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

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації

Виконала: студентка 2(4) курсу, гр.2КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Скидан Т.М
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. МБІС
Грицак А.В.
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Хомченко І.В.
(прізвище та ініціали)
«____» _____ 2025 р.

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

Вінниця ВНТУ – 2025

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

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

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

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

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Скидан Тетяни Миколаївни
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації
Керівник роботи Грицак Анатолій Васильович, к.т.н., доцент кафедри МБІС
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи 11.06.2025
3. Вихідні дані до роботи: система управління базами даних SQLite, мова програмування Python, середовище розробки Visual Studio Code.
4. Зміст текстової частини: вступ, аналіз сучасних засобів зберігання та захисту корпоративних даних, актуальність проблеми безпечного зберігання даних у корпоративному середовищі, огляд сучасних підходів до зберігання даних у централізованих та децентралізованих системах, аналіз існуючих методів автентифікації, аналіз існуючих децентралізованих систем для зберігання та управління корпоративними даними, проєктування децентралізованої системи для зберігання даних, розробка архітектури децентралізованої системи для зберігання даних, розробка механізмів конфіденційності та автентифікації з використанням Zero-Knowledge Proof та MFA, обґрунтування інтеграції IPFS в децентралізовану систему, обґрунтування вибору засобів програмування, програмна реалізація системи для децентралізованого зберігання корпоративних даних, програмна реалізація впровадження IPFS у систему, інструкція користувача по роботі з розробленою системою, висновки, перелік посилань, додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язк
креслень)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Грицак Анатолій Васильович, к.т.н., доцент кафедри МБІС	20.03.2025	11.06.2025
Розділ II	Грицак Анатолій Васильович, к.т.н., доцент кафедри МБІС	20.03.2025	11.06.2025
Розділ III	Грицак Анатолій Васильович, к.т.н., доцент кафедри МБІС	20.03.2025	11.06.2025

7.Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	24.03.2025	
2	Аналіз предметної області обраної теми	26.03.2025	27.03.2025	
3	Розробка алгоритму роботи	29.03.2025	11.04.2025	
4	Написання бакалаврської роботи на основі розробленої теми	12.04.2025	22.04.2025	
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	
6	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	03.06.2025	
7	Захист бакалаврської кваліфікаційної роботи	11.06.2025	12.06.2025	

Студент Скидан Т.М.
(підпис) (прізвище та ініціали)

Керівник роботи Грицак А.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 68 сторінок формату А4, на яких є 15 рисунків, 2 таблиці, список використаних джерел містить 24 найменувань.

Метою роботи є розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації.

У загальній частині роботи розглянуто сучасні проблеми безпечного зберігання даних у корпоративному середовищі та проаналізовано існуючі рішення, зокрема з використанням децентралізованих підходів. У проєктній частині виконано розробку архітектури системи, алгоритмів обробки запитів і контролю доступу, а також реалізовано прототип з використанням IPFS для зберігання, протоколів Zero-Knowledge Proof для забезпечення конфіденційності та мультифакторної аутентифікації для надійної автентифікації користувачів.

У третьому розділі представлено програмну реалізацію захищеної системи. Обґрунтовано вибір мови програмування та середовища розробки, детально описано реалізовані програмні модулі, зокрема збереження даних у IPFS, підтвердження доступу за допомогою Zero-Knowledge Proof та реалізацію мультифакторної аутентифікації. Також надано інструкцію користувача щодо взаємодії з системою для управління та безпечного доступу до даних.

Ключові слова: децентралізоване зберігання, корпоративні дані, IPFS, Zero-Knowledge Proof, мультифакторна аутентифікація, інформаційна безпека.

ABSTRACT

The bachelor's thesis consists of 68 A4 pages, which have 15 figures, 2 tables, the list of sources used contains 24 items.

The purpose of the work is to develop a secure system for decentralized storage and management of corporate data using IPFS, Zero-Knowledge Proof, and multi-factor authentication.

The general part of the work considers modern problems of secure data storage in a corporate environment and analyzes existing solutions, in particular using decentralized approaches. The project part includes the development of the system architecture, request processing algorithms, and access control, as well as the implementation of a prototype using IPFS for storage, Zero-Knowledge Proof protocols to ensure confidentiality, and multi-factor authentication for reliable user authentication.

The third section presents the software implementation of the secure system. The choice of programming language and development environment is justified, the implemented software modules are described in detail, in particular, data storage in IPFS, access confirmation using Zero-Knowledge Proof and implementation of multi-factor authentication. User instructions for interacting with the system for managing and securely accessing data are also provided.

Keywords: decentralized storage, corporate data, IPFS, Zero-Knowledge Proof, multi-factor authentication, information security.

ЗМІСТ

ВСТУП

Актуальність теми дослідження обумовлена стрімким розвитком цифрових технологій, зростанням обсягів даних, що обробляються у корпоративному середовищі, та посиленням загроз інформаційній безпеці. У сучасних умовах цифровізації бізнес-процесів організації стикаються з проблемами забезпечення конфіденційності, цілісності та доступності корпоративної інформації, що зберігається та передається через інформаційні системи. Класичні централізовані підходи до зберігання даних втрачають свою ефективність через уразливість до кібератак, відмов одного вузла та обмежену масштабованість.

