мова C# надає вбудовану асинхронну модель програмування на базі операторів async та await[17]. Генерація ключів RSA-4096 та шифрування AES-256 вимагають значних процесорних потужностей і належать до обчислювально складних задач, тому важлива асинхронна архітектура завдяки якій

сервер не блокує головні робочі потоки під час виконання цих операцій, натомість система динамічно перемикає звільнені ресурси на обробку інших мережевих запитів, що забезпечує стабільну роботу вебдодатка навіть при одночасному зверненні багатьох користувачів.

Важливим аспектом безпеки є строга типізація мови C#, яка ще на етапі компіляції виключає виникнення поширених програмних вразливостей, таких як переповнення буфера (buffer overflow). Водночас використання вбудованого простору імен System.Security.Cryptography дає прямий керований доступ до апаратних прискорювачів процесора, зокрема криптографічних інструкцій AES-NI. Завдяки цьому алгоритми шифрування виконуються на рівні операційної системи з максимальною швидкістю, усуваючи потребу в залученні сторонніх бібліотек, які можуть містити приховані вразливості або недекларовані можливості.

Мережева взаємодія забезпечується вбудованим вебсервером Kestrel. Цей сервер здатний асинхронно обробляти тисячі одночасних з'єднань. Читання файлів відбувається потоковим методом без повного завантаження масивів пікселів у пам'ять, що гарантує високу пропускну здатність системи.

Програмний комплекс спроєктовано за класичним архітектурним патерном модель-представлення-контролер. Такий підхід реалізує багаторівневий розподіл абстракцій і повністю усуває проблему монолітності вихідного коду [15,16]. Архітектура програми логічно розділяється на незалежні компоненти, кожен з яких виконує суворо відведену йому роль в обробці інформації. Компонент моделей інкапсулює логічні сутності предметної області. У розробленому коді моделі представлені об'єктами класів, які описують структуру користувачів, медіафайлів та блоків реєстру, а вбудовані механізми валідації автоматично перевіряють коректність вхідних даних ще до моменту їхнього потрапляння у ядро безпеки, блокуючи потенційно небезпечні запити.

Шар контролерів виконує роль диспетчера мережевих процесів. Контролери відповідають виключно за маршрутизацію запитів за протоколом HTTP. Головний контролер каталогу приймає бінарні потоки від клієнта через інтерфейс форм, аналізує параметри поточної сесії, перевіряє права доступу та викликає відповідні

сервіси обробки. Головною інженерною вимогою до контролерів є відсутність у них складної математичної логіки. Контролер лише делегує виконання криптографічних операцій спеціалізованим класам безпеки. Такий поділ відповідальності значно спрощує подальше масштабування вебдодатка.

Представлення генерують фінальний графічний інтерфейс для кінцевого користувача. Цей шар побудовано за допомогою розмітки HTML5, каскадних таблиць стилів та серверного синтаксису Razor. Представлення отримують від контролерів уже оброблені дані та динамічно формують безпечне віртуальне полотно для відображення захищеного контенту безпосередньо у браузері.

Ключовим елементом серверної архітектури виступає механізм інверсії управління через впровадження залежностей. У головному конфігураційному файлі налаштовується загальний конвеєр проміжного програмного забезпечення. Усі криптографічні та блокчейн-сервіси реєструються в ізольованому контейнері.

```
// Налаштування підключення до бази даних SQLite через Entity Framework Core
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite("Data Source=crypto.db"));
// Реєстрація користувацьких сервісів бізнес-логіки у контейнері залежностей
builder.Services.AddScoped<SecurityService>();
builder.Services.AddScoped<BlockchainService>();
```

Наведений фрагмент коду демонструє реєстрацію залежностей із життєвим циклом у межах одного мережевого запиту. Платформа автоматично створює нові екземпляри криптографічних сервісів під час надходження файлу від користувача. Після успішного шифрування та відправки архіву середовище виконання так само автоматично знищує ці екземпляри. Подібне інженерне рішення підтримує слабку зв'язність класів і повністю запобігає випадковому збереженню приватних ключів чи розшифрованих масивів пікселів у глобальній пам'яті сервера.

Додатково конвеєр обробки запитів включає підсистеми автентифікації та авторизації. Вони перехоплюють кожен вхідний запит до закритих каталогів і перевіряють наявність криптографічно підписаного сесійного ідентифікатора.

Якщо користувач не пройшов процедуру входу або його токен закінчився, конвеєр миттєво блокує запит на найнижчому рівні маршрутизації. Це не дозволяє системі витрачати процесорний час на завантаження сторінок експертизи чи обробку нелегітимних даних.

```
app.UseAuthentication();
app.UseAuthorization();
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Catalog}/{action=Index}/{id?}");Інтеграція механізмів проміжного програмного забезпечення формує нездоланний бар'єр перед бізнес-логікою додатка. Кожен HTTP-запит проходить через сувору фільтрацію, що гарантує виконання криптографічних операцій виключно для авторизованих власників контенту.
```

3.2 Організація бази даних SQLite та Entity Framework Core

Успішне функціонування будь-якого сучасного вебдодатка неможливе без надійної системи зберігання інформації. Після проєктування загальної архітектури серверного ядра на базі ASP.NET Core MVC виникла необхідність вибору оптимальної системи керування базами даних. Головним завданням цього архітектурного рівня виступає забезпечення тривалого збереження облікових записів, ліцензійних метаданих медіафайлів та ведення безперервного локального реєстру транзакцій. Для вирішення цих завдань у розробленому програмному комплексі реалізовано інтеграцію реляційної бази даних SQLite [17].

На відміну від громіздких клієнт-серверних систем управління базами даних, таких як PostgreSQL або Microsoft SQL Server, технологія SQLite пропонує кардинально інший інженерний підхід, ця система функціонує як максимально оптимізована локальна бібліотека, вона зберігає всю реляційну структуру таблиць, пошукові індекси та безпосередні записи у межах єдиного бінарного файлу, який розміщується безпосередньо на жорсткому диску сервера. Такий підхід повністю усуває мережеві затримки під час обміну даними між вебсервером і окремим сервером баз даних, а відсутність додаткових витрат часу на мережеву маршрутизацію значно підвищує загальну швидкість обробки інформації.

Зазначена характеристика є визначальною під час обчислення складних криптографічних відбитків та перевірки ланцюгів спадкоємства блоків у режимі реального часу, коли сервер не може чекати на відповідь від віддаленого вузла.

Взаємодія програмного коду, написаного мовою C#, з фізичним файлом бази даних здійснюється не напряму, а через об'єктно-реляційний перетворювач Entity Framework Core [18]. Використання цієї технології базується на сучасній методології попереднього опису коду. Фреймворк повністю абстрагує розробника від необхідності ручного написання низькорівневих запитів мовою SQL. Програміст формує строгу логічну структуру даних у вигляді звичайних класів, а перетворювач самостійно генерує та застосовує системні міграції для створення відповідної реляційної схеми.

Спеціальний асинхронний виклик методу ініціалізації у головному файлі програми гарантує автоматичну побудову таблиць при першому запуску вебдодатка. Цей програмний механізм позбавляє системного адміністратора необхідності вручну налаштовувати таблиці та зв'язки, зводячи до мінімуму людський фактор та ризик порушення цілісності схеми під час розгортання продукту.

```
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }
    // Декларація наборів даних (таблиць) реляційної СУБД SQLite
    public DbSet<User> Users { get; set; }
    public DbSet<MediaFile> MediaFiles { get; set; }
    public DbSet<BlockchainBlock> Blocks { get; set; }
}
```

Логічна схема розробленої бази даних суворо нормалізована і включає три ключові оптимізовані таблиці. Першою сутністю виступає таблиця користувачів, яка відповідає за зберігання облікових записів авторів контенту та налаштування політик безпеки, кожен запис містить унікальний ідентифікатор користувача та криптографічний відбиток його пароля. Алгоритм системи хешує паролі за

стандартом SHA-256 із додаванням модифікатора ще до моменту формування запиту на збереження у базу. Зберігання паролів у відкритому текстовому вигляді алгоритмічно виключено для запобігання компрометації облікових записів у випадку несанкціонованого фізичного доступу зловмисників до серверного сховища.

Другою і найважливішою сутністю є реєстр захищених цифрових активів. Ця таблиця гарантує внутрішню безпечну маршрутизацію файлів у системі. Відповідний запис інкапсулює початкову назву завантаженого медіафайлу, унікальний глобальний ідентифікатор формату GUID для безпечного рендерингу прев'ю та ім'я фізичного шифротексту у захищеному каталозі. Використання формату GUID повністю блокує атаки типу прямого доступу до об'єктів, оскільки зловмисник не здатний передбачити або підібрати випадковий 128-бітний ідентифікатор. Наявність зовнішнього ключа користувача у цій таблиці створює сувору логічну ізоляцію особистих каталогів. Це надійно блокує спроби доступу до чужих матеріалів на базовому рівні запитів бази даних. Додатково реєстр фіксує поточний хеш-відбиток файлу та криптографічний зв'язок із попереднім записом.

```
public string OriginalFileName { get; set; } = string.Empty;  // 
Початкове ім'я файлу при завантаженні
public string EncryptedFileName { get; set; } = string.Empty; // 
Ім'я фізичного .enc контейнера на сервері
public string Hash { get; set; } = string.Empty;              // 
SHA-256 геш-відбиток поточного контейнера
public string PreviousHash { get; set; } = string.Empty;      // 
Геш-відбиток попереднього блоку (Блокчейн-зв'язок)
```

Третя таблиця формує локальну імітаційну модель розподіленого блокчейн-реєстру. Структура цієї таблиці спроєктована виключно для забезпечення історичної незмінності подій у системі. Вона містить математичні індекси блоків, точні мітки часу генерації, відбитки даних транзакції та хеші попередніх операцій. Зазначені компоненти фізично реалізують математично нерозривну ланцюгову структуру на рівні реляційної системи керування базами даних. Зчитування даних із цієї таблиці відбувається за допомогою технології інтегрованих мовних запитів LINQ. Вбудований провайдер перетворює об'єктні запити у специфічні інструкції

бази даних, гарантуючи при цьому стовідсотковий захист від атак типу впровадження шкідливого SQL-коду.

Для забезпечення миттєвого пошуку записів під час проведення судово-криміналогічної експертизи реляційна схема використовує оптимізований механізм індексування. Текстові поля, які містять хеш-відбитки файлів, автоматично індексуються за допомогою алгоритму збалансованих дерев. Це забезпечує логарифмічний час пошуку інформації навіть за умови наявності десятків тисяч транзакцій у реєстрі. Без використання індексів серверу довелося б виконувати повне послідовне сканування таблиці під час кожного порівняння файлів, що неминуче призвело б до суттєвого зниження швидкодії програмного комплексу.

Усі операції запису та оновлення метаданих виконуються суворо в межах ізольованих транзакцій та в асинхронному режимі. Якщо під час багатоетапного процесу гібридного шифрування виникає апаратна або програмна помилка, об'єктно-реляційний перетворювач миттєво скасовує всі внесені зміни, повертаючи базу до попереднього стабільного стану. Така архітектурна особливість повністю відповідає строгим принципам узгодженості, ізольованості та довговічності. Виклик асинхронних методів збереження звільняє робочий потік вебсервера на час фізичного запису байтів на диск, загалом інтеграція такої оптимізованої бази даних безпосередньо у ядро вебдодатка дозволяє досягти максимальної пропускної здатності при формуванні цифрових конвертів.

3.3 Детальний опис програмної реалізації гібридного шифрування та логіки роботи коду

Конвеєр криптографічного перетворення ініціюється автоматично в момент отримання серверним контролером запиту з прикріпленим мультимедійним файлом. Основна логіка захисту інкапсульована у спеціалізованих сервісних класах і складається із суворо регламентованих етапів, вся архітектура програми

побудована так, щоб мінімізувати навантаження на оперативну пам'ять сервера під час обробки масивних графічних файлів.

Перший логічний етап ініціалізує процес просторової ізоляції та ідентифікації даних. Завантажений користувачем оригінальний файл перехоплюється системою та конвертується у двійковий потік в оперативній пам'яті за допомогою системних потокових класів. Програма ідентифікує поточного авторизованого користувача через механізми перевірки токенів доступу. Кожному новому об'єкту автоматично призначається унікальний криптографічний ідентифікатор. Цей крок гарантує повне виключення колізій імен у файловій системі сервера. Візуально зменшена та незашифрована копія файлу зберігається в окрему системну директорію для потреб майбутнього рендерингу віртуального полотна на стороні клієнта у закритій галереї.

```
// Визначення унікального ідентифікатора авторизованого користувача (автора)
int userId = 
int.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier)!);
// Генерація глобально унікального ідентифікатора (GUID) для депонованого файлу
string uniqueId = Guid.NewGuid().ToString();
string previewName = uniqueId + Path.GetExtension(file.FileName);
// Фізичне збереження попередньої копії файлу у спеціалізовану директорію сервера
System.IO.File.WriteAllBytes(Path.Combine(previewPath, previewName), originalFileData);
```

Наступний крок формує математичну нерозривність транзакцій: аналітична підсистема здійснює запит до таблиці блоків бази даних за допомогою інтегрованих мовних запитів для отримання криптографічного відбитка попереднього запису і у випадку якщо реєстр є абсолютно порожнім, то система розпізнає запуск нової мережі та застосовує заздалегідь визначену константу генезис-блоку, далі отриманий рядок хешу передається до методу стеганографічної ін'єкції, цей сервісний компонент здійснює пряму бінарну модифікацію, вшиваючи рядок попереднього відбитка безпосередньо у маркер кінця вихідного бінарного файлу, що формує фізичну нерозривність графічного контенту з історією всього

блокчейн-ланцюга, та навіть у випадку пошкодження або втрати центральної бази даних експертиза зможе зчитати метадані безпосередньо з самого файлу.

```
Buffer.BlockCopy(originalData, 0, combinedPayload, 0,
originalData.Length);
Buffer.BlockCopy(metadataBytes, 0, combinedPayload,
originalData.Length, metadataBytes.Length);
```

Після успішної інкапсуляції метаданих контролер ініціює етап гібридного криптоперетворення. Модифікований двійковий потік обробляється симетричним алгоритмом стандарту AES-256. Програма динамічно генерує унікальний псевдовипадковий сеансовий ключ розміром 32 байти та вектор ініціалізації розміром 16 байтів. Бінарний масив зображення жорстко шифрується у режимі зчеплення блоків. Використання режиму зчеплення блоків гарантує відсутність будь-яких візуальних патернів у результуючому шифротексті. Значення згенерованого вектора ініціалізації відкрито записується на самий початок згенерованого шифротексту, утворюючи преамбулу. Це є важливою умовою для забезпечення гарантованого детермінованого розшифрування легітимним власником у майбутньому без необхідності окремого зберігання вектора у базі даних.