У цьому контексті набувають актуальності децентралізовані технології зберігання даних, зокрема IPFS (InterPlanetary File System), які забезпечують підвищену надійність, масштабованість та стійкість до збоїв. Водночас зростає потреба у впровадженні надійних механізмів автентифікації користувачів. Поєднання таких інструментів, як мультифакторна автентифікація (MFA) та протоколи Zero-Knowledge Proof (ZKP), дозволяє досягти високого рівня захисту даних навіть у відкритому або розподіленому середовищі, не розкриваючи зайвої інформації та мінімізуючи ризики несанкціонованого доступу.

Метою даного дослідження є розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної автентифікації.

Основними завданнями дослідження є:

1) аналіз сучасного стану питань безпеки корпоративних даних;

2) порівняння централізованих і децентралізованих підходів до зберігання даних;

3) розробка архітектури захищеної децентралізованої системи;

4) інтеграція протоколу IPFS для розподіленого зберігання;

5) впровадження Zero-Knowledge Proof для забезпечення конфіденційності доступу;

6) реалізація механізмів мультифакторної автентифікації;

7) створення програмного прототипу та перевірка його ефективності в умовах моделювання.

Об’єктом дослідження є інформаційна інфраструктура корпоративних систем, в межах якої здійснюється зберігання, передача та обробка конфіденційних даних, зокрема в умовах децентралізованого середовища.

Предметом дослідження є методи, технології та засоби забезпечення безпеки даних у децентралізованих системах, зокрема застосування IPFS для зберігання даних, Zero-Knowledge Proof для захисту автентифікації та механізмів MFA для підтвердження особи користувача.

Наукова новизна роботи полягає в поєднанні децентралізованої файлової системи IPFS з інноваційними методами автентифікації та криптографічного захисту даних. Зокрема, використання Zero-Knowledge Proof дозволяє забезпечити конфіденційність процесів автентифікації без передачі критичних даних, що підвищує загальний рівень безпеки. Запропонована модель автентифікації є комбінованою та стійкою до типових атак на ідентифікацію користувачів.

Практична цінність дослідження полягає в реалізації прототипу захищеної системи, яка може бути застосована в реальних корпоративних середовищах для забезпечення надійного зберігання та контролю доступу до даних. Результати дослідження також можуть бути використані для подальших розробок у сфері інформаційної безпеки, зокрема в побудові хмарних і гібридних архітектур корпоративних IT-систем.

1 АНАЛІЗ СУЧАСНИХ ЗАСОБІВ ЗБЕРІГАННЯ ТА ЗАХИСТУ КОРПОРАТИВНИХ ДАНИХ

В даному розділі було проведено комплексний аналіз сучасних засобів зберігання та захисту корпоративних даних. Розглянуті основні проблеми безпечного зберігання інформації в корпоративних середовищах, а також технології, які забезпечують високий рівень захисту даних, зокрема децентралізовані файлові системи, мультифакторна автентифікація та методи криптографії, такі як Zero-Knowledge Proof.

1.1 Актуальність проблеми безпечного зберігання даних у корпоративному середовищі

За останні кілька років дані стали одним з найважливіших активів для компаній майже в усіх галузях. Вони важливі не лише для компаній, пов'язаних з індустрією комп'ютерних наук, але й для таких організацій, як уряди країн, заклади охорони здоров'я, освіти чи інженерного сектору. Дані необхідні для здійснення їхньої повсякденної діяльності, а також допомагають керівництву підприємств досягати поставлених цілей і приймати найкращі рішення на основі інформації, отриманої з них. Підраховано, що з усіх даних за всю історію людства 90 відсотків було створено за останні кілька років. У 2003 році люди створили п'ять ексабайт даних, і такий обсяг інформації в даний час створюється протягом двох днів [1].

Ця тенденція до збільшення обсягу та деталізації даних, які збираються компаніями, не зміниться в найближчому майбутньому, оскільки розвиток соціальних мереж, мультимедіа та Інтернету речей (IoT) призводить до величезного потоку даних. Крім того, ці дані здебільшого неструктуровані, а це означає, що традиційні системи не здатні їх аналізувати. Організації прагнуть отримати більше корисної інформації з цього великого обсягу та різноманіття даних. Таким чином, з'явилася нова парадигма аналізу, за допомогою якої можна

аналізувати і краще розуміти ці дані з метою отримання не лише приватних, але й суспільних переваг, і це були великі дані [1].

Кожна нова технологія приносить із собою нові проблеми. У випадку з великими даними ці питання пов'язані не лише з обсягом чи різноманітністю даних, але й з їхньою якістю, конфіденційністю та безпекою. Великі дані не тільки збільшують масштаб проблем, пов'язаних з конфіденційністю та безпекою, які вирішуються в рамках традиційного управління безпекою, але й створюють нові, до яких потрібно підходити по-новому. Оскільки організації та уряди зберігають і аналізують все більше даних, потрібно більше нормативних актів для вирішення цих проблем. Забезпечення безпеки великих даних стало одним з найважливіших бар'єрів, який може сповільнити поширення технології; без належних гарантій безпеки великі дані не досягнуть необхідного рівня довіри [1].

За даними Робочої групи з великих даних організації (Cloud Security Alliance), існує, в основному, чотири різні аспекти безпеки великих даних: безпека інфраструктури, конфіденційність даних, управління даними, а також цілісність і реактивна безпека [2].

Безпека інфраструктури є основою для захисту систем великих даних, які базуються на складних мережах серверів, сховищ і програмних платформ. Вона включає як фізичний захист, що передбачає контроль доступу до дата-центрів, моніторинг і резервування, так і віртуальний захист, спрямований на забезпечення безпеки віртуальних машин, мереж і API. Важливим аспектом є шифрування трафіку, сегментація мереж і використання багаторівневих файрволів для мінімізації ризиків [2].

Конфіденційність даних у системах великих даних вимагає комплексного підходу для захисту чутливої інформації, яка може включати особисті дані, комерційні таємниці або державні документи. Шифрування як у сховищах, так і під час передачі забезпечує, що навіть у разі перехоплення дані залишатимуться недоступними. Методи анонімізації та псевдонімізації дозволяють використовувати інформацію без розкриття персональних деталей, що особливо важливо для відповідності регуляторним вимогам, таким як GDPR. Окрім того,

розмежування доступу та впровадження багаторівневої автентифікації мінімізують ризик несанкціонованого доступу до даних [2].

Управління даними є критичним аспектом, який забезпечує, що дані залишаються організованими, доступними та захищеними. Воно включає управління правами доступу, яке визначає, хто і які операції може виконувати з інформацією. Автоматизовані інструменти для аудиту та логування допомагають відстежувати всі дії, що стосуються даних, забезпечуючи прозорість і контроль над їх використанням. Важливим елементом є регулярна перевірка відповідності даних внутрішнім і зовнішнім політикам безпеки, а також створення резервних копій і використання реплікації для забезпечення їхньої доступності [2].

Цілісність і реактивна безпека зосереджені на забезпеченні збереження даних у їхньому початковому стані та можливості швидкого реагування на інциденти. Для перевірки цілісності даних використовуються хеш-функції та механізми контрольних сум, які дозволяють виявляти несанкціоновані зміни. Системи виявлення аномалій і вторгнень працюють у реальному часі, щоб ідентифікувати потенційні загрози і мінімізувати їх вплив. Реактивна безпека включає створення планів реагування на інциденти, автоматизацію відновлення після атак і впровадження механізмів, які знижують ризики повторних атак [2].

Усі ці аспекти працюють у комплексі, щоб забезпечити високий рівень захисту даних і надійність систем великих даних, що є критично важливим у сучасному кіберсередовищі.

1.2 Огляд сучасних підходів до зберігання даних у централізованих та децентралізованих системах

У сучасному світі, що керується даними, організації стикаються з критичними рішеннями щодо зберігання цінних активів даних і керування ними. Одним із фундаментальних рішень є вибір між централізованою та децентралізованою архітектурами даних. Кожен підхід має свої переваги, і розуміння переваг і недоліків може допомогти організаціям приймати обґрунтовані рішення, які відповідають їхнім унікальним потребам і цілям.

Централізоване зберігання даних стосується практики зберігання даних в одному централізованому місці, як правило, у центрі обробки даних або хмарному середовищі. Цей підхід пропонує кілька переваг, які організації вважають привабливими. До них відносяться [3]:

1) узгодженість даних. Завдяки централізованому сховищу даних організації можуть підтримувати узгоджені та стандартизовані дані по всіх напрямках. Це сприяє цілісності даних і зменшує ризик розбіжностей або невідповідностей.

2) доступ і контроль. Централізовані дані забезпечують легший доступ, керування та контроль. Це спрощує управління даними, безпеку та управління відповідністю, оскільки політики та процедури можуть бути реалізовані однаково.

3) покращена аналітика. Централізоване сховище даних надає комплексне уявлення про дані організації, покращуючи можливості аналітики та звітності. Цей інтегрований підхід сприяє ефективному аналізу даних, визначенню тенденцій і прийняттю обґрунтованих рішень.

Однак централізоване зберігання даних також має деякі обмеження, які організації повинні враховувати [3]:

1) єдина точка відмови. Використання єдиного централізованого місця для зберігання означає, що збій у цьому місці може мати значний вплив на доступність даних і безперервність бізнесу. Надійні стратегії резервного копіювання та аварійного відновлення є важливими для зменшення цього ризику.

2) проблеми масштабованості. Оскільки обсяги даних швидко зростають, централізоване сховище може зіткнутися з проблемами масштабованості. Розширення ємності зберігання може стати складним і дорогим процесом, що вимагає ретельного планування та розподілу ресурсів.

3) вразливість до атак. Якщо зловмисники отримують доступ до центрального сервера, вони можуть компрометувати всю систему.

З іншого боку, децентралізоване зберігання даних, що передбачає розподіл даних між кількома місцями або серверами. Цей підхід пропонує кілька переваг [4]:

1) розподілене зберігання даних. Інформація розбивається на сегменти, які зберігаються на різних вузлах мережі. Вузли взаємодіють між собою для синхронізації та обміну даними.

2) надмірність і відмово стійкість. Децентралізоване сховище підвищує стійкість даних. Розповсюджуючи дані між кількома місцями, організації можуть зменшити ризик їхньої втрати або збою в роботі служби через єдину точку збою. Якщо в одному місці виникають проблеми, інші можуть продовжувати безперебійну роботу.

3) масштабованість і гнучкість. Децентралізоване сховище надає організаціям гнучкість для масштабування своєї інфраструктури на основі конкретних бізнес-підрозділів або відділів. Це дозволяє легше розширювати та розподіляти ресурси зберігання за потреби, підтримуючи нові вимоги до даних.

4) високий рівень безпеки. Децентралізована структура ускладнює можливість централізованих атак, таких як DDoS або проникнення в єдиний сервер. Дані часто дублюються і шифруються, що додатково захищає інформацію від несанкціонованого доступу.