Подальша алгоритмічна операція ілюструє процес безпечного експорту створеного симетричного ключа. Математичне ядро генерує надійну пару асиметричних ключів RSA із розміром модуля 4096 бітів. Сесійний ключ AES жорстко зашифровується публічною експонентою з обов'язковим застосуванням маскувальної схеми доповнення. Таке алгебраїчне перетворення формує повноцінний цифровий конверт. Масив байтів набуває характеристик абсолютно випадкового шуму, повністю блокуючи спроби частотного криптоаналізу.

Зашифрований криптоконтейнер записується на диск сервера у захищене фізичне сховище. Одразу після цього розраховується його фінальний відбиток за стандартом SHA-256. Цей відбиток фіксується у таблицях бази даних разом із початковим ім'ям об'єкта, завершуючи формування нового блоку розподіленого реєстру. Нижче на рисунку 3.1 приведена Блок-схема етапів шифрування зображення.

Рисунок 3.1- Блок-схема алгоритму шифрування

На фінальному етапі процесу шифрування сервер формує потоковий стиснений архів формату ZIP безпосередньо в оперативній пам'яті. У цей архів програмний комплекс упаковує зашифрований симетричний ключ та закритий приватний ключ RSA для майбутнього декодування. Створений архів примусово відправляється на локальний пристрій користувача через мережевий інтерфейс. Після успішного завершення передачі даних середовище виконання виконує безумовне очищення пам'яті від секретних матеріалів. Така архітектура повністю відповідає концепції нульового розголошення, залишаючи сервер без жодних інструментів для розшифрування інформації.

Процес зворотного перетворення ініціюється виключно легітимним власником контенту через завантаження відповідних ключів доступу у форму дешифрування. Сервіс безпеки зчитує завантажений приватний ключ RSA та

застосовує математичний апарат модульного піднесення до степеня для розшифрування симетричного ключа. Відновлений симетричний ключ передається алгоритму AES для повної реконструкції початкової матриці пікселів. Розшифрований файл повертається користувачу під новим ідентифікаційним ім'ям без збереження проміжних відкритих копій на жорсткому диску сервера.

```
byte[] aesKey = _sec.DecryptAESKeyWithRSA(encryptedAesKey,
privateRsaXml);
byte[] originalData = _sec.DecryptFileAES(encryptedFile, aesKey);
```

3.4 Механізми безпечної візуалізації (HTML5 Canvas)

Забезпечення конфіденційності медіафайлу на етапі його зберігання та мережевої передачі повністю вирішується криптографічними методами. Проте повноцінна система управління цифровими правами потребує ефективних засобів захисту контенту безпосередньо під час його легітимного відображення у браузері користувача. Головним завданням цього етапу виступає блокування спроб несанкціонованого копіювання розшифрованих візуальних даних кінцевими споживачами. Для реалізації цього завдання у розробленому вебдодатку застосовано комплексний підхід на базі технології віртуального полотна HTML5 Canvas.[21,22]

Більшість класичних вебдодатків використовують стандартні теги розмітки для виведення зображень у структуру об'єктної моделі документа. Зазначений підхід створює критичну вразливість на стороні клієнта. Будь-який відвідувач сторінки отримує можливість безперешкодно завантажити файл на локальний диск через базове контекстне меню браузера або застосувавши метод перетягування об'єкта мишею. Крім того, прямі посилання на графічні файли у вихідному коді легко перехоплюються автоматизованими скриптами та парсерами для масового піратського копіювання цілих галерей.

Розроблена система повністю відмовляється від використання стандартних графічних тегів. Бінарні дані розшифрованого прев'ю-файлу передаються на сторону клієнта та обробляються спеціальним алгоритмом мовою JavaScript. Цей

скрипт ініціалізує графічний контекст віртуального полотна та динамічно відмальовує матрицю пікселів у пам'яті браузера. Оскільки елемент віртуального полотна інтерпретується рушієм як програмно згенерована растрова область, базові системні інструменти збереження контенту для нього апаратно відсутні.

```
const ctx = canvas.getContext('2d');
ctx.drawImage(img, 0, 0);
```

Наведений лістинг демонструє процес ініціалізації двовимірного графічного контексту та позиціонування об'єкта. Для посилення превентивного захисту поверх відмальованого полотна з фотографією накладається абсолютно прозорий захисний блок. Цей контейнер позиціонується абсолютно відносно головного блоку та отримує максимальний індекс перекриття за віссю аплікат (координата Z). Якщо зловмисник спробує скопіювати елемент за допомогою вбудованих інструментів розробника у браузері, система дозволить виділити лише цей порожній прозорий шар. Оригінальний графічний контекст нижнього рівня залишається ізольованим від прямого доступу та інспектування.

Додатковим рівнем безпеки виступає жорстке алгоритмічне блокування інтерфейсних подій миші та клавіатури. За допомогою інтеграції спеціальних інструкцій система перехоплює та анулює виклик контекстного меню на всій робочій області закритої галереї. Одночасно деактивуються події початку перетягування. Це повністю унеможливлює спроби захоплення зображення курсором та його переміщення на робочий стіл операційної системи.

Критично важливим компонентом захисту виступає програмна генерація динамічних водяних знаків: перед фінальним виведенням пікселів на екран алгоритм змінює параметри глобальної прозорості графічного контексту і наносить діагональний напис для інформування про захист авторського права. Під текстовим попередженням скрипт автоматично виводить точне значення поточного криптографічного хешу транзакції та відбиток попереднього блоку бази даних.

```
ctx.font = "bold 24px Arial";
ctx.fillStyle = "rgba(255, 255, 255, 0.2)";
ctx.fillText("© @item.Author?.Username", 0, 0);
```

Застосування алгоритмічного рендерингу тексту безпосередньо поверх пікселів зображення радикально знижує комерційну цінність контенту у разі спроби застосування програм для створення знімків екрана. Вкрадена таким способом графічна копія міститиме незмивний маркер, який прямо вказує на факт крадіжки, він містить ідентифікаційні дані оригінального власника та засвідчує присутність об'єкта у захищеному блокчейн-реєстрі. Подібна архітектура фронтенд-захисту формує надійний превентивний бар'єр проти дрібного піратства та несанкціонованого розповсюдження інтелектуальної власності.

3.5 Модуль криптографічного аудиту та порівняння гешів

Проектування ефективної системи захисту авторських прав вимагає не лише надійного шифрування об'єктів, але й наявності інструментів для вирішення можливих юридичних суперечок. Саме для цього архітектура розробленого програмного комплексу містить окрему підсистему криптографічного аудиту. Головне завдання цього модуля полягає у проведенні судово-криміналогічної експертизи зашифрованих контейнерів. Аналіз базується на порівнянні математичних властивостей файлів та перевірці цілісності їхніх криптографічних зв'язків у локальному блокчейн-реєстрі.

Процедура аудиту починається на спеціалізованій вебсторінці експертизи. Користувач або експерт з інформаційної безпеки завантажує два захищені файли. Перший документ виконує роль оригінального еталона із закритого сховища автора. Другий файл розглядається як підозрілий об'єкт перевірки. Під час обробки цих масивних мультимедійних файлів серверний контролер застосовує оптимізований механізм потокового зчитування. Програма не завантажує весь обсяг файлу в оперативну пам'ять за один раз. Замість цього інформація читається невеликими порціями через системні класи потокового доступу. Це інженерне рішення запобігає переповненню купи великих об'єктів та гарантує стабільну пропускну здатність сервера навіть при одночасному аналізі кількох відеофайлів високої роздільної здатності.

Для отримання математичних відбитків файлів застосовується алгоритм SHA-256. Розрахунок виконується за допомогою вбудованих класів криптографічного простору імен платформи ASP.NET Core. Оскільки криптографічні об'єкти використовують некеровані ресурси операційної системи, їхнє створення та використання обгорнуто у спеціальні конструкції для гарантованого звільнення пам'яті після завершення обчислень.

```
using (SHA256 sha256Hash = SHA256.Create())
{
    byte[] bytes = sha256Hash.ComputeHash(rawData);
```

Безпосередній аналітичний процес експертизи поділяється на дві суворо регламентовані фази. Перша фаза передбачає перевірку абсолютної ідентичності цифрових активів. Аналітична підсистема здійснює пряме побітове порівняння двох згенерованих хеш-дайджестів. Повний збіг цих унікальних значень слугує беззаперечним математичним доказом того, що другий файл є точною копією першого. Такий результат свідчить про відсутність будь-яких несанкціонованих змін, переформатування чи спроб впровадження шкідливого коду у структуру медіафайлу.

Якщо ж алгоритм виявляє найменші розбіжності між первинними відбитками, автоматично запускається друга фаза аналізу. Цей етап полягає у перевірці блокчейн-зв'язків. Механізм формує запит до реляційної бази даних SQLite за допомогою технології інтегрованих мовних запитів LINQ. Програма здійснює пошук записів про завантажені криптоконтейнери. Далі програмний комплекс перевіряє сувору математичну умову спадкоємства блоків. Суть перевірки зводиться до порівняння хешу першого файлу зі значенням поля попереднього криптографічного зв'язку у записі другого файлу.

```
var file2Record = _db.MediaFiles.FirstOrDefault(m => m.Hash ==
hash2);
if (file2Record != null && file2Record.PreviousHash == hash1)
{
    isBlockchainLink = true;
}
```

Інтерфейсна частина експертного модуля побудована на базі адаптивного фреймворку Bootstrap та набору графічних елементів Bootstrap Icons. Форма завантаження документів містить інтегровані скрипти мовою JavaScript. Ці скрипти автоматично зчитують імена обраних файлів із файлової системи клієнта та виводять їх у спеціальні текстові поля. Зазначені поля жорстко заблоковані для ручного редагування властивістю лише для читання. Таке рішення повністю унеможливлює випадкові помилки експерта під час введення даних у процесі аудиту.

Панель результатів перевірки має чітке візуальне зонування. Перший файл відображається у картці з синім кольором акценту за допомогою каскадного класу первинної рамки. Другий файл виділяється червоним кольором для привернення уваги. Згенеровані хеші виводяться із застосуванням спеціальних класів розриву тексту та моноширинного шрифту. Використання моноширинного шрифту гарантує рівне відображення довгих шістнадцяткових рядків. Це суттєво полегшує візуальний контроль з боку експерта з інформаційної безпеки.

```
<h6 class="fw-bold text-primary">@ViewBag.File1Name</h6>
<code>@ViewBag.Hash1</code>

<h6 class="fw-bold text-danger">@ViewBag.File2Name</h6>
<code>@ViewBag.Hash2</code>
```

Серверна логіка динамічно обробляє результати перевірки та передає їх у представлення через словник даних системи. Змінна збігу інформує експерта про фізичну ідентичність об'єктів. Якщо ж підтверджується блокчейн-ланцюг, на екрані формується спеціальне інформаційне вікно з небесно-блакитним фоном та іконкою ланцюга. Система генерує текстове підтвердження успішної валідації. З юридичної точки зору це виступає доказом інкапсуляції другого файлу в систему строго хронологічно пізніше за перший. Будь-які інші комбінації трактуються підсистемою як порушення черговості транзакцій або спроба фальсифікації. У таких випадках на екран виводиться червоний попереджувальний індикатор.

Запропонований алгоритм формує міцну доказову базу для захисту прав автора в суді. Нижче на рисунку 3.2 продемонстровано Блок-схему цифрової експертизи

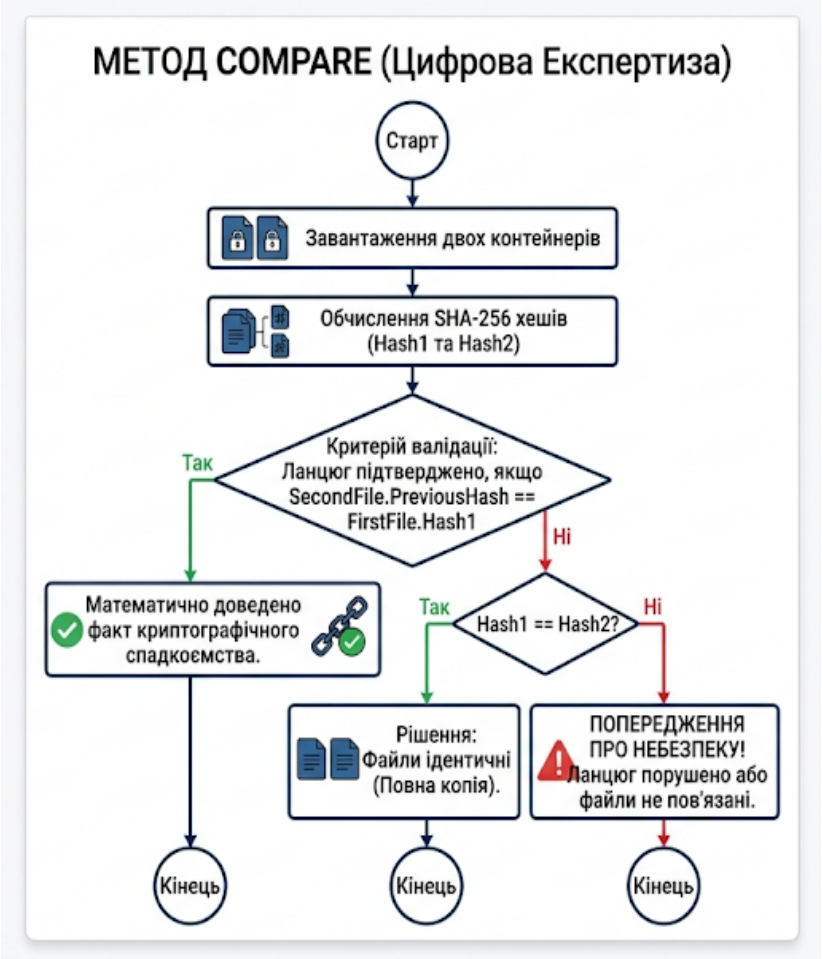

Рисунок 3.2 - Блок-схема цифрової експертизи

3.6 Проектування користувацького інтерфейсу (UI/UX) та взаємодія клієнтської частини з сервером

Програмна реалізація розробленої системи базується на класі контролера каталогу, цей компонент обробляє мережеві запити користувачів та керує процесом захисту медіафайлів. Контролер побудовано з використанням механізму впровадження залежностей. Базові сервіси підключаються через конструктор класу системи і до них належать: контекст бази даних, сервіс криптографічних операцій та модуль роботи з блокчейн-реєстром, що дає змогу відокремити математичну логіку шифрування від процесів маршрутизації та значно підвищує стабільність роботи програми.