Однак децентралізоване зберігання даних також має свої недоліки [4]:

1) синхронізація даних. Розповсюдження даних у кількох місцях потребує надійних механізмів синхронізації, щоб забезпечити їх узгодженість і уникнути конфліктів або дублювання. У порівнянні з централізованими системами, децентралізовані можуть бути повільнішими через необхідність синхронізації між вузлами. Це стає критичним для додатків, що потребують швидкої реакції.

2) ресурсоємність. Багато децентралізованих систем потребують значних обчислювальних ресурсів для забезпечення консенсусу та шифрування даних. Це може впливати на швидкість обробки запитів.

Отже, централізовані системи забезпечують ефективність, оптимізоване керування та покращені можливості аналізу, але це може призвести до розмежування даних і викликати проблеми з конфіденційністю. Децентралізовані системи забезпечують автономність, масштабованість і стійкість, але їх

застосування вимагає ретельної синхронізації та використання більшої кількості ресурсів.

1.3 Аналіз існуючих методів автентифікації

Як правило, для захисту персональної інформації користувачів у програмних додатках використовуються різні методи автентифікації, щоб запобігти доступу зловмисників до неї. Таким чином, автентифікація – це процес перевірки особи користувача перед наданням доступу до інформаційних систем або ресурсів. Іншими словами – це процес підтвердження, що користувач є саме тим, за кого себе видає.

Існують різні методи автентифікації, такі як текстові паролі, смарт-картки та біометрія. Всі згадані методи належать до різних факторів автентифікації. Фактор автентифікації – це частина інформації, яка може бути використана для перевірки особи користувача [5].

Існує три основні групи або фактори методів автентифікації [5]:

1) засновані на знаннях, тобто на тому, що користувач знає, наприклад, текстові паролі;

2) засновані на володінні, тобто на тому, чим користувач володіє, наприклад, старт-картки;

3) засновані на властивостях, тобто на тому, ким є користувач, наприклад, біометрія.

Два або більше з цих методів можуть бути об'єднані для підвищення безпеки, яка відома як багатофакторна автентифікація.

Кіберзлочинці намагаються знайти вразливості в системах автентифікації, щоб незаконно отримати доступ до чужих облікових записів з метою здійснення шахрайських дій або крадіжки даних. Таким чином, надійні та ефективні підходи до автентифікації є обов'язковими для забезпечення захисту від несанкціонованого доступу [6].

Наразі існує декілька доведених та схвалених методів автентифікації для забезпечення легітимного доступу до ресурсів. Кожен з них має свої переваги і недоліки, що ускладнює їхню адаптацію до кожного можливого сценарію [7]:

1) пароль: Використання паролю або пін-коду є одним з найстаріших та найпоширеніших механізмів захисту ресурсів від несанкціонованого доступу. Паролі вважаються безпечними, виходячи з їх розміру та складності. Але кількість паролів, випущених у темну мережу, а також інші компрометації призвели до збою схеми або навіть, в деяких випадках, до вимагання викупу чи нанесення шкоди людині. По-друге, якщо пароль скомпрометований, вся схема безпеки стає недійсною. Паролі також можуть ненадійно зберігатися, записуватися на стікерах або підслуховуватися деякими мережевими інструментами.

2) біометрія: Для забезпечення доступу використовується декілька біометричних схем, таких як розпізнавання обличчя, відбитків пальців або райдужної оболонки ока. Це дозволяє гарантувати реальний і законний доступ людини до системи, але прогрес у відтворенні біометричних зображень призвів до компрометації цієї схеми.

3) одноразовий пароль (OTP): Схема OTP широко використовується в наш час, оскільки дозволяє отримати доступ до ресурсів в режимі реального часу за допомогою нового пароля. Зазвичай OTP використовується в поєднанні з іншими схемами автентифікації, але все одно може бути скомпрометована за допомогою атаки «Людина в браузері» або шкідливого програмного забезпечення.

4) смарт-картки: Смарт-картки – це апаратні засоби, які зберігають цифрові сертифікати, ключі або облікові дані користувача і використовуються для безпечної перевірки особи, проте серед недоліків їх використання є можливі втрата чи крадіжка, а також існує ризик опосередкованих атак, зокрема за допомогою так званого «ручного ретранслятора».

5) профілювання: Профілювання користувачів часто розглядається як адаптивна схема автентифікації коли профіль користувача повністю побудований, і при незначному відхиленні від нормальної поведінки знову вводяться складні

схеми автентифікації. Така схема забезпечує швидкий доступ і гарантує, що ресурси використовують уповноваженні користувачі, але в деяких випадках, у разі надзвичайної ситуації, якщо користувач поводиться незвично, весь доступ може бути заблокований, що може призвести до серйозних наслідків. Профілювання може бути засноване на місцезнаходженні, часі, типі пристрою або навіть на підключенні. Всі вони можуть бути використані в поєднанні з іншими підходами для досягнення кращого рівня безпеки.