Усі методи контролера працюють в асинхронному режимі. Використання асинхронних завдань запобігає блокуванню потоків вебсервера під час виконання тривалих обчислень,а завантажений користувачем файл надходить до контролера через стандартний інтерфейс обміну даними. Програма зчитує бінарний потік безпосередньо в оперативну пам'ять.

```
public CatalogController(AppDbContext db, SecurityService sec,
BlockchainService bc, IWebHostEnvironment env)
{
    _db = db;
    _sec = sec;
    _bc = bc;
    _env = env;
}
[HttpPost]
[Authorize]
public IActionResult Encrypt(IFormFile file)
```

Програмна реалізація захисту ініціюється викликом **асинхронного** методу контролера

`Encrypt`, який обробляє бінарний потік медіафайлу, завантажений користувачем через фронтенд-форму. Послідовність дій контролера суворо регламентована і забезпечує логічний ланцюг обробки, мінімізуючи ризики:

Адміністративна ідентифікація: Визначається ідентифікатор авторизованого користувача, генерується унікальний глобальний ідентифікатор (GUID) для файлу, а оригінальний файл тимчасово копіюється у спеціальну директорію для подальшої безпечної візуалізації мініатюр у закритій галереї (HTML5 Canvas).

Створення блокчейн-зв'язку: Контролер запитує хеш попереднього блоку транзакцій із реляційної бази даних. Знайдений хеш вбудовується у бінарну структуру файлу за допомогою методу стеганографічної ін'єкції в маркер кінця файлу, що забезпечує нерозривність контенту з історією реєстру.

Делегування шифрування: Після підготовки даних контролер викликає сервіс безпеки, який виконує повний цикл гібридного криптоперетворення (симетричне шифрування контенту та асиметричний захист сеансового ключа).

Фіналізація та експорт: Зашифрований контейнер записується у захищене сховище сервера. Контролер обчислює його фінальний відбиток (SHA-256) і фіксує його в базі даних як нову транзакцію. Створений ZIP-архів із приватними ключами (від RSA та AES) негайно передається користувачу для дотримання парадигми нульового розголошення.

```
byte[] encryptedFile = _sec.EncryptFileAES(combinedPayload, aesKey, iv);
var rsaKeys = _sec.GenerateRSAKeys();
```

На фінальному етапі зашифрований контейнер зберігається у закрите серверне сховище, а обчислений відбиток цього контейнера фіксується у реляційній базі даних. Зміни зберігаються за допомогою транзакційного виклику. Далі сервер створює в оперативній пам'яті стиснений архів формату ZIP. У цей архів система пакує зашифрований симетричний ключ та приватний ключ. Готовий архів відправляється на пристрій користувача. Після успішної передачі сервер виконує очищення пам'яті від приватних ключів.

Процес зворотного перетворення керується окремим асинхронним методом дешифрування, де для відновлення файлу користувач повинен надати зашифрований ключ доступу та власний приватний ключ. Сервіс безпеки застосовує математичний апарат алгоритму RSA для розшифрування симетричного ключа, потім ввідновлений ключ передається до дешифратора. Алгоритм знімає захист із масиву даних і ввідновлений файл повертається користувачу під новим іменем без збереження проміжних відкритих копій на жорсткому диску сервера.

```
byte[] aesKey = _sec.DecryptAESKeyWithRSA(encryptedAesKey, privateRsaXml);
byte[] originalData = _sec.DecryptFileAES(encryptedFile, aesKey);
```

```
return File(originalData, "application/octet-stream",
originalFileName);
```

3.7 Реалізація API-контролерів та опис життєвого циклу обробки HTTP-запитів

Серверна Програмна частина вебдодатка побудована навколо класу контролера каталогу. Цей клас приймає запити від користувачів і керує процесом захисту графічних файлів. Структура контролера базується на стандартному механізмі передачі залежностей. Через конструктор до класу підключаються контекст бази даних, сервіс криптографії та модуль для роботи з блокчейн-реєстром. Таке архітектурне рішення відокремлює математичні обчислення від маршрутизації мережевих запитів, завдяки чому вихідний код стає чистішим, а сервер працює стабільніше при збільшенні навантаження.

Більшість методів контролера працюють в асинхронному режимі. Використання асинхронних функцій довебсерверу не блокувати робочі потоки під час шифрування великих мультимедійних файлів. Завантаження графіки від клієнта відбувається через стандартний інтерфейс обміну даними форм. Програма зчитує вхідний бінарний потік по частинах безпосередньо в оперативну пам'ять. Це запобігає перевантаженню сервера у ситуаціях, коли кілька користувачів одночасно завантажують об'ємні матеріали.

```
public CatalogController(AppDbContext db, ILogger<CatalogController>
logger)
{
    _db = db;
    _logger = logger;
}
public IActionResult Encrypt(IFormFile file)
{
    using var ms = new MemoryStream();
    file.CopyTo(ms);
    byte[] data = ms.ToArray();
```

Процес створення цифрового конверта починається з виклику методу шифрування. Спочатку програма визначає ідентифікатор поточного користувача через механізм перевірки сесії. Це потрібно для правильного розподілу доступу до файлів у майбутньому та логічної ізоляції каталогів. Для кожного нового об'єкта генерується унікальний ідентифікатор формату GUID. Використання такого формату виключає можливість збігу імен файлів на жорсткому диску. Одразу після ідентифікації оригінальний файл копіюється у спеціальну папку для збереження прев'ю. Ця відкрита зменшена копія потрібна тільки для відмальовки мініатюр у закритій галереї на стороні клієнта.

Далі програма формує зв'язок нового файлу з блокчейн-реєстром. Виконується запит до бази даних за допомогою об'єктно-реляційного перетворювача для отримання криптографічного відбитка попереднього запису. Якщо таблиця є порожньою, система використовує базову текстову константу генезис-блоку. Знайдений відбиток передається до сервісу безпеки. Програмний алгоритм знаходить маркер кінця завантаженого файлу і дописує туди рядок з отриманим відбитком. Ця операція створює фізичну нерозривність контенту з історією всього ланцюга транзакцій. Навіть при пошкодженні центральної бази даних спеціаліст зможе прочитати ці метадані безпосередньо з самого графічного файлу.

```
string uniqueId = Guid.NewGuid().ToString();
string fileName = uniqueId + Path.GetExtension(file.FileName);
System.IO.File.WriteAllBytes(Path.Combine(previewPath, fileName),
data);
string previousHash = _db.MediaFiles.OrderByDescending(m => m.Id)
                    .Select(m => m.Hash).FirstOrDefault() ?? "0";
```

Після фіксації блокчейн-зв'язку (вбудовування попереднього хешу у бінарний потік) контролер ініціює етап гібридного криптоперетворення, повністю делегуючи виконання всіх криптографічних операцій спеціалізованому сервісному компоненту SecurityService. Це відповідає архітектурному

принципу розділення відповідальності (Separation of Concerns), відповідно до якого контролер виступає лише диспетчером запитів.

Сервіс безпеки виконує повний цикл захисту, що включає генерацію ключів RSA-4096, симетричне шифрування файлу (AES-256 у режимі CBC) та асиметричну інкапсуляцію сеансового ключа.

Контролер, отримавши на виході повністю зашифрований контейнер та криптографічний конверт із захищеним симетричним ключем, зберігає їх у відповідні сховища. Розрахований фінальний відбиток контейнера (SHA-256) фіксується у реляційній базі даних.

```
byte[] encryptedFile = _sec.EncryptFileAES(combinedPayload, aesKey, iv);
byte[] encryptedAesKey = _sec.EncryptAESKeyWithRSA(aesKey, rsaPublicKey);
_db.MediaFiles.Add(new MediaFile { Hash = currentHash, PreviousHash = previousHash });
await _db.SaveChangesAsync();
byte[] zipArchive = _sec.CreateArchive(encryptedFile, encryptedAesKey, currentHash);]
```

Готовий стиснений архів передається безпосередньо на пристрій користувача через мережеву відповідь. Архів генерується виключно в пам'яті без створення тимчасових файлів на жорсткому диску сервера, також ппісля завершення передачі даних середовище виконання очищує оперативну пам'ять від приватних ключів.

Процес відновлення файлу керується окремим методом дешифрування:

Для отримання доступу користувач завантажує через вебформу зашифрований симетричний ключ та власний приватний ключ. Контролер перехоплює ці файли та передає їх у сервіс безпеки. За допомогою закритої експоненти розшифровується симетричний ключ. Відновлений ключ передається до модуля симетричного дешифратора, програма знімає криптографічний захист із масиву даних та відкидає вектор ініціалізації. Відновлений медіафайл повертається користувачу у вигляді двійкового потоку під новим іменем без збереження відкритих копій у системі.

```
byte[] aesKey = _sec.DecryptAESKeyWithRSA(encryptedAesKey,
privateRsaXml);
byte[] originalData = _sec.DecryptFileAES(encryptedFile, aesKey);
```

4 ОХОРОНА ПРАЦІ ТА ПРОФЕСІЙНА БЕЗПЕКА ФАХІВЦІВ З ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Професійна діяльність фахівців з інформаційних технологій (ІТ) пов'язана з тривалим перебуванням у приміщенні та безперервним використанням комп'ютерної техніки. З точки зору охорони праці, таких працівників відносять до категорії офісних службовців — користувачів комп'ютерів. Їхня праця є суто розумовою та характеризується високою напруженістю, значними статичними фізичними навантаженнями, малою руховою активністю (гіподинамією), а також тривалим напруженням зору та нервової системи [24].

Надійність роботи в системі «людина-комп'ютер» безпосередньо залежить від функціонального стану працівника. Будь-які психофізіологічні перевантаження, втома чи стрес знижують увагу та призводять до помилок у роботі. Незадовільні умови праці можуть викликати професійні захворювання, що веде до втрати працездатності. Тому розробка та дотримання вимог охорони праці є обов'язковою умовою для збереження здоров'я та високої продуктивності ІТ-фахівців.

4.1 Способи зниження навантаження на серцево-судинну систему та ЦНС фахівців ІТ-галузі при їх професійній діяльності

Трудова діяльність ІТ-фахівців проходить у виробничому середовищі, яке постійно впливає на їхній організм. Найбільший вплив мають фізичні фактори: електромагнітні хвилі, статичні електричні поля, рівень шуму, параметри мікроклімату та освітлення. Хімічні та біологічні фактори в офісах мають значно менший вплив [25].

Для зниження навантаження на серцево-судинну систему та центральну нервову систему (ЦНС) застосовують комплекс технічних, санітарно-гігієнічних та організаційних заходів.

4.1.1 Створення оптимального мікроклімату та обміну повітря

Розумова праця за комп'ютером належить до категорії легких фізичних робіт 1а. Для цієї категорії, згідно з нормами ДСН 3.3.6.042-99, в робочій зоні необхідно підтримувати оптимальні мікрокліматичні умови [25]. Вони забезпечують нормальний тепловий стан організму без напруження системи терморегуляції, що безпосередньо знижує навантаження на серце та судини.

Нормовані параметри мікроклімату для категорій робіт 1а наведено в таблиці 4.1.

Таблиця 4.1. Оптимальні параметри мікроклімату для приміщень з комп'ютерною технікою (категорія 1а)

Пора року	Температура повітря, °С	Відносна вологість, %	Швидкість руху повітря, м/с
Холодна	22-24	40-60	0.1
Тепла	23-25	40-60	0.1

Окрім температури та вологості, важливе значення має іонний склад повітря. Оптимальний вміст легких іонів у 1 куб. см повітря становить: від 1500 до 3000 позитивних іонів та від 3000 до 5000 негативних іонів [26].

Для підтримки цих параметрів приміщення обладнують системами опалення, кондиціювання або припливно-витяжною вентиляцією. Необхідний об'єм подачі свіжого повітря розраховують залежно від об'єму приміщення на одного працівника :

при об'ємі приміщення до 20 куб. м на особу — подача повітря має бути не менше 30 куб. м/год на кожного працівника ;

при об'ємі від 20 до 40 куб. м — не менше 20 куб. м/год ;

при об'ємі понад 40 куб. м (за наявності вікон та відсутності шкідливих виділень) допускається використання природної вентиляції.

4.1.2 Організація раціонального освітлення

Недостатнє або неправильне освітлення викликає швидку втому очей, що рефлекторно передається на ЦНС, викликаючи головний біль, дратівливість та загальну слабкість. Освітлення приміщень має відповідати ДБН В.2.5-28-2006 [27].

Природне освітлення має здійснюватися через бічні вікна, які бажано орієнтувати на північ або північний схід. Коефіцієнт природної освітленості (КПО) повинен бути не нижчим за 1,5%. Для запобігання відблискам вікна обладнують регульованими жалюзі або шторами. Всі поверхні в приміщенні повинні мати матову або напівматову фактуру, щоб уникнути світлових блисків.

Норми коефіцієнтів відбиття світла становлять: для стелі — 0,7-0,8; стін — 0,5-0,6; підлоги — 0,3-0,5; робочих поверхонь — 0,4-0,5.

Штучне освітлення організовують за системою загального рівномірного освітлення. Світильники розміщують паралельно лінії зору (переважно ліворуч від робочих місць). Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300-500 лк. Коефіцієнт пульсації світла не повинен перевищувати 5%. Для запобігання зоровому дискомфорту пряма блисківка від джерел світла обмежується яскравістю до 200 кд/кв. м, а яскравість відблисків на екрані комп'ютера — не більше 40 кд/кв. м. Очищення скла вікон та світильників має проводитися не рідше ніж два рази на рік.

4.1.3 Організаційні та профілактичні заходи

Основними організаційними способами зниження навантаження на організм є правильний підбір кадрів, раціональне облаштування робочих місць та дотримання оптимального режиму праці й відпочинку.

Для організації безпечного робочого простору застосовують такі ергономичні вимоги (згідно з ДСТУ 8604:2015)[28] :

площа на одне робоче місце має бути не менше 6 кв. м, а об'єм — не менше 20 куб. м ;

робочі столи розміщують на відстані не менше 1 м від стін зі світловими прорізами, а відстань між бічними поверхнями сусідніх моніторів має бути не менше 1,2 м ;

висота робочого столу повинна становити 680-800 мм, ширина — 600-1400 мм, глибина — 800-1000 мм ;

стіл обов'язково обладнують підставкою для ніг шириною не менше 300 мм ;

робоче крісло має бути підйомно-поворотним, з регулюванням висоти сидіння (400-500 мм) та кута нахилу спинки; для зниження напруження рук обов'язкова наявність підлокітників довжиною не менше 250 мм ;

відстань від очей до екрана монітора має становити 600-800 мм.

Найважливішим чинником захисту серцево-судинної системи та ЦНС є впровадження регламентованих перерв протягом 8-годинної робочої зміни. Санітарними нормами встановлено такі режими відпочинку залежно від характеру праці :

Для розробників програмного забезпечення — регламентована перерва тривалістю 15 хвилин через кожну годину роботи за комп'ютером.

Для операторів комп'ютерного набору — регламентована перерва тривалістю 10 хвилин після кожної години роботи.

Для інших операторів комп'ютерних систем — перерва 15 хвилин через кожні дві години роботи.

При 12-годинній зміні перерви в перші 8 годин призначають аналогічно, а протягом останніх 4 годин — незалежно від виду діяльності, обов'язково через кожну годину тривалістю 15 хвилин. Безперервна робота за комп'ютером без перерви за будь-яких обставин не повинна перевищувати 4 години.

Для профілактики застійних явищ та відновлення тонусу судин під час перерв рекомендується проводити виробничу гімнастику. Вона допомагає зняти м'язову напругу, покращує кровообіг у ногах та органах малого таза. Також ефективним методом є використання кімнат психофізіологічного розвантаження, де проводяться 10-хвилинні сеанси релаксації, які складаються з трьох періодів:

відволікання (під тиху музику), заспокоєння (перегляд слайдів при зеленому світлі) та активізації (бадьора музика та червоне світло для відновлення працездатності).

4.2 Аналіз ризиків, пов'язаних з переробками, та їх вплив на здоров'я та продуктивність праці фахівців інформаційних технологій

У сфері інформаційних технологій через часті стислі терміни (дедлайни), нерівномірне планування та високі вимоги до результату виникає проблема надурочної роботи. Однак, згідно зі статтею 65 Кодексу законів про працю України[29] надурочні роботи суворо обмежуються: вони не повинні перевищувати 4 годин протягом двох днів поспіль та 120 годин на рік для кожного працівника.

Систематичне порушення нормального розпорядку дня та постійні переробки несуть серйозні ризики для здоров'я працівників.

4.2.1 Вплив переробок на соматичне здоров'я та ЦНС

Постійна робота понад норму без належного відпочинку негативно впливає на всі системи організму :

Порушення сну. Розумове збудження під час тривалого вирішення складних завдань у вечірній час перешкоджає нормальному засинанню. Світло від моніторів додатково пригнічує вироблення мелатоніну, що призводить до хронічного безсоння та заважає відновленню нервових клітин.

Артеріальна гіпертензія та розлади кровообігу. Постійна напруга активує викид стресових гормонів, що спричиняє звуження судин та підвищення тиску. Сидяча робота тривалістю понад 10,6 години на добу подвоює ризик розвитку серцево-судинних хвороб через депонування крові в нижніх кінцівках та венозний застій.

Гіподинамія. Недостатня рухова активність послаблює м'язовий корсет, порушує поставу, викликає дистрофічні зміни в хребті (остеохондроз).

Порушення шлунково-кишкового тракту. Нерегулярне харчування, поспіх під час їжі та стрес порушують секрецію шлункового соку й моторику кишківника, що веде до гастритів чи розладів травлення.

Зниження фізичної працездатності. Організм втрачає здатність опиратися інфекціям, падає загальний тонус [30].

4.2.2 Динаміка працездатності та розвиток втоми

Працездатність відображає можливість людини виконувати завдання з належною якістю протягом визначеного часу. Вона має фазовий характер і змінюється протягом робочого дня. Виділяють чотири періоди працездатності, які наведено в таблиці 4.2.

Таблиця 4.2. Характеристика періодів працездатності протягом зміни

Період	Назва періоду	Фізіологічна характеристика	Тривалість та особливості
I	Втягування в роботу	Поступове підвищення працездатності. В розумову діяльність людина включається повільніше, ніж у фізичну.	Може тривати до години і більше.
II	Стійка працездатність	Висока продуктивність праці, всі фізіологічні функції перебувають у нормі.	Тривалість залежить від інтенсивності навантаження.

Період	Назва періоду	Фізіологічна характеристика	Тривалість та особливості
III	Субкомпенсація	Початок розвитку втоми. Продуктивність праці ще висока, але підтримується за рахунок вольових зусиль та перенапруження організму.	Є перехідною фазою до втоми.
IV	Втома	Падіння продуктивності праці, різке погіршення функціонального стану.	Супроводжується суб'єктивними відчуттями втоми та болю.

При переробках працівник вимушений працювати переважно в четвертому періоді — періоді втоми. Для ІТ-фахівців у цей час характерні такі симптоми:

сильний головний біль ;

судоми (спазми) м'язів кисті руки від однотипних рухів мишкою та клавіатурою ;

ниючий біль у ділянці спини та шиї через тривале сидяче положення ;

морально-психологічне напруження, підвищена дратівливість або, навпаки, повна апатія.

Якщо в цей момент роботу не припинити, втома накопичується і переходить у перевтому. Втома є природним зворотнім процесом, що зникає після звичайного відпочинку, тоді як перевтома характеризується накопиченням несприятливих змін, які є важко зворотними. При хронічній перевтомі відновлювального періоду

(звичайної ночі сну) вже недостатньо, що призводить до стійких патологічних змін у ЦНС та серцево-судинній системі[37].

На працездатність впливають як внутрішні чинники (рівень підготовки, вік, загальний стан здоров'я, тренованість), так і зовнішні (зручність робочого місця, санітарно-гігієнічні умови в приміщенні, психологічний клімат у колективі). Нераціональні переробки, так само як і недостатнє або нерівномірне навантаження, ведуть до дезорганізації роботи, різкого зростання кількості помилок у програмному коді та швидкого виснаження працівника [33].

Окремим небезпечним наслідком тривалого некомпенсованого робочого стресу та переробок є синдром професійного вигорання. Він розвивається поступово. Основними його проявами є постійне відчуття браку енергії, психологічне відсторонення від своїх обов'язків, дратівливість, погіршення концентрації уваги, цинічне ставлення до колег та результатів власної праці. За даними досліджень, IT-фахівці є вкрай вразливими до цього синдрому через постійну роботу в умовах жорстких дедлайнів та високої розумової напруги.

Таким чином, усунення практики систематичних переробок, дотримання нормального розпорядку робочого дня та впровадження регламентованих перерв є базовими вимогами гігієни праці, що забезпечують збереження здоров'я та стабільно високу продуктивність праці IT-фахівців [39].

ВИСНОВКИ

Перехід до цифрових форматів передачі інформації створив умови, що потребують нових методів захисту інтелектуальної власності. Через властивості цифрових медіаоб'єктів, що дозволяють створювати безліч ідентичних копій без деградації якості, виникла потреба у розробці новітніх інструментів технічного захисту контенту.

Проведене дослідження довело, що використання виключно ізольованих симетричних чи асиметричних криптографічних систем є неефективним рішенням для динамічного вебсередовища. Симетричні алгоритми надають високу швидкість шифрування масивних файлів, але не здатні самостійно вирішити проблему безпечного розповсюдження ключів через відкриті канали зв'язку, а пряме використання асиметричних алгоритмів для захисту мегабайтів пікселів призводить до критичного перевантаження серверних потужностей та неприпустимих затримок у роботі платформи.

Розв'язання цієї проблеми було досягнуто шляхом розробки та впровадження оптимізованої математичної моделі гібридного шифрування. Створений програмний комплекс успішно комбінує швидкість потокових симетричних перетворень (AES-256) із надійністю асиметричної інкапсуляції сесійних маркерів (RSA-4096). Для RSA-шифрування обрано стандарт доповнення OAEP, тому що його застосування робить шифрування ймовірнісним, тобто навіть за умови повторного шифрування одного і того самого файлу - вихідний результат завжди різниться і це усуває проблему детермінованості RSA і захищає дані від спроб статистичного аналізу, коли зловмисник намагається вирахувати структуру даних за частотою появи однакових блоків шифротексту

Розроблений вебдодаток "CryptoShield Pro" продемонстрував високу ефективність поєднання класичних методів криптографії та децентралізованих технологій, а впровадження імітаційної блокчейн-моделі дозволило реалізувати механізм беззаперечного доведення авторства на базі національного стандарту криптографічного хешування. Стеганографічне вбудовування маркерів попередніх

блоків транзакцій безпосередньо у бінарний потік графічного контенту гарантує збереження ланцюга навіть у разі пошкодження центральної бази даних.

Практична значимість роботи підтверджена створенням багаторівневого модульного середовища, здатного не лише надійно шифрувати графічні активи, але й проводити автоматизовану експертизу файлів. Фронтенд-захист на основі технології HTML5 Canvas із динамічними водяними знаками довів свою здатність ефективно блокувати спроби несанкціонованого копіювання медіаданих кінцевими користувачами. Розроблені методи захисту підтвердили ефективність поєднання криптографічних моделей та архітектури нульового розголошення для забезпечення безпеки авторського права в умовах відкритого мережевого середовища.

ПЕРЕЛІК ПОСИЛАНЬ

1. **Diffie W., Hellman M.** New directions in cryptography. *IEEE Transactions on Information Theory*. 1976. Vol. 22, No. 6. P. 644–654.
2. **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.
3. **Bellare M., Rogaway P.** Optimal Asymmetric Encryption -- How to encrypt with RSA. *Advances in Cryptology - EUROCRYPT '94*. Lecture Notes in Computer Science, Vol. 950. Berlin: Springer, 1995. P. 92–111.
4. **FIPS PUB 180-2.** Secure Hash Standard. Federal Information Processing Standards Publication 180-2. National Institute of Standards and Technology (NIST), 2002.
5. **ДСТУ 7564:2014.** Інформаційні технології. Криптографічний захист інформації. Функція гешування. Київ: ДП «УкрНДНЦ», 2015. 40 с.
6. **Shor P. W.** Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*. 1997. Vol. 26, No. 5. P. 1484–1509.
7. **Anderson R.** *Security Engineering: A Guide to Building Dependable Distributed Systems*. 3rd ed. Indianapolis: John Wiley & Sons, 2020. 1222 p.
8. **Bonnet S., Teuteberg F.** Impact of blockchain and distributed ledger technology for the management, protection, enforcement and monetization of intellectual property: a systematic literature review. *Information Systems and e-Business Management*. 2023. Vol. 21, No. 2. P. 229–275.
9. **Haber S., Stornetta W. S.** How to time-stamp a digital document. *Journal of Cryptology*. 1991. Vol. 3, No. 2. P. 99–111.
10. **Bayer D., Haber S., Stornetta W. S.** Improving the Efficiency and Reliability of Digital Time-Stamping. In: Capocelli R., De Santis A., Vaccaro U. (Eds.). *Sequences II: Methods in Communication, Security, and Computer Science*. New York: Springer, 1993. P. 329–334.

11. **Ganic E., Eskicioglu A. M.** Robust DWT-SVD domain image watermarking: Embedding data in all frequencies. *Proceedings of the Workshop on Multimedia and Security*. ACM, 2004. P. 166–174.
12. **Freeman A.** *Pro ASP.NET Core 8: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages*. 10th ed. New York: Apress, 2024. 1100 p.
13. **Smith S.** *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. Redmond: Microsoft Corporation, 2022. 119 p.
14. **Freeman A.** *Pro Entity Framework Core 8 for ASP.NET Core MVC*. New York: Apress, 2024. 640 p.
15. **Haldar S.** *SQLite Database System: Design and Implementation*. 2nd ed. Sibsankar Haldar, 2016. 264 p.
16. **Albahari J.** *C# 12 in a Nutshell: The Definitive Reference*. Sebastopol: O'Reilly Media, 2023. 1086 p.
17. **Price M. J.** *C# 12 and.NET 8 – Modern Cross-Platform Development*. Birmingham: Packt Publishing, 2023. 822 p.
18. **Flanagan D.** *JavaScript: The Definitive Guide*. 7th ed. Sebastopol: O'Reilly Media, 2020. 706 p.
19. **Rauschmayer A.** *JavaScript for Impatient Programmers*. ECMA International, 2021. 520 p.
20. **Duckett J.** *HTML and CSS: Design and Build Websites*. Indianapolis: John Wiley & Sons, 2011. 490 p.
21. **Ferguson N., Schneier B.** *Practical Cryptography*. Indianapolis: John Wiley & Sons, 2003. 410 p.
22. **Трофименко О. Г., Прокоп Ю. В.** *Архітектура та екосистема сучасних вебдодатків: навчальний посібник*. Одеса: ОНАТ, 2021. 256 с.
23. **Lidl R., Niederreiter H. Finite Fields**. *Cambridge: Cambridge University Press,* 1997. 755 p.
24. ПРОФЕСІЙНА БЕЗПЕКА ФАХІВЦІВ З ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ. Лекція 7. Фактори впливу та працездатність.

25. БІОЛОГІЧНІ НЕБЕЗПЕКИ. ЗАБЕЗПЕЧЕННЯ ЗДОРОВ'Я ЛЮДИНИ У СУЧАСНОМУ ВИРОБНИЦТВІ. Лекція 3. Профілактичні заходи та режими відпочинку.
26. ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень.
27. ДБН В.2.5-28-2006. Природне і штучне освітлення.
28. ДСТУ 8604:2015. Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи.
29. Кодекс законів про працю України (ст. 65, 106).
30. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями.
31. Оптимальні та допустимі мікрокліматичні умови - Служба охорони праці.
32. МІКРОКЛІМАТ - кнуба.
33. Понаднормова робота: чому ми перепрацьовуємо та що каже КЗпП - Happy Monday.
34. Визначаємо рівень професійного вигорання у працівників - Охорона праці.
35. Тема лекції 10. Робочий час План 1. Поняття робочого часу. 2. Нормований р.
36. НАПРУЖЕНІСТЬ ТРУДОВОГО ПРОЦЕСУ - Роменська міська рада.
37. Як сидяча робота шкодить вашому серцю - ТСН.
38. Втома на роботі і безпека праці - Профспілка працівників освіти.
39. Професійне вигорання в ІТ: ознаки, причини, способи подолання - QATestLab.
40. Безпека праці та промислова санітарія - 2.2.3. Нормування параметрів мікроклімату.
41. Оплачуємо надурочні години. Оплата праці, № 23, Грудень, 2018 | Factor.
42. Робота за комп'ютером: які правила та вимоги діють в Україні - Kadroland.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ "CRYPTOSHIELD PRO"

А.1 Шар моделей даних (папка `Models`)

Файл `Models/User.cs`

C#

```
namespace CryptoWebProject.Models
{
    /// <summary>
    /// Сутність користувача для підсистеми автентифікації та розмежування прав
доступу.
    /// Використовується ініціалізація порожнім рядком (string.Empty) для
запобігання
    /// виключенням типу NullReferenceException згідно зі стандартами безпеки C#
8.0+.
    /// </summary>
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; } = string.Empty;
        public string PasswordHash { get; set; } = string.Empty; // Зберігання
криптографічного хешу замість відкритого пароля
    }
}}
```