Порівняння методів автентифікації наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння методів автентифікації

Метод автентифікації	Опис	Переваги	Недоліки	Рівень безпеки	Зручність
Пароль / PIN-код	Використання секретного слова або числової комбінації.	Простота реалізації, не потребує додаткових пристроїв.	Вразливість до атак перебору, фішингу, соціальної інженерії; слабкі паролі.	Низький	Висока
Біометрія	Розпізнавання обличчя, відбитків пальців, райдужки ока.	Унікальність, зручність використання, не потребує запам’ятовування.	Може бути підроблена, складно змінити при компрометації.	Середній/ Високий	Висока
Одноразовий пароль (OTP)	Тимчасовий код, який надсилається або генерується для кожної сесії.	Високий рівень захисту, особливо в поєднанні з іншими методами.	Може бути перехоплений, залежність від пристрою, можливість атак типу «людина в браузері».	Високий	Середня
Смарт-картка / Токен	Апаратний носій, що містить сертифікати або ключі користувача.	Висока безпека, складність підробки, підходить для корпоративного використання.	Можлива втрата або крадіжка, потреба у спеціальному обладнанні, ризик ретрансляційних атак.	Високий	Низька/ Середня

Продовження таблиці 1.1

Профілювання (поведінкова модель)	Аналіз звичної поведінки: місце, час, пристрій, стиль набору тощо.	Адаптивність, зручність, можливість безперервної автентифікації.	Низька надійність при нетиповій поведінці, складність реалізації, помилкові блокування.	Середній	Висока
Мультифактор на автентифікація	Комбінація щонайменше двох різних методів (знання, володіння, властивості).	Найвищий рівень захисту, мінімізація ризиків кожного окремого методу.	Складність використання, потреба в додаткових ресурсах, впровадження може бути затратним.	Дуже високий	Середня

Мультифакторна автентифікація, або MFA, зараз вважається більш надійним захистом для користувачів. При мультифакторній автентифікації для перевірки особи користувача перед наданням доступу використовуються два або більше елементів. Щоб успішно проникнути в систему, зловмисникові доведеться обійти два або більше етапів автентифікації. У контексті IT-безпеки це називається багаторівневою безпекою, яка зменшує ймовірність того, що зловмисник зможе подолати всі етапи. За допомогою мультифакторної автентифікації вразливості в системі безпеки можна обійти або принаймні зменшити їхню кількість, але слід пам’ятати про зручність, масштабованість і простоту використання [6].

1.4 Аналіз існуючих децентралізованих систем для зберігання та управління корпоративними даними

Для обґрунтування доцільності розробки власної системи децентралізованого зберігання корпоративних даних із засобами автентифікації та перевірки доступу, необхідно проаналізувати існуючі рішення, які реалізують схожі функціональні можливості. До таких систем належать Filecoin, Storj та Hivenet – відомі платформи, що використовують розподілену архітектуру та

забезпечують збереження даних у децентралізованому середовищі. Їхній аналіз дозволяє визначити сильні й слабкі сторони вже реалізованих підходів і спроектувати власну систему з урахуванням усіх важливих аспектів.

Hivenet є прикладом сучасної децентралізованої платформи для зберігання даних, яка реалізує альтернативний підхід до побудови хмарної інфраструктури. Система використовує розподілену мережу повсякденних пристроїв (персональних комп'ютерів, ноутбуків, смартфонів тощо) як джерело обчислювальних ресурсів і дискового простору. Такий підхід дозволяє зменшити залежність від централізованих дата-центрів і, відповідно, знизити енергоспоживання та експлуатаційні витрати [8].

Для забезпечення конфіденційності, надійності та цілісності даних Hivenet застосовує механізми попереднього шифрування, фрагментації та розподілу фрагментів між вузлами глобальної мережі. Це дозволяє забезпечити високий рівень стійкості до втрат даних і зовнішніх атак.

Однією з ключових переваг системи є принцип повного контролю користувача над власною інформацією. Платформа підтримує гнучке налаштування прав доступу, а також забезпечує прозорість обробки даних. Hivenet функціонує на основі децентралізованої економічної моделі, що передбачає стимулювання учасників мережі (вузлів) через винагороди за надання ресурсів. Такий підхід сприяє ширшій доступності технології та знижує витрати для кінцевих користувачів [8].

До недоліків відносять [8]:

1) нестабільність вузлів мережі. Оскільки Hivenet покладається на ресурси звичайних користувацьких пристроїв, вузли можуть бути нестабільними або непостійно доступними (вимикаються, втрачають з'єднання тощо), що впливає на загальну надійність зберігання та доступу до даних.

2) проблеми з масштабованістю при високому навантаженні. За відсутності достатньої кількості учасників мережі, система може стикатися з обмеженнями в обчислювальних ресурсах, що негативно позначається на продуктивності в умовах великого обсягу запитів або зберігання.

3) залежність від добросовісності учасників. У децентралізованих середовищах важко гарантувати виконання зобов’язань усіма учасниками. Відмова вузлів зберігати або повертати дані може призводити до часткових або повних втрат фрагментів, якщо не реалізовано додаткових механізмів резервування.

4) підвищена складність забезпечення синхронізації та контролю цілісності. Децентралізована архітектура вимагає спеціальних механізмів перевірки актуальності даних і цілісності фрагментів, що може ускладнювати реалізацію та обслуговування системи.

5) Високі вимоги до безпеки кінцевих пристроїв. Безпека платформи значною мірою залежить від захищеності пристроїв користувачів, які виступають вузлами. Компрометація таких пристроїв може призвести до витоку або модифікації збережених фрагментів.

Filecoin є однією з провідних платформ у сфері децентралізованого зберігання даних, яка реалізує однорангову модель із використанням власної криптовалюти (FIL). Архітектура Filecoin базується на Міжпланетній файловій системі (IPFS), що дозволяє уникнути обмежень традиційних протоколів, таких як HTTP, та забезпечує більш ефективну, розподілену та стійку до збоїв модель передачі й зберігання інформації [9].

Однією з ключових технічних особливостей платформи є використання криптографічних доказів збереження (Proof-of-Replication) та доказів постійності зберігання (Proof-of-Spacetime). Ці механізми дають змогу підтвердити, що дані не лише були збережені, але й залишаються незмінними протягом визначеного періоду, що підвищує довіру до платформи з боку користувачів і розробників.