Файл Models/MediaFile.cs

C#

```
using System;
using System.ComponentModel.DataAnnotations;

namespace SecureCryptoSystem.Models
{
    /// <summary>
    /// Клас-сутність "Медіафайл" (MediaFile).
    /// Призначення: Опис структури збереження метаданих захищеного контенту.
    /// Реалізує розподілену топологію імен для коректного рендерингу та
скачування.
    /// </summary>
    public class MediaFile
    {
        [Key]
        public int Id { get; set; } // Первинний ключ таблиці

        public string OriginalFileName { get; set; } = string.Empty;  // Початкове
ім'я файлу при завантаженні
        public string PreviewFileName { get; set; } = string.Empty;   // Унікальне
GUID-ім'я для безпечної галереї Canvas
        public string EncryptedFileName { get; set; } = string.Empty; // Ім'я
фізичного.enc контейнера на сервері

        public string Hash { get; set; } = string.Empty;              // SHA-256
геш-відбиток поточного контейнера
        public string PreviousHash { get; set; } = string.Empty;      // Геш-
відбиток попереднього блоку (Блокчейн-зв'язок)
```

```
        public int UserId { get; set; } // Зовнішній ключ (Foreign Key) для
зв'язку з автором
        public User? Author { get; set; } // Навігаційна властивість для ORM
Entity Framework

        public DateTime UploadDate { get; set; } = DateTime.UtcNow; // Точний
часовий штамп депонування
    }
}
```

Файл `Models/BlockchainBlock.cs`

C#

```
using System;
using System.ComponentModel.DataAnnotations;

namespace SecureCryptoSystem.Models
{
    /// <summary>
    /// Клас-сутність "Блокчейн-блок" (BlockchainBlock).
    /// Призначення: Опис структури даних вузла розподіленого реєстру для фіксації
транзакцій.
    /// </summary>
    public class BlockchainBlock
    {
        [Key]
        public int Id { get; set; } // Первинний ключ

        public int Index { get; set; } // Порядковий індекс блоку у ланцюгу
        public DateTime Timestamp { get; set; } // Часовий штамп реєстрації блоку
        public string DataHash { get; set; } = string.Empty; // Геш корисного
навантаження (зашифрованого файлу)
        public string PreviousHash { get; set; } = string.Empty; // Геш-вказівник
на попередній блок ланцюга
        public string Hash { get; set; } = string.Empty; // Власний
криптографічний геш-відбиток всього блоку
    }
}
```

Файл Models/ErrorViewModel.cs

C#

```
using System;

namespace SecureCryptoSystem.Models
{
    /// <summary>
    /// Клас-модель представлення помилок (ErrorViewModel).
    /// Призначення: Передача технічних параметрів та кодів збоїв для шару
представлення.
    /// </summary>
    public class ErrorViewModel
    {
        public string? RequestId { get; set; } // Унікальний ідентифікатор HTTP-
запиту

        public bool ShowRequestId =>!string.IsNullOrEmpty(RequestId); // Геттер
перевірки наявності ID
    }
}
```

А.2 Шар доступу до даних (папка `Data`)

Файл Data/AppDbContext.cs

C#

```
using Microsoft.EntityFrameworkCore;
using SecureCryptoSystem.Models;

namespace SecureCryptoSystem.Data
{
    /// <summary>
    /// Клас контексту бази даних (AppDbContext).
    /// Призначення: Реалізація доступу до бази даних SQLite за технологією ORM
Entity Framework Core.
    /// </summary>
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

        // Декларація наборів даних (таблиць) реляційної СУБД SQLite
        public DbSet<User> Users { get; set; }
        public DbSet<MediaFile> MediaFiles { get; set; }
        public DbSet<BlockchainBlock> Blocks { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
        {
            // Налаштування підключення до локального файлу бази даних
            if (!optionsBuilder.IsConfigured)
            {
                optionsBuilder.UseSqlite("Data Source=crypto.db");
            }
        }
    }
}
```

А.3 Шар сервісів бізнес-логіки (папка `Services`)

Файл Services/SecurityService.cs

C#

```
using System;
using System.IO;
using System.Security.Cryptography;
using System.Text;

namespace SecureCryptoSystem.Services
{
    /// <summary>
    /// Клас криптографічного ядра (SecurityService).
    /// Призначення: Реалізація алгоритмів гібридного шифрування (AES-256 + RSA-
4096),
    /// обчислення гешів SHA-256 та стеганографічного впровадження метаданих у
бінарний потік.
    /// </summary>
    public class SecurityService
    {
        // Унікальна системна мітка-заголовок для первинної верифікації
        public const string SIG_ID = "CRYPTPRO";

        /// <summary>
```

```
        /// Обчислення криптографічного відбитка даних за алгоритмом SHA-256.
        /// </summary>
        public string ComputeHash(byte rawData)
        {
            using (SHA256 sha256Hash = SHA256.Create())
            {
                byte bytes = sha256Hash.ComputeHash(rawData);
                StringBuilder builder = new StringBuilder();
                for (int i = 0; i < bytes.Length; i++)
                    builder.Append(bytes[i].ToString("x2"));
                return builder.ToString();
            }
        }

        /// <summary>
        /// Стеганографічне впровадження метаданих Блокчейну (попереднього гешу)
безпосередньо у бінарну структуру файлу.
        /// </summary>
        public byte EmbedBlockchainMetadata(byte originalData, string
previousHash)
        {
            byte metadataBytes = Encoding.UTF8.GetBytes($"");
            byte combinedPayload = new byte;

            Buffer.BlockCopy(originalData, 0, combinedPayload, 0,
originalData.Length);
            Buffer.BlockCopy(metadataBytes, 0, combinedPayload,
originalData.Length, metadataBytes.Length);

            return combinedPayload;
        }

        /// <summary>
        /// Генерація пари асиметричних ключів RSA (2048 біт) у форматі XML.
        /// </summary>
        public (string PublicKey, string PrivateKey) GenerateRSAKeys()
        {
            using (RSA rsa = RSA.Create(2048))
            {
                return (rsa.ToXmlString(false), rsa.ToXmlString(true));
            }
        }

        /// <summary>
        /// Асиметричне шифрування сесійного ключа AES за допомогою відкритого
ключа RSA (OAEP Padding).
        /// </summary>
        public byte EncryptAESKeyWithRSA(byte aesKey, string publicKeyXml)
        {
            using (RSA rsa = RSA.Create())
            {
                rsa.FromXmlString(publicKeyXml);
                return rsa.Encrypt(aesKey, RSAEncryptionPadding.Pkcs1);
            }
        }

        /// <summary>
        /// Асиметричне дешифрування сесійного ключа AES за допомогою закритого
ключа RSA.
        /// </summary>
        public byte DecryptAESKeyWithRSA(byte encryptedAesKey, string
privateKeyXml)
        {
```

```
            using (RSA rsa = RSA.Create())
            {
                rsa.FromXmlString(privateKeyXml);
                return rsa.Decrypt(encryptedAesKey, RSAEncryptionPadding.Pkcs1);
            }
        }

        /// <summary>
        /// Симетричне блокове шифрування масиву даних за стандартом AES-256
(режим CBC).
        /// </summary>
        public byte EncryptFileAES(byte fileData, out byte key, out byte iv)
        {
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.KeySize = 256;
                aesAlg.GenerateKey();
                aesAlg.GenerateIV();

                key = aesAlg.Key;
                iv = aesAlg.IV;

                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    msEncrypt.Write(iv, 0, iv.Length); // Записуємо IV в преамбулу
для дешифрування
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt,
aesAlg.CreateEncryptor(), CryptoStreamMode.Write))
                    {
                        csEncrypt.Write(fileData, 0, fileData.Length);
                        csEncrypt.FlushFinalBlock();
                    }
                    return msEncrypt.ToArray();
                }
            }
        }

        /// <summary>
        /// Симетричне дешифрування масиву даних за стандартом AES-256.
        /// </summary>
        public byte DecryptFileAES(byte encryptedData, byte key)
        {
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = key;
                byte iv = new byte;

                using (MemoryStream msInput = new MemoryStream(encryptedData))
                {
                    msInput.Read(iv, 0, iv.Length); // Зчитуємо IV
                    aesAlg.IV = iv;

                    using (CryptoStream csDecrypt = new CryptoStream(msInput,
aesAlg.CreateDecryptor(), CryptoStreamMode.Read))
                    using (MemoryStream msOutput = new MemoryStream())
                    {
                        csDecrypt.CopyTo(msOutput);
                        return msOutput.ToArray();
                    }
                }
            }
        }
    }
```

```
}
```

Файл Services/BlockchainService.cs

C#

```
using System;
using System.Security.Cryptography;
using System.Text;
using SecureCryptoSystem.Models;

namespace SecureCryptoSystem.Services
{
    /// <summary>
    /// Сервіс управління Блокчейном (BlockchainService).
    /// Призначення: Формування нових вузлів ланцюжка та розрахунок їх
криптографічних зв'язків.
    /// </summary>
    public class BlockchainService
    {
        /// <summary>
        /// Ініціалізація та генерація нового Блоку реєстру.
        /// </summary>
        public BlockchainBlock CreateNewBlock(string dataHash, string
previousHash, int index)
        {
            var block = new BlockchainBlock
            {
                Index = index,
                Timestamp = DateTime.UtcNow,
                DataHash = dataHash,
                PreviousHash = previousHash
            };

            block.Hash = CalculateBlockHash(block);
            return block;
        }

        /// <summary>
        /// Математичне обчислення хешу блоку на основі його конкатенованих
властивостей.
        /// </summary>
        private string CalculateBlockHash(BlockchainBlock block)
        {
            using (SHA256 sha256 = SHA256.Create())
            {
                string rawData = $"{block.Index}-{block.Timestamp:O}-
{block.DataHash}-{block.PreviousHash}";
                byte bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(rawData));

                StringBuilder builder = new StringBuilder();
                foreach (byte b in bytes)
                {
                    builder.Append(b.ToString("x2"));
                }
                return builder.ToString();
            }
        }
    }
}
```

А.4 Шар контролерів (папка `Controllers`)

Файл Controllers/AuthController.cs

C#

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using System.Security.Claims;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using SecureCryptoSystem.Data;
using SecureCryptoSystem.Models;

namespace SecureCryptoSystem.Controllers
{
    /// <summary>
    /// Клас-контролер системи автентифікації та доступу (AuthController).
    /// Призначення: Керування обліковими записами авторів та життєвим циклом
Cookie-сесії.
    /// </summary>
    public class AuthController : Controller
    {
        private readonly AppDbContext _db;
        public AuthController(AppDbContext db) { _db = db; }

        public IActionResult Login() => View();
        public IActionResult Register() => View();

        /// <summary>
        /// Реєстрація нового користувача-автора в базі даних.
        /// </summary>
        [HttpPost]
        public IActionResult Register(string username, string password)
        {
            if (string.IsNullOrWhiteSpace(username) ||
string.IsNullOrWhiteSpace(password))
            {
                TempData = "Помилка: Поля вводу не можуть бути порожніми.";
                return View();
            }

            if (_db.Users.Any(u => u.Username.ToLower() == username.ToLower()))
            {
                TempData = "Помилка: Користувач із таким логіном вже існує.";
                return View();
            }

            var user = new User { Username = username, PasswordHash = password };
            _db.Users.Add(user);
            _db.SaveChanges();

            TempData = "Успіх: Обліковий запис успішно створено. Авторизуйтесь.";
            return RedirectToAction("Login");
        }

        /// <summary>
        /// Авторизація користувача та встановлення Cookie сесії.
        /// </summary>
        [HttpPost]
        public async Task<IActionResult> Login(string username, string password)
        {
            var user = _db.Users.FirstOrDefault(u => u.Username == username &&
u.PasswordHash == password);
            if (user == null)
```

```
            {
                TempData = "Помилка: Невірні облікові дані або пароль.";
                return View();
            }

            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, user.Username),
                new Claim(ClaimTypes.NameIdentifier, user.Id.ToString())
            };

            var identity = new ClaimsIdentity(claims,
CookieAuthenticationDefaults.AuthenticationScheme);
            await
HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new
ClaimsPrincipal(identity));