Filecoin створює відкритий ринок для зберігання та доступу до даних, у якому децентралізація дозволяє уникнути монополізації з боку окремих провайдерів. Постачальники сховища (майнери) отримують винагороду у вигляді токенів FIL за надання вільного місця, тоді як користувачі сплачують FIL за зберігання своїх файлів у захищеному середовищі. Реплікація даних у різних вузлах забезпечує їхню високу доступність і стійкість до втрат [9].

Поєднання блокчейн-технології, розподіленої файлової системи IPFS та стимулюючої економічної моделі робить Filecoin одним із найперспективніших рішень у сфері децентралізованого зберігання корпоративних і приватних даних.

Окрім переваг Filecoin має також низку недоліків. До них відносять [9]:

1) високі вимоги до ресурсів. Запуск вузлів Filecoin потребує значних обчислювальних потужностей і дискового простору, що ускладнює участь для звичайних користувачів і зменшує децентралізацію в практичному сенсі.

2) складність у налаштуванні та використанні. Встановлення та конфігурація клієнтів Filecoin, особливо вузлів зберігання, вимагає глибоких технічних знань, що обмежує її доступність для широкого кола користувачів або малих підприємств.

3) затримки в доступі до даних. Через специфіку децентралізованого зберігання та реплікації, час отримання даних з мережі Filecoin може бути вищим порівняно з централізованими хмарними сервісами (наприклад, Amazon S3 чи Google Cloud Storage).

4) залежність від криптовалютного ринку. Оскільки Filecoin використовує власну криптовалюту FIL, її стабільність і економічна доцільність напряму пов’язані з коливаннями ринку цифрових активів. Це може створювати фінансові ризики для користувачів та постачальників послуг.

Storj – це децентралізована платформа хмарного зберігання, яка вирізняється високим рівнем надійності, безпеки та доступності даних. Основною архітектурною особливістю Storj є розподіл зашифрованих фрагментів даних по всій мережі незалежних вузлів. Такий підхід забезпечує не лише стійкість до збоїв, але й конфіденційність збереженої інформації, оскільки жоден окремий вузол не зберігає повний файл [10].

Схема зберігання в Storj має подібність до принципів P2P-мереж, що дозволяє значно зменшити ризики втрати даних. Технологія фрагментації забезпечує резервування та самовідновлення файлів, навіть якщо частина вузлів

виходить з ладу. Завдяки сервісу Tardigrade платформа досягає довговічності та доступності на рівні 99,95%, що відповідає вимогам корпоративного рівня.

Окрему увагу Storj приділяє економічній доступності – користувачі отримують переваги децентралізованого зберігання без високих фінансових витрат, що робить платформу привабливою як для приватних осіб, так і для бізнесу. Такий підхід сприяє ширшому впровадженню безпечних технологій зберігання даних [10].

До недоліків Storj відносять [10]:

1) залежність від стабільності вузлів. Доступність даних напряму залежить від активності незалежних вузлів у мережі. У разі значного зменшення їх кількості можливе зниження швидкості відновлення даних.

2) обмежена підтримка великих обсягів даних у реальному часі. Через специфіку фрагментації та розподілу даних, обробка великих файлів або потокової інформації може бути менш ефективною порівняно з централізованими рішеннями.

3) необхідність інтеграції сторонніх сервісів. Для повноцінного використання Storj часто потрібно інтегрувати додаткові сервіси або SDK, що може ускладнити впровадження в існуючу корпоративну інфраструктуру.

Аналіз сучасних децентралізованих систем зберігання даних показав, що такі платформи, як Filecoin, Storj та Hivenet, пропонують ефективні альтернативи традиційним системам для зберігання даних з різним ступенем надійності, безпеки та доступності. Водночас кожне з рішень має свої недоліки, що обґрунтовує доцільність розробки власної захищеної системи з урахуванням конкретних вимог корпоративного середовища.

1.5 Висновки до розділу 1

Отже, у першому розділі було розглянуто актуальність проблеми безпечного зберігання даних у корпоративному середовищі, що набуває особливої важливості в умовах швидкого зростання обсягів даних і їх різноманітності.

Проведено огляд сучасних підходів до зберігання корпоративних даних як у централізованих системах, які забезпечують зручність, але залежать від однієї точки відмови, так і в децентралізованих рішеннях, що базуються на розподілених мережах та підвищують стійкість до атак і втрат.

Також проаналізовано сучасні методи автентифікації – від паролів і смарт-карток до біометрії та одноразових кодів – із виявленням їхніх переваг, недоліків та доцільності використання мультифакторної автентифікації. Окрему увагу приділено аналізу існуючих рішень у сфері децентралізованого зберігання, таких як Filecoin, Storj та Hivenet, що демонструють різні підходи до забезпечення конфіденційності, доступності та довговічності даних.

Проведене дослідження дозволило визначити ключові технології та концепції, які можуть бути використані для побудови нової захищеної системи для зберігання та управління корпоративними даними. Це обґрунтовує вибір таких рішень, як IPFS, Zero-Knowledge Proof та мультифакторна автентифікація, як основу подальшої розробки.

2 ПРОЕКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ДЛЯ ЗБЕРІГАННЯ ДАНИХ

У цьому розділі було описано архітектуру децентралізованої системи для захищеного зберігання розподілених даних. Розглянуто ключові компоненти системи, зокрема механізми шифрування, управління доступом та інтеграцію блокчейн-технологій та бази даних.

2.1 Розробка архітектури децентралізованої системи для зберігання даних

Для розробки архітектури децентралізованої системи для зберігання розподілених даних спершу необхідно спроектувати блок-схему для розумння принципу взаємодії клієнта із системою.

Блок-схема – це діаграма, на якій зазвичай зображено процес, систему або комп’ютерний алгоритм і яка використовується для документування, планування, уточнення або візуалізації багатоетапного робочого процесу. За допомогою блок-схем можна визначити цілі й обсяги робочого процесу, а також розташувати необхідні завдання в хронологічному порядку [11].

Виходячи з мети розробки структура системи складається з наступних етапів:

– реєстрація користувача;

– авторизація та аутентифікація;

– завантаження та шифрування даних;

– обробка даних у серверній частині;

– зберігання даних;

– отримання даних за запитом користувача після перевірки прав доступу;

– дешифрування на стороні клієнта.

Описану структуру взаємодії користувача з програмою можна відобразити за допомогою наступної схеми (рис. 2.1).

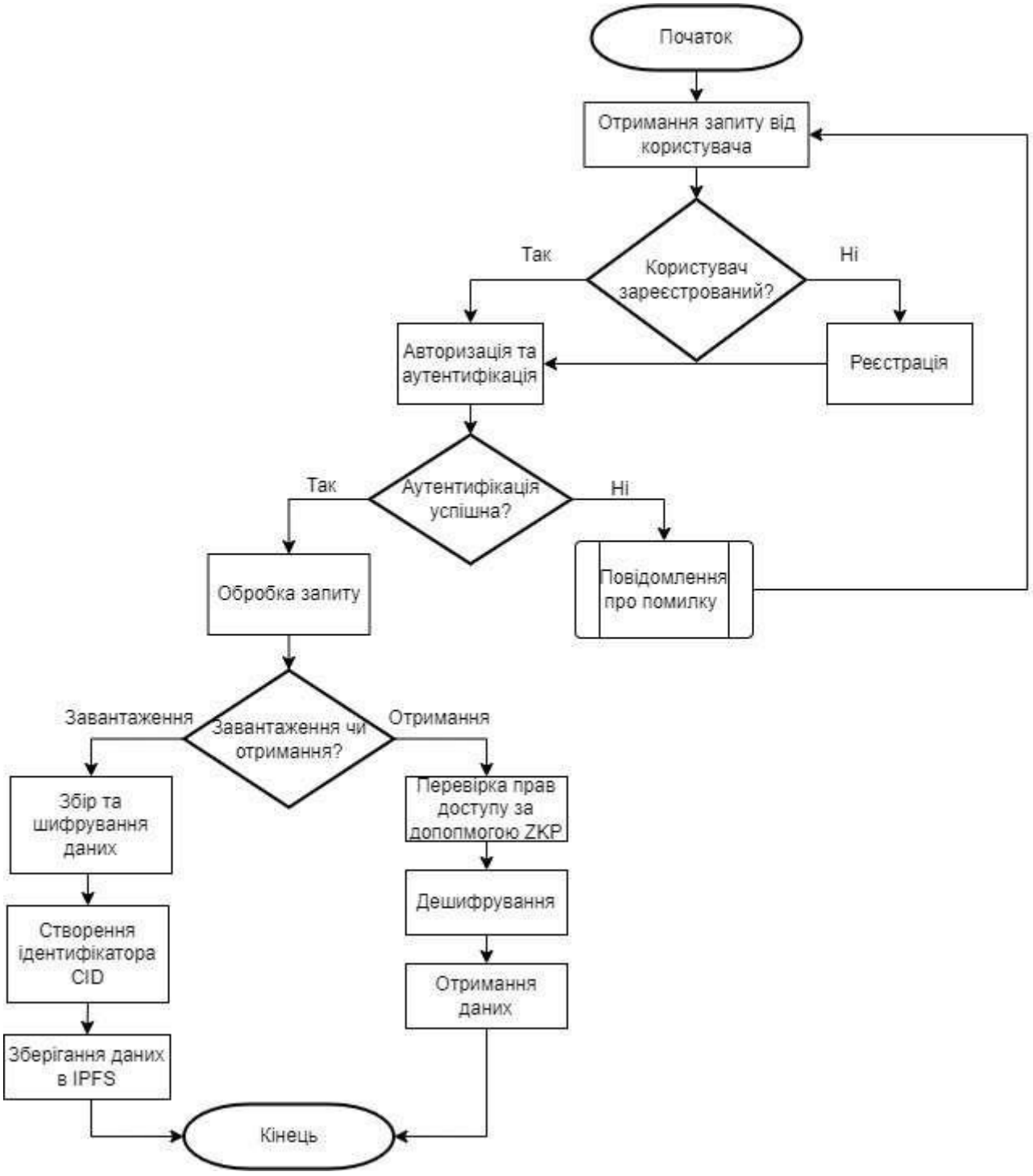


Рисунок 2.1 – Схема взаємодії користувача із системою

Крок 1. Система отримує запит від користувача на вхід.

Крок 2. Перевірка, чи користувач зареєстрований. Якщо так – перехід до авторизації та автентифікації, якщо ні – перенаправлення користувача на реєстрацію, після якої повернення до авторизації та автентифікації.

Крок 3. Клієнт вводить облікові дані та проходить процедуру перевірки особистості, тобто мультифакторну автентифікацію.