            return RedirectToAction("Index", "Catalog");
        }

        /// <summary>
        /// Знищення сеансу авторизації.
        /// </summary>
        public async Task<IActionResult> Logout()
        {
            await
HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
            return RedirectToAction("Index", "Catalog");
        }
    }
}
```

Файл Controllers/CatalogController.cs

C#

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Hosting;
using System.IO;
using System.IO.Compression;
using System.Text;
using System.Security.Claims;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System;
using System.Collections.Generic;
using SecureCryptoSystem.Data;
using SecureCryptoSystem.Services;
using SecureCryptoSystem.Models;

namespace SecureCryptoSystem.Controllers
{
    /// <summary>
    /// Клас-контролер "Каталог" (CatalogController).
    /// Призначення: Керування процесами шифрування, дешифрування, збереження та
експертизи медіаактивів.
    /// </summary>
    public class CatalogController : Controller
    {
        private readonly AppDbContext _db;
        private readonly SecurityService _sec;
        private readonly BlockchainService _bc;
```

```
        private readonly IWebHostEnvironment _env;

        public CatalogController(AppDbContext db, SecurityService sec,
BlockchainService bc, IWebHostEnvironment env)
        {
            _db = db;
            _sec = sec;
            _bc = bc;
            _env = env;
        }

        public IActionResult Index()
        {
            if (User.Identity!= null && User.Identity.IsAuthenticated)
            {
                int userId =
int.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier)!);

                // Відображаємо тільки ті файли, що належать поточному
авторизованому користувачу
                var files = _db.MediaFiles.Include(m => m.Author)
                                          .Where(m => m.UserId == userId)
                                          .ToList();
                return View(files);
            }
            return View(new List<MediaFile>());
        }

        /// <summary>
        /// Метод шифрування контенту та додавання транзакції у Блокчейн (Етап 1).
        /// </summary>
        [HttpPost]
        [Authorize]
        public IActionResult Encrypt(IFormFile file)
        {
            if (file == null)
            {
                TempData = "Помилка: Файл не обрано.";
                return RedirectToAction("Index");
            }

            int userId =
int.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier)!);

            string storagePath = Path.Combine(_env.WebRootPath, "SecureStorage");
            string previewPath = Path.Combine(_env.WebRootPath, "Previews");
            if (!Directory.Exists(storagePath))
Directory.CreateDirectory(storagePath);
            if (!Directory.Exists(previewPath))
Directory.CreateDirectory(previewPath);

            using var ms = new MemoryStream();
            file.CopyTo(ms);
            byte originalFileData = ms.ToArray();

            // Пошук маркувальної мітки SIG_ID у заголовку файлу
            if (originalFileData.Length >= 8 &&
Encoding.UTF8.GetString(originalFileData.Take(8).ToArray()) ==
SecurityService.SIG_ID)
            {
                TempData = "Помилка: Файл вже зашифровано (SIG_ID виявлено).";
                return RedirectToAction("Index");
            }
```

```
            string uniqueId = Guid.NewGuid().ToString();
            string previewName = uniqueId + Path.GetExtension(file.FileName);
            string encName = uniqueId + Path.GetExtension(file.FileName) + ".enc";

            // 1. Збереження оригінального зображення у папку Previews для
безпечного рендерингу
            System.IO.File.WriteAllBytes(Path.Combine(previewPath, previewName),
originalFileData);

            // 2. Блокчейн-зв'язування
            var lastBlock = _db.Blocks.OrderByDescending(b =>
b.Id).FirstOrDefault();
            string prevHash = lastBlock!= null? lastBlock.DataHash :
"GENESIS_BLOCK_00000000000000000000000000000000";

            // Вбудовування попереднього гешу у структуру файлу перед шифруванням
            byte payloadWithMetadata =
_sec.EmbedBlockchainMetadata(originalFileData, prevHash);

            // 3. Симетричне шифрування AES-256
            byte encryptedFile = _sec.EncryptFileAES(payloadWithMetadata, out byte
aesKey, out byte aesIv);

            // Впровадження мітки SIG_ID в заголовок контейнера.enc
            byte finalContainer =
Encoding.UTF8.GetBytes(SecurityService.SIG_ID).Concat(encryptedFile).ToArray();

            // 4. Асиметричний захист ключа (RSA-2048)
            var rsaKeys = _sec.GenerateRSAKeys();
            byte encryptedAesKey = _sec.EncryptAESKeyWithRSA(aesKey,
rsaKeys.PublicKey);

            // 5. Запис зашифрованого контейнера у фізичне сховище на сервері
            System.IO.File.WriteAllBytes(Path.Combine(storagePath, encName),
finalContainer);

            // 6. Фіксація транзакції у Блокчейн-реєстрі SQLite
            string currentHash = _sec.ComputeHash(finalContainer);
            _db.MediaFiles.Add(new MediaFile {
                OriginalFileName = file.FileName,
                PreviewFileName = previewName,
                EncryptedFileName = encName,
                Hash = currentHash,
                PreviousHash = prevHash,
                UserId = userId
            });

            _db.Blocks.Add(_bc.CreateNewBlock(currentHash, prevHash,
(lastBlock?.Index?? 0) + 1));
            _db.SaveChanges();

            // 7. Формування та видача ZIP-архіву з ключами користувачу
            using var zipStream = new MemoryStream();
            using (var archive = new ZipArchive(zipStream, ZipArchiveMode.Create,
true))
            {
                var keyEntry = archive.CreateEntry("aes_encrypted.key");
                using (var entryStream = keyEntry.Open())
entryStream.Write(encryptedAesKey);

                var rsaEntry = archive.CreateEntry("private_rsa.pem");
```

```
                using (var entryStream = rsaEntry.Open())
entryStream.Write(Encoding.UTF8.GetBytes(rsaKeys.PrivateKey));
            }
            zipStream.Position = 0;

            TempData = "Фотографію успішно зашифровано та додано до Блокчейну.";
            return File(zipStream.ToArray(), "application/zip", "Keys_" +
file.FileName + ".zip");
        }

        /// <summary>
        /// Метод дешифрування контенту на основі валідації ключів користувача
(Етап 2).
        /// </summary>
        [HttpPost]
        public IActionResult Decrypt(int fileId, IFormFile encKeyFile, IFormFile
privateKeyFile)
        {
            try
            {
                if (encKeyFile == null || privateKeyFile == null)
                {
                    TempData = "Помилка: Для дешифрування необхідні файли.KEY
та.PEM.";
                    return RedirectToAction("Index");
                }

                var mediaFile = _db.MediaFiles.FirstOrDefault(f => f.Id ==
fileId);
                if (mediaFile == null)
                {
                    TempData = "Помилка: Запис не знайдено.";
                    return RedirectToAction("Index");
                }

                string fullPath = Path.Combine(_env.WebRootPath, "SecureStorage",
mediaFile.EncryptedFileName);
                if (!System.IO.File.Exists(fullPath))
                {
                    TempData = "Помилка: Фізичний файл контейнера відсутній на
сервері.";
                    return RedirectToAction("Index");
                }

                byte rawContainer = System.IO.File.ReadAllBytes(fullPath);

                // L1: Аналіз заголовка на наявність сигнатури
                if (Encoding.UTF8.GetString(rawContainer.Take(8).ToArray())!=
SecurityService.SIG_ID)
                    throw new FormatException("L1_ERROR: Системну мітку не
знайдено.");

                // L2: Зняття маркувального заголовка SIG_ID
                byte encryptedFile = rawContainer.Skip(8).ToArray();

                // Зчитування ключів користувача
                using var ms2 = new MemoryStream(); encKeyFile.CopyTo(ms2); byte
encryptedAesKey = ms2.ToArray();
                using var ms3 = new StreamReader(privateKeyFile.OpenReadStream());
string privateRsaXml = ms3.ReadToEnd();

                // L3 & L4: Дешифрування ключів та контенту
```

```
                byte aesKey = _sec.DecryptAESKeyWithRSA(encryptedAesKey,
privateRsaXml);
                byte originalData = _sec.DecryptFileAES(encryptedFile, aesKey);

                string originalName = "Restored_" + mediaFile.OriginalFileName;

                TempData = "Криптографічну автентичність верифіковано Блокчейном.
Файл відновлено.";
                return File(originalData, "application/octet-stream",
originalName);
            }
            catch (Exception ex)
            {
                TempData = $"Помилка верифікації: {ex.Message}";
                return RedirectToAction("Index");
            }
        }

        /// <summary>
        /// Скачування сирого зашифрованого.ENC файлу.
        /// </summary>
        [HttpGet]
        public IActionResult DownloadEncrypted(int fileId)
        {
            var mediaFile = _db.MediaFiles.FirstOrDefault(f => f.Id == fileId);
            if (mediaFile == null) return RedirectToAction("Index");

            string fullPath = Path.Combine(_env.WebRootPath, "SecureStorage",
mediaFile.EncryptedFileName);
            byte fileBytes = System.IO.File.ReadAllBytes(fullPath);

            return File(fileBytes, "application/octet-stream",
mediaFile.EncryptedFileName);
        }

        [HttpGet]
        public IActionResult Compare() => View();

        /// <summary>
        /// Проведення криптографічного аудиту та аналізу ланцюга зв'язності.
        /// </summary>
        [HttpPost]
        public IActionResult Compare(IFormFile file1, IFormFile file2)
        {
            if (file1 == null || file2 == null)
            {
                TempData = "Помилка експертизи: Необхідно надати обидва файли.";
                return View();
            }

            using var ms1 = new MemoryStream(); file1.CopyTo(ms1);
            using var ms2 = new MemoryStream(); file2.CopyTo(ms2);

            string hash1 = _sec.ComputeHash(ms1.ToArray());
            string hash2 = _sec.ComputeHash(ms2.ToArray());

            ViewBag.Hash1 = hash1;
            ViewBag.Hash2 = hash2;
            ViewBag.File1Name = file1.FileName;
            ViewBag.File2Name = file2.FileName;

            bool isExactMatch = (hash1 == hash2);
            bool isBlockchainLink = false;
```

```
            var file2Record = _db.MediaFiles.FirstOrDefault(m => m.Hash == hash2);
            if (file2Record!= null && file2Record.PreviousHash == hash1)
            {
                isBlockchainLink = true;
            }

            ViewBag.IsMatch = isExactMatch;
            ViewBag.IsBlockchainLink = isBlockchainLink;

            return View();
        }
    }
}
```

А.5 Шар представлення (папка `Views`)

Файл Views/Shared/_Layout.cshtml

HTML

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData - CryptoShield</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.11.0/font/bootstrap-icons.css" rel="stylesheet" />
</head>
<body class="bg-light d-flex flex-column min-vh-100">
    <header>
        <nav class="navbar navbar-expand-sm navbar-dark bg-dark border-bottom box-
shadow mb-4 py-3">
            <div class="container">
                <a class="navbar-brand fw-bold text-info" asp-controller="Catalog"
asp-action="Index">
                    <i class="bi bi-shield-shaded"></i> CryptoShield Pro
                </a>
                <div class="navbar-collapse collapse d-sm-inline-flex justify-
content-between">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-white" asp-
controller="Catalog" asp-action="Index">
                                <i class="bi bi-person-workspace me-1"></i> Мій
Каталог
                            </a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-warning fw-bold" asp-
controller="Catalog" asp-action="Compare">
                                <i class="bi bi-cpu me-1"></i> Експертиза Гешів
                            </a>
                        </li>
                    </ul>
                    <ul class="navbar-nav">
                        @if (User.Identity!= null &&
User.Identity.IsAuthenticated)
                        {
                            <li class="nav-item me-3 d-flex align-items-center">
```

```
                                <span class="text-light"><i class="bi bi-person-
circle text-info me-1"></i> <strong>@User.Identity.Name</strong></span>
                            </li>
                            <li class="nav-item">
                                <a class="btn btn-outline-danger btn-sm px-3" asp-
controller="Auth" asp-action="Logout">Вийти</a>
                            </li>
                        }
                        else
                        {
                            <li class="nav-item me-2">
                                <a class="btn btn-outline-light btn-sm px-3" asp-
controller="Auth" asp-action="Login">Увійти</a>
                            </li>
                            <li class="nav-item">
                                <a class="btn btn-info btn-sm px-3 fw-bold text-
dark" asp-controller="Auth" asp-action="Register">Реєстрація</a>
                            </li>
                        }
                    </ul>
                </div>
            </div>
        </nav>
    </header>

    <main role="main" class="container flex-grow-1">
        @RenderBody()
    </main>

    <footer class="border-top footer text-muted bg-white py-4 mt-5 shadow-sm">
        <div class="container text-center">
            &copy; 2026 - <span class="fw-bold text-dark">CryptoShield Pro</span>
— Дипломний проєкт.
        </div>
    </footer>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
></script>
</body>
</html>
```

Файл Views/Catalog/Index.cshtml

HTML

```
@model IEnumerable<CryptoWebProject.Models.MediaFile>
@{
    ViewData = "Каталог";
}

<div class="container py-4">
    <div class="d-flex justify-content-between align-items-center mb-5">
        <div>
            <h1 class="display-6 fw-bold text-dark"><i class="bi bi-shield-lock-
fill text-primary"></i> CryptoShield Portfolio</h1>
            <p class="text-muted mb-0">Захищене сховище ваших фотографій у вигляді
інтерактивної галереї.</p>
        </div>
    </div>

    @if (TempData!= null) {
        <div class="alert alert-success shadow-sm rounded-3"><i class="bi bi-
check-circle-fill me-2"></i> @TempData</div>
    }
```

```
    @if (TempData!= null) {
        <div class="alert alert-warning shadow-sm rounded-3"><i class="bi bi-
exclamation-triangle-fill me-2"></i> @TempData</div>
    }

    @if(!User.Identity!.IsAuthenticated) {
        <div class="alert alert-info border-0 shadow-sm rounded-4 p-4 text-center
mb-5">
            <h5 class="fw-bold"><i class="bi bi-lock-fill"></i> Доступ
обмежено</h5>
            <p class="mb-3">Для роботи з вашим каталогом необхідно увійти або
зареєструватись у системі.</p>
            <a asp-controller="Auth" asp-action="Login" class="btn btn-primary px-
4 me-2">Увійти</a>
            <a asp-controller="Auth" asp-action="Register" class="btn btn-outline-
primary px-4">Реєстрація</a>
        </div>
    }
    else {
        <div class="card shadow border-0 mb-5 rounded-4">
            <div class="card-header bg-primary text-white py-3">
                <h5 class="mb-0 fw-bold"><i class="bi bi-cloud-arrow-up-fill me-
2"></i> Зашифрувати нове фото</h5>
            </div>
            <div class="card-body p-4 bg-light">
                <form asp-action="Encrypt" id="encryptForm"
enctype="multipart/form-data" method="post" class="row g-3 align-items-center">
                    <div class="col-md-9">
                        <div class="input-group input-group-lg shadow-sm">
                            <label class="btn btn-outline-primary fw-bold"
for="mainFileInput"><i class="bi bi-folder2-open me-1"></i> Огляд...</label>
                            <input type="file" name="file" id="mainFileInput"
class="d-none" required onchange="document.getElementById('mainFileText').value =
this.files.name" />
                            <input type="text" id="mainFileText" class="form-
control bg-white text-muted" placeholder="Файл не обрано" readonly
onclick="document.getElementById('mainFileInput').click()" style="cursor:
pointer;" />
                        </div>
                    </div>
                    <div class="col-md-3 d-grid">
                        <button type="submit" id="encryptBtn" class="btn btn-
primary btn-lg shadow-sm fw-bold"><i class="bi bi-lock-fill me-1"></i>
Зашифрувати</button>
                    </div>
                </form>
            </div>
        </div>

        <h4 class="fw-bold mb-4 text-dark"><i class="bi bi-images text-primary me-
2"></i> Мій каталог</h4>

        @if (!Model.Any()) {
            <div class="text-center p-5 bg-white rounded-4 shadow-sm border">
                <i class="bi bi-folder-x display-1 text-muted opacity-25"></i>
                <p class="lead mt-3 text-muted">Ваш каталог порожній. Завантажте
перше фото.</p>
            </div>
        }
        else {
            <div class="row row-cols-1 row-cols-md-2 row-cols-lg-3 g-4">
                @foreach (var item in Model)
                {
```

```
                    <div class="col">
                        <div class="card h-100 border-0 shadow-sm rounded-4
overflow-hidden">

                            <div class="position-relative bg-dark" style="height:
250px; overflow: hidden; user-select: none;">
                                <canvas id="canvas-@item.Id" class="w-100 h-100
object-fit-cover" style="pointer-events: none;"></canvas>
                                <div class="position-absolute top-0 start-0 w-100
h-100" style="z-index: 10; cursor: not-allowed;" oncontextmenu="return
false;"></div>
                            </div>

                            <div class="card-body bg-white p-4">
                                <h6 class="fw-bold text-truncate mb-3 text-
dark">@item.OriginalFileName</h6>

                                <a asp-action="DownloadEncrypted" asp-route-
fileId="@item.Id" class="btn btn-outline-primary btn-sm w-100 fw-bold mb-3">
                                    <i class="bi bi-download me-1"></i>
Скачати.ENC файл
                                </a>

                                <div class="bg-light p-3 rounded-3 border">
                                    <small class="fw-bold text-dark d-block mb-2
text-center small"><i class="bi bi-unlock-fill text-success"></i>
Розшифрування</small>
                                    <form asp-action="Decrypt"
enctype="multipart/form-data" method="post">
                                        <input type="hidden" name="fileId"
value="@item.Id" />

                                        <div class="input-group input-group-sm mb-
2 shadow-sm">
                                            <span class="input-group-text bg-white
fw-bold text-muted" style="width: 60px;">.KEY</span>
                                            <label class="btn btn-outline-
secondary fw-bold" for="key-@item.Id">Огляд</label>
                                            <input type="file" name="encKeyFile"
id="key-@item.Id" class="d-none" required onchange="document.getElementById('text-
key-@item.Id').value = this.files.name" />
                                            <input type="text" id="text-key-
@item.Id" class="form-control bg-white" placeholder="Не обрано" readonly
onclick="document.getElementById('key-@item.Id').click()" style="cursor: pointer;"
/>
                                        </div>

                                        <div class="input-group input-group-sm mb-
3 shadow-sm">
                                            <span class="input-group-text bg-white
fw-bold text-muted" style="width: 60px;">.PEM</span>
                                            <label class="btn btn-outline-
secondary fw-bold" for="pem-@item.Id">Огляд</label>
                                            <input type="file"
name="privateKeyFile" id="pem-@item.Id" class="d-none" required
onchange="document.getElementById('text-pem-@item.Id').value = this.files.name" />
                                            <input type="text" id="text-pem-
@item.Id" class="form-control bg-white" placeholder="Не обрано" readonly
onclick="document.getElementById('pem-@item.Id').click()" style="cursor: pointer;"
/>
                                        </div>
```

```
                                        <button type="submit" class="btn btn-
success btn-sm w-100 fw-bold shadow-sm">
                                            Розшифрувати
                                        </button>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>

                    <script>
                        document.addEventListener("DOMContentLoaded", function() {
                            const canvas = document.getElementById('canvas-
@item.Id');
                            if(canvas) {
                                const ctx = canvas.getContext('2d');
                                const img = new Image();
                                img.src = '/Previews/@item.PreviewFileName';
                                img.onload = function() {
                                    canvas.width = img.width;
                                    canvas.height = img.height;
                                    ctx.drawImage(img, 0, 0);

                                    ctx.fillStyle = "rgba(0, 0, 0, 0.7)";
                                    ctx.fillRect(0, canvas.height - 60,
canvas.width, 60);

                                    ctx.font = "bold 13px monospace";
                                    ctx.fillStyle = "#00FF00";
                                    ctx.textAlign = "left";
                                    ctx.fillText("CURR HASH:
@item.Hash.Substring(0, 40)...", 10, canvas.height - 35);

                                    ctx.fillStyle = "#00CCFF";
                                    ctx.fillText("PREV HASH:
@item.PreviousHash.Substring(0, 40)...", 10, canvas.height - 15);

                                    ctx.font = "bold 24px Arial";
                                    ctx.fillStyle = "rgba(255, 255, 255, 0.2)";
                                    ctx.translate(canvas.width / 2, canvas.height
/ 2);
                                    ctx.rotate(-Math.PI / 6);
                                    ctx.textAlign = "center";
                                    ctx.fillText("© @item.Author?.Username", 0,
0);
                                };
                            }
                        });
                    </script>
                }
            </div>
        }
    }
</div>

<script>
    document.addEventListener("DOMContentLoaded", function() {
        const encryptForm = document.getElementById('encryptForm');
        if(encryptForm) {
            encryptForm.addEventListener('submit', async function (e) {
                e.preventDefault();
                const btn = document.getElementById('encryptBtn');
                const originalHtml = btn.innerHTML;
```

```
                btn.innerHTML = '<span class="spinner-border spinner-border-sm me-
2" role="status" aria-hidden="true"></span>Обробка...';
                btn.disabled = true;
                try {
                    const response = await fetch(this.action, { method: 'POST',
body: new FormData(this) });
                    if (response.ok) {
                        const blob = await response.blob();
                        let filename = "Keys.zip";
                        const disposition = response.headers.get('Content-
Disposition');
                        if (disposition && disposition.includes('attachment')) {
                            const matches =
/filename[^;=\n]*=((['"]).*?\2|[^;\n]*)/.exec(disposition);
                            if (matches!= null && matches) {
                                filename = matches.replace(/['"]/g, '');
                            }
                        }
                        const downloadUrl = window.URL.createObjectURL(blob);
                        const a = document.createElement('a');
                        a.style.display = 'none';
                        a.href = downloadUrl;
                        a.download = decodeURIComponent(filename);
                        document.body.appendChild(a);
                        a.click();
                        window.URL.revokeObjectURL(downloadUrl);
                        window.location.reload();
                    } else { throw new Error('Помилка сервера'); }
                } catch (error) {
                    alert('Виникла помилка під час обробки.');
                    btn.innerHTML = originalHtml;
                    btn.disabled = false;
                }
            });
        }
    });
</script>
```

Файл Views/Catalog/Compare.cshtml

HTML

```
@{
    ViewData = "Експертиза Гешів";
}

<div class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-10">
            <div class="text-center mb-5">
                <h2 class="display-6 fw-bold text-warning"><i class="bi bi-
cpu"></i> Експертиза Гешів та Ланцюга</h2>
                <p class="text-muted">Модуль перевірки криптографічних зв'язків
між транзакціями.</p>
            </div>
            @if (TempData!= null) {
                <div class="alert alert-warning shadow-sm"><i class="bi bi-
exclamation-triangle-fill me-2"></i> @TempData</div>
            }
            <div class="card shadow border-0 rounded-4 mb-5">
                <div class="card-header bg-dark text-white py-3">
                    <h5 class="mb-0 fw-bold"><i class="bi bi-file-earmark-binary
me-2"></i> Завантаження файлів.enc для аналізу</h5>
                </div>
```

```
                <div class="card-body p-4 bg-light">
                    <form asp-action="Compare" enctype="multipart/form-data"
method="post">
                        <div class="row g-4">
                            <div class="col-md-6">
                                <label class="fw-bold text-primary mb-2">Перший
файл (Попередній):</label>
                                <div class="input-group shadow-sm">
                                    <label class="btn btn-outline-primary fw-bold"
for="file1">Огляд...</label>
                                    <input type="file" name="file1" id="file1"
class="d-none" required onchange="document.getElementById('text-file1').value =
this.files.name" />
                                    <input type="text" id="text-file1"
class="form-control bg-white" placeholder="Оберіть файл.enc" readonly
onclick="document.getElementById('file1').click()" style="cursor: pointer;" />
                                </div>
                            </div>
                            <div class="col-md-6">
                                <label class="fw-bold text-danger mb-2">Другий
файл (Наступний):</label>
                                <div class="input-group shadow-sm">
                                    <label class="btn btn-outline-danger fw-bold"
for="file2">Огляд...</label>
                                    <input type="file" name="file2" id="file2"
class="d-none" required onchange="document.getElementById('text-file2').value =
this.files.name" />
                                    <input type="text" id="text-file2"
class="form-control bg-white" placeholder="Оберіть файл.enc" readonly
onclick="document.getElementById('file2').click()" style="cursor: pointer;" />
                                </div>
                            </div>
                        </div>
                        <div class="text-center mt-4">
                            <button type="submit" class="btn btn-warning btn-lg
fw-bold shadow px-5">Порівняти</button>
                        </div>
                    </form>
                </div>
            </div>
            @if (ViewBag.Hash1!= null && ViewBag.Hash2!= null)
            {
                <h4 class="fw-bold mb-4 text-center">Результат математичного
аналізу</h4>
                <div class="row g-4 mb-4">
                    <div class="col-md-6">
                        <div class="bg-white p-4 rounded-4 shadow-sm border h-
100">
                            <h6 class="fw-bold text-
primary">@ViewBag.File1Name</h6>
                            <hr>
                            <small class="text-muted fw-bold d-block mb-2">ГЕШ
ПЕРШОГО ФАЙЛУ:</small>
                            <code class="d-block bg-light p-3 rounded border text-
break font-monospace text-dark fs-6">@ViewBag.Hash1</code>
                        </div>
                    </div>
                    <div class="col-md-6">
                        <div class="bg-white p-4 rounded-4 shadow-sm border h-
100">
                            <h6 class="fw-bold text-
danger">@ViewBag.File2Name</h6>
                            <hr>
```

```
                            <small class="text-muted fw-bold d-block mb-2">ГЕШ
ДРУГОГО ФАЙЛУ:</small>
                            <code class="d-block bg-light p-3 rounded border text-
break font-monospace text-dark fs-6">@ViewBag.Hash2</code>
                        </div>
                    </div>
                </div>
                @if (ViewBag.IsMatch)
                {
                    <div class="alert alert-success shadow text-center p-4
rounded-4 border-0">
                        <h4 class="fw-bold">Файли АБСОЛЮТНО ІДЕНТИЧНІ</h4>
                    </div>
                }
                else if (ViewBag.IsBlockchainLink)
                {
                    <div class="alert alert-info shadow text-center p-4 rounded-4
border-0" style="background-color: #e0f2fe;">
                        <h4 class="fw-bold text-primary">БЛОКЧЕЙН-ЛАНЦЮГ
ПІДТВЕРДЖЕНО!</h4>
                        <p class="mb-0 text-dark">Файл
<strong>@ViewBag.File2Name</strong> є послідовним криптографічним нащадком файлу
<strong>@ViewBag.File1Name</strong>.</p>
                    </div>
                }
                else
                {
                    <div class="alert alert-danger shadow text-center p-4 rounded-
4 border-0">
                        <h4 class="fw-bold">Файли РІЗНІ ТА НЕ ПОВ'ЯЗАНІ</h4>
                    </div>
                }
            }
        </div>
    </div>
</div>
```

Файл Views/Auth/Login.cshtml

HTML

```
@{ ViewData = "Вхід"; }
<div class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-5">
            <div class="card shadow border-0 rounded-4 p-5">
                <h3 class="fw-bold text-center mb-4">Вхід до системи</h3>
                @if (TempData!= null) { <div class="alert alert-
warning">@TempData</div> }
                @if (TempData!= null) { <div class="alert alert-
success">@TempData</div> }
                <form asp-action="Login" method="post">
                    <div class="mb-3">
                        <label class="form-label text-muted fw-bold
small">ЛОГІН</label>
                        <input type="text" name="username" class="form-control
form-control-lg bg-light" required />
                    </div>
                    <div class="mb-4">
                        <label class="form-label text-muted fw-bold
small">ПАРОЛЬ</label>
                        <input type="password" name="password" class="form-control
form-control-lg bg-light" required />
                    </div>
```

```
                    <button type="submit" class="btn btn-primary btn-lg w-100 fw-
bold shadow-sm">Увійти</button>
                </form>
                <div class="text-center mt-4">
                    <a asp-action="Register" class="text-decoration-none text-
muted">Створити новий акаунт</a>
                </div>
            </div>
        </div>
    </div>
</div>
```

Файл Views/Auth/Register.cshtml

HTML

```
@{ ViewData = "Реєстрація"; }
<div class="container py-5">
    <div class="row justify-content-center">
        <div class="col-md-5">
            <div class="card shadow border-0 rounded-4 p-5">
                <h3 class="fw-bold text-center mb-4">Реєстрація</h3>
                @if (TempData!= null) { <div class="alert alert-
warning">@TempData</div> }
                <form asp-action="Register" method="post">
                    <div class="mb-3">
                        <label class="form-label text-muted fw-bold small">ОБЕРІТЬ
ЛОГІН</label>
                        <input type="text" name="username" class="form-control
form-control-lg bg-light" required />
                    </div>
                    <div class="mb-4">
                        <label class="form-label text-muted fw-bold
small">ПАРОЛЬ</label>
                        <input type="password" name="password" class="form-control
form-control-lg bg-light" required />
                    </div>
                    <button type="submit" class="btn btn-success btn-lg w-100 fw-
bold shadow-sm">Зареєструватись</button>
                </form>
                <div class="text-center mt-4">
                    <a asp-action="Login" class="text-decoration-none text-
muted">Вже є акаунт? Увійти</a>
                </div>
            </div>
        </div>
    </div>
</div>
```

ДОДАТОК А. ПОВНИЙ ЛІСТИНГ КЛАСІВ ТА ФАЙЛІВ СЕРВЕРНОЇ ЧАСТИНИ

Нижче наведено лістинг криптографічних модулів та шару доступу до бази даних (SQLite) у суворій послідовності їх розташування у проєкті Visual Studio.

1. Шар моделей даних (`Models/`)

Файл Models/User.cs

C#

```
using System.ComponentModel.DataAnnotations;

namespace CryptoWebProject.Models
```

```
{
    public class User
    {
        [Key]
        public int Id { get; set; }


        public string Username { get; set; } = string.Empty;


        public string PasswordHash { get; set; } = string.Empty;
    }
}
```

Файл Models/MediaFile.cs

C#

```
using System;
using System.ComponentModel.DataAnnotations;

namespace CryptoWebProject.Models
{
    public class MediaFile
    {
        [Key]
        public int Id { get; set; }

        public string OriginalFileName { get; set; } = string.Empty;
        public string PreviewFileName { get; set; } = string.Empty;
        public string EncryptedFileName { get; set; } = string.Empty;

        public string Hash { get; set; } = string.Empty;
        public string PreviousHash { get; set; } = string.Empty;

        public int UserId { get; set; }
        public User? Author { get; set; }

        public DateTime UploadDate { get; set; } = DateTime.UtcNow;
    }
}
```

Файл Models/BlockchainBlock.cs

C#

```
using System;
using System.ComponentModel.DataAnnotations;

namespace CryptoWebProject.Models
{
    public class BlockchainBlock
    {
        [Key]
        public int Id { get; set; }

        public int Index { get; set; }
        public DateTime Timestamp { get; set; }
        public string DataHash { get; set; } = string.Empty;
        public string PreviousHash { get; set; } = string.Empty;
        public string Hash { get; set; } = string.Empty;
    }
}
```

Файл Models/ErrorViewModel.cs

C#

```
namespace CryptoWebProject.Models
{
    public class ErrorViewModel
    {
        public string? RequestId { get; set; }
        public bool ShowRequestId =>!string.IsNullOrEmpty(RequestId);
    }
}
```

2. Контекст Бази Даних (`Data/`)

Файл `Data/AppDbContext.cs`

C#

```
using Microsoft.EntityFrameworkCore;
using CryptoWebProject.Models;

namespace CryptoWebProject.Data
{
    public class AppDbContext : DbContext
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

        public DbSet<User> Users { get; set; }
        public DbSet<MediaFile> MediaFiles { get; set; }
        public DbSet<BlockchainBlock> Blocks { get; set; }
    }
}
```

3. Криптографічні сервіси (`Services/`)

Файл Services/SecurityService.cs

C#

```
using System;
using System.Security.Cryptography;
using System.Text;
using System.IO;

namespace CryptoWebProject.Services
{
    public class SecurityService
    {
        public const string SIG_ID = "CRYPTPRO";

        public string ComputeHash(byte rawData)
        {
            using (SHA256 sha256Hash = SHA256.Create())
            {
                byte bytes = sha256Hash.ComputeHash(rawData);
                StringBuilder builder = new StringBuilder();
                for (int i = 0; i < bytes.Length; i++)
                    builder.Append(bytes[i].ToString("x2"));
                return builder.ToString();
            }
        }
```

```
        public byte EmbedBlockchainMetadata(byte originalData, string
previousHash)
        {
            byte metadataBytes = Encoding.UTF8.GetBytes($"");
            byte combinedPayload = new byte;

            Buffer.BlockCopy(originalData, 0, combinedPayload, 0,
originalData.Length);
            Buffer.BlockCopy(metadataBytes, 0, combinedPayload,
originalData.Length, metadataBytes.Length);

            return combinedPayload;
        }

        public (string PublicKey, string PrivateKey) GenerateRSAKeys()
        {
            using (RSA rsa = RSA.Create(2048))
            {
                return (rsa.ToXmlString(false), rsa.ToXmlString(true));
            }
        }

        public byte EncryptAESKeyWithRSA(byte aesKey, string publicKeyXml)
        {
            using (RSA rsa = RSA.Create())
            {
                rsa.FromXmlString(publicKeyXml);
                return rsa.Encrypt(aesKey, RSAEncryptionPadding.Pkcs1);
            }
        }

        public byte DecryptAESKeyWithRSA(byte encryptedAesKey, string
privateKeyXml)
        {
            using (RSA rsa = RSA.Create())
            {
                rsa.FromXmlString(privateKeyXml);
                return rsa.Decrypt(encryptedAesKey, RSAEncryptionPadding.Pkcs1);
            }
        }

        public byte EncryptFileAES(byte fileData, out byte key, out byte iv)
        {
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.KeySize = 256;
                aesAlg.GenerateKey();
                aesAlg.GenerateIV();

                key = aesAlg.Key;
                iv = aesAlg.IV;

                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    msEncrypt.Write(iv, 0, iv.Length);
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt,
aesAlg.CreateEncryptor(), CryptoStreamMode.Write))
                    {
                        csEncrypt.Write(fileData, 0, fileData.Length);
                        csEncrypt.FlushFinalBlock();
                    }
                    return msEncrypt.ToArray();
                }
```

```
            }
        }

        public byte DecryptFileAES(byte encryptedData, byte key)
        {
            using (Aes aesAlg = Aes.Create())
            {
                aesAlg.Key = key;
                byte iv = new byte;

                using (MemoryStream msInput = new MemoryStream(encryptedData))
                {
                    msInput.Read(iv, 0, iv.Length);
                    aesAlg.IV = iv;

                    using (CryptoStream csDecrypt = new CryptoStream(msInput,
aesAlg.CreateDecryptor(), CryptoStreamMode.Read))
                    using (MemoryStream msOutput = new MemoryStream())
                    {
                        csDecrypt.CopyTo(msOutput);
                        return msOutput.ToArray();
                    }
                }
            }
        }
    }
}
```

Файл Services/BlockchainService.cs

C#

```
using System;
using System.Security.Cryptography;
using System.Text;
using CryptoWebProject.Models;

namespace CryptoWebProject.Services
{
    public class BlockchainService
    {
        public BlockchainBlock CreateNewBlock(string dataHash, string
previousHash, int index)
        {
            var block = new BlockchainBlock
            {
                Index = index,
                Timestamp = DateTime.UtcNow,
                DataHash = dataHash,
                PreviousHash = previousHash
            };

            block.Hash = CalculateBlockHash(block);
            return block;
        }

        private string CalculateBlockHash(BlockchainBlock block)
        {
            using (SHA256 sha256 = SHA256.Create())
            {
                string rawData = $"{block.Index}-{block.Timestamp:O}-
{block.DataHash}-{block.PreviousHash}";
                byte bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(rawData));
```

```
                StringBuilder builder = new StringBuilder();
                foreach (byte b in bytes)
                {
                    builder.Append(b.ToString("x2"));
                }
                return builder.ToString();
            }
        }
    }
}
```

Метадані

ДОКУМЕНТ

Заголовок

2026 КвРбак - Руденко

Автор	Науковий керівник / Експерт	ІД документу
Руденко Ольга Володимирівна	**С.М. Коновалов**	**334318780**

ОРГАНІЗАЦІЯ

Назва організації	підрозділ
Odesa National Maritime University	**Технічна кібернетика й інформаційні технології ім. професора Р.В. Мерктa**

ЗВІТ

Дата звіту	Дата редагування
6/15/2026	**---**

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1	КП 2	КЦ
7.37% (7.37%)	0.65% (0.65%)	8.6% (8.6%)

25	18059	157977
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	0
Інтервали	A→	0
Мікропробіли	·	10
Білі знаки	ß	0
Парафрази (SmartMarks)	a	92

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

1	2026 КвРбак - Запорожан [redacted] 6/12/2026 Odesa National Maritime University (Технічна кібернетика й інформаційні технології ім. професора Р.В. Меркта)	35 (0.19 %)
2	2026 КвРбак - Запорожан [redacted] 6/12/2026 Odesa National Maritime University (Технічна кібернетика й інформаційні технології ім. професора Р.В. Меркта)	30 (0.17 %)
3	https://learn.microsoft.com/en-us/answers/questions/1192579/vs-code-displayed-on-webpage-as-text-rather-than-p	27 (0.15 %)
4	https://ccprograming.blogspot.com/	25 (0.14 %)
5	https://ru.stackoverflow.com/questions/604547/%D0%94%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%B2-%D0%91%D0%94-%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D1%89%D0%B8%D1%85%D1%81%D1%8F-%D0%B2-%D0%BE%D0%B4%D0%BD%D0%BE%D0%BC-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B5	24 (0.13 %)
6	https://duikt.edu.ua/repozitorii/%D0%9A%D0%B0%D1%84%D0%B5%D0%B4%D1%80%D0%B0%20%D0%86%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D1%96%D1%97%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F/2025/%D0%9A%D0%B2%D0%B0%D0%BB%D1%96%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0%20%D0%B1%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80%D0%B0%20%D0%9A%D0%BE%D1%80%D0%BB%D1%8F%D0%BA%D0%BE%D0%B2%D0%B0%20%D0%94.%D0%A1.%202025%20%D1%80%D1%96%D0%BA.pdf	22 (0.12 %)
7	https://github.com/indiser/The-Fake-Shop/compare/indiser:fa97b6f...indiser:7b6e7ec.diff	22 (0.12 %)
8	ФКНТ_125_2025_1_МарущакАВ [redacted] 12/11/2025 Ukrainian national aviation university (ФКНТ Кафедра кібербезпеки)	17 (0.09 %)
9	20191027 – Яна Цветкова.pdf [redacted] 10/27/2019 Sofia University St. Kliment Ohridski (Deanary)	17 (0.09 %)
10	https://stackoverflow.com/questions/68937613/ef-core-many-to-many-relation-table-naming	17 (0.09 %)

Домашня база даних (0.36 %)

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	2026 КвРбак - Запорожан [redacted] 6/12/2026 **Odesa National Maritime University (Технічна кібернетика й інформаційні технології ім. професора Р.В. Меркта)**	**65 (2) (0.36 %)**

Програма обміну базами даних (2.77 %)

#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

2	Гоменюк КІ-22-2 дипломна++ [redacted] 6/8/2026 Ivano-Frankivsk National Technical University of Oil and Gas (Каф. КСМ)	123 (21) (0.68 %)
3	20191027 – Яна Цветкова.pdf [redacted] 10/27/2019 Sofia University St. Kliment Ohridski (Deanary)	76 (7) (0.42 %)
4	БР_КІ-22-2_Маренков [redacted] 6/9/2026 Ivano-Frankivsk National Technical University of Oil and Gas (Каф. КСМ)	63 (8) (0.35 %)
5	ФКНТ_125_2025_1_МарущакАВ [redacted] 12/11/2025 Ukrainian national aviation university (ФКНТ Кафедра кібербезпеки)	50 (5) (0.28 %)
6	Спосіб сегментації об'єктів шляхом сегментації усіх екземплярів та їх керованого вибору [redacted] 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	41 (4) (0.23 %)
7	БПЗ_ІПЗ_Суховій_Курсова робота.docx [redacted] 5/14/2024 National University Kyiv Mohyla Academy course papers (National University Kyiv Mohyla Academy)	30 (5) (0.17 %)
8	веб-додаток для закладів харчування на базі Flask та PostgreSQL.docx [redacted] 6/7/2024 East Ukrainian National University named after Volodymyr Dahl (East Ukrainian National University named after Volodymyr Dahl)	29 (5) (0.16 %)
9	ВКР_Андрух [redacted] 5/30/2025 State University of Trade and Economics (Кафедра комп'ютерних наук та інформаційних систем)	29 (3) (0.16 %)
10	Процук Р.КН.2025.КР [redacted] 5/27/2025 Boyarka Professional College of National University of Life and Environmental Sciences of Ukraine (Boyarka Professional College of National University of Life and Environmental Sciences of Ukraine)	16 (2) (0.09 %)
11	2024_406_ІПЗ_Крищенко [redacted] 7/11/2024 Ukrainian national aviation university (Ukrainian national aviation university)	14 (2) (0.08 %)
12	Магістерська робота.docx [redacted] 12/8/2025 National University of Water and Environmental Engineering (National University of Water and Environmental Engineering)	10 (1) (0.06 %)
13	Цікаленко М.О. КН.2025 КР [redacted] 5/25/2025 Boyarka Professional College of National University of Life and Environmental Sciences of Ukraine (Boyarka Professional College of National University of Life and Environmental Sciences of Ukraine)	9 (1) (0.05 %)
14	2023_61720000_Kupets_Dmytro_Oleksandrovych_197458 [redacted] 11/21/2024 National University "Lviv Politechnika" (National University Lviv Politechnika)	5 (1) (0.03 %)
15	Розробка алгоритму вбудовування інформації у зображення із використанням машинного навчання [redacted] 10/13/2020 Odessa National Polytechnic University (ІІБРТ, Каф. кібербезпеки та програмного забезпечення)	5 (1) (0.03 %)

Інтернет (4.24 %)

#	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

16	https://duikt.edu.ua/repozitorii/%D0%9A%D0%B0%D1%84%D0%B5%D0%B4%D1%80%D0%B0%20%D0%86%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D1%96%D1%97%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%20%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F/2025/%D0%9A%D0%B2%D0%B0%D0%BB%D1%96%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0%20%D1%80%D0%BE%D0%B1%D0%BE%D1%82%D0%B0%20%D0%B1%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80%D0%B0%20%D0%9A%D0%BE%D1%80%D0%BB%D1%8F%D0%BA%D0%BE%D0%B2%D0%B0%20%D0%94.%D0%A1.%202025%20%D1%80%D1%96%D0%BA.pdf	164 (17) (0.91 %)
17	https://library.kre.dp.ua/Books/2-4%20kurs/%D0%9E%D1%85%D0%BE%D1%80%D0%BE%D0%BD%D0%B0%20%D0%BF%D1%80%D0%B0%D1%86%D1%96%20%D0%B2%20%D0%B3%D0%B0%D0%BB%D1%83%D0%B7%D1%96/%D0%9E%D0%BF%D0%BE%D1%80%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82_%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D0%B9_%D0%B7_%D0%B4%D0%B8%D1%81%D1%86._%D0%9E%D1%85%D0%BE%D1%80%D0%BE%D0%BD%D0%B0_%D0%BF%D1%80%D0%B0%D1%86%D1%96_%D0%B2_%D0%B3%D0%B0%D0%BB%D1%83%D0%B7%D1%96_%D0%9A%D0%BE%D0%BC%D0%BF%E2%80%99%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%96%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8%20%D1%82%D0%B0%20%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D1%96.pdf	103 (11) (0.57 %)
18	https://ir.nmu.org.ua/jspui/bitstream/123456789/164850/1/23_06_16_%D0%9A%D1%83%D1%82%D0%BD%D0%B8%D1%87%20%D0%A2.%D0%A2.pdf	71 (9) (0.39 %)
19	https://github.com/indiser/The-Fake-Shop/compare/indiser:fa97b6f...indiser:7b6e7ec.diff	54 (4) (0.3 %)
20	**https://ccprograming.blogspot.com/**	**45 (4) (0.25 %)**
21	**https://learn.microsoft.com/en-us/answers/questions/1192579/vs-code-displayed-on-webpage-as-text-rather-than-p**	**41 (2) (0.23 %)**
22	https://stackoverflow.com/questions/68937613/ef-core-many-to-many-relation-table-naming	34 (3) (0.19 %)
23	http://lib.pnu.edu.ua:8080/bitstream/123456789/4004/1/C%23_conspect3.pdf	33 (5) (0.18 %)
24	https://dspace.znu.edu.ua/xmlui/bitstream/handle/12345/24414/%D0%90%D0%B1%D0%B4%D1%83%D0%BB%D0%BB%D0%B0%D1%94%D0%B2%D0%B0.pdf?sequence=1	25 (2) (0.14 %)
25	https://ru.stackoverflow.com/questions/604547/%D0%94%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-%D0%B2-%D0%91%D0%94-%D1%81%D0%BE%D0%B4%D0%B5%D1%80%D0%B6%D0%B0%D1%89%D0%B8%D1%85%D1%81%D1%8F-%D0%B2-%D0%BE%D0%B4%D0%BD%D0%BE%D0%BC-%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%B5	24 (1) (0.13 %)
26	https://ir.nmu.org.ua/bitstream/handle/123456789/164857/23_06_14_121-19-1%20%D0%9A%D0%BE%D1%87%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%9C.%20%D0%92..pdf?sequence=1	24 (2) (0.13 %)
27	https://ela.kpi.ua/bitstreams/d593fd51-345c-47f6-94de-8b7e627cb98f/download	21 (2) (0.12 %)
28	https://studfile.net/preview/5093161/page:4/	17 (2) (0.09 %)
29	https://www.c-sharpcorner.com/article/input-validation-and-sanitization-in-asp-net-core-end-to-end-example/	16 (2) (0.09 %)
30	https://studfile.net/preview/5163128/page:4/	16 (1) (0.09 %)
31	https://entityframeworkcore.com/knowledge-base/51311008/entity-framework-core-assign-foreign-key-constraint-in-onmodelcreating-	16 (1) (0.09 %)
32	http://repository.ukd.edu.ua:8080/bitstream/handle/123456789/801/%D0%91%D0%B0%D1%82%D	15 (2) (0.08 %)

	1%96%D0%B3%20%D0%92.pdf?sequence=1	
33	http://moodle2.snu.edu.ua/mod/resource/view.php?id=51797	13 (1) (0.07 %)
34	https://theses.oa.edu.ua/DATA/12781/%D0%A2%D0%B8%D1%89%D1%83%D0%BA_%D0%BA%D0%B2%D0%B0%D0%BB%D1%96%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D0%9A%D0%9D-4.pdf	12 (1) (0.07 %)
35	https://www.c-sharpcorner.com/article/building-a-web-api-with-c-sharp-records-for-dtos/	9 (1) (0.05 %)
36	https://essuir.sumdu.edu.ua/bitstream/123456789/85077/1/Kaliukina_Bachelor_thesis.pdf	7 (1) (0.04 %)
37	http://repo.darmajaya.ac.id/797/8/Lampiran%20%28Coding%20program%29.pdf	6 (1) (0.03 %)

Список прийнятих фрагментів

#	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)