Крок 4. Система перевіряє успішність автентифікації. Якщо так – обробка запиту, якщо ні – виводиться повідомлення про помилку, і користувач може повторити запит.

Крок 5. Клієнт обирає файл для зберігання. Дані локально шифруються на стороні клієнта за допомогою симетричного алгоритму Fernet.

Крок 6. Для зашифрованих даних генерується CID – хеш, який однозначно ідентифікує вміст файлу

Крок 7. Зашифровані дані зберігаються в IPFS, система повертає CID.

Крок 8. Клієнт надсилає запит на отримання даних.

Крок 9. Система перевіряє, чи має користувач право доступу до даних, використовуючи Zero-Knowledge Proof та дані бази SQLite.

Крок 10. Після чого система надає CID. Дані, отримані клієнтом, дешифруються локально за допомогою криптографічного ключа, що виключає ризики витоку інформації.

Крок 11. Клієнт отримує доступ до розшифрованої інформації у початковому вигляді.

Далі було розроблено захищену базу даних, використовуючи СУБД SQLite для зберігання інформації про користувачів, права доступу та інформацію про завантажені файли без розкриття їхнього вмісту.

Для створення бази даних потрібні наступні сутності: КОРИСТУВАЧІ, ПРАВА ДОСТУПУ та ФАЙЛИ. До атрибутів (властивостей) вказаних сутностей віднесено:

USERS (Id, User_Name, Password, TOPT_Secret);

ACCESS_RIGHTS (Id, File_Id, User_Id, Access_Level);

FILES (Id, User_Id, File_Name, CID, Encryption_Key).

Для наочності було створено ER-модель, яка демонструє зв'язки між основними сутностями системи (рис. 2.2).

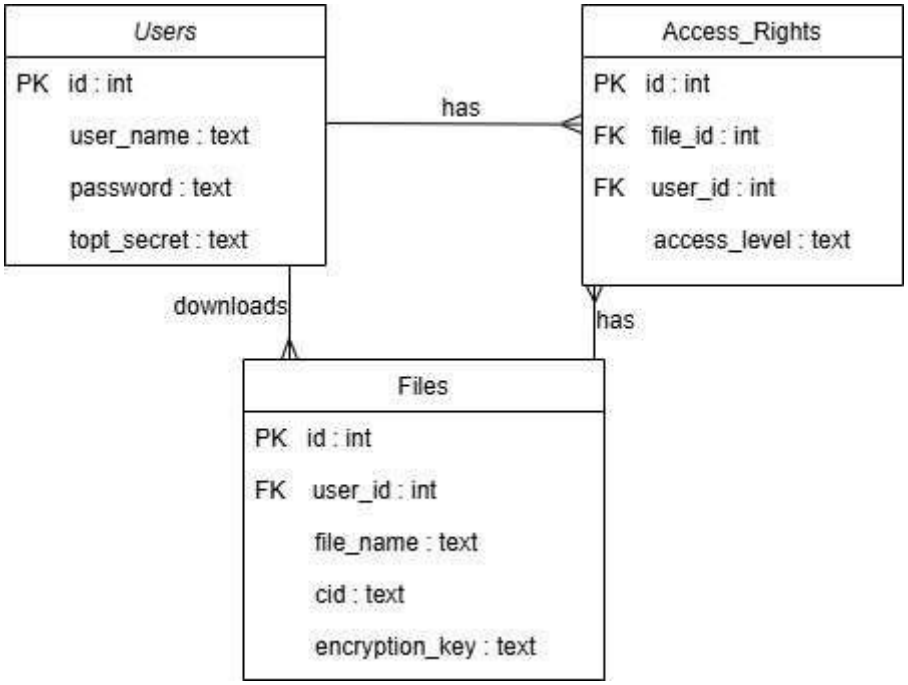


Рисунок 2.2 – ER-модель реалізованої бази даних

Такі дані, як пароль користувача, одноразовий пароль з часовим обмеженням (TOPT), однозначний ідентифікатор вмісту файлу (CID) та ключ шифрування файлу зберігаються у базі даних у вигляді хеш-значення. Хеш-функція – це функція, яка приймає на вхід відносно довільну кількість даних і видає на виході бітовий рядок фіксованого розміру [12].

Хешування здійснюється з використанням алгоритму SHA-256 – це ітераційна криптографічна хеш-функція, заснована на функції стиснення, яка оновлює вісім 32-бітових змінних стану A,..., H відповідно до значень 16-ти 32-бітових слів M0,..., M15 повідомлення. Функція стиснення складається з 64 однакових кроків (рис. 2.3) [13].

Для покрокового перетворення використовуються порозрядні булеві функції f_{MAJ} та f_{IF}.

$\Sigma_0(x)=ROTR^2(x)\oplus ROTR^{13}(x)\oplus ROTR^{22}(x)$,

$\Sigma_1(x)=ROTR^6(x)\oplus ROTR^{11}(x)\oplus ROTR^{25}(x)$,

Де $ROTR^n(x)$ – циклічний зсув X на n позицій праворуч.

і-й крок використовує фіксовану константу K_i та і-те слово W_i розширеного повідомлення. Розширення повідомлення працює наступним чином: вхідне

повідомлення доповнюється (додаються додаткові біти, зазвичай це «1», за якою слідує декілька «0» в кінець даних, а також біт, що вказує на вихідну довжину даних) та розбивається на 512-бітні блоки повідомлень. Нехай ME позначає функцію розширення повідомлення. ME приймає на вхід вектор M з 16 координатами та виводить вектор W з N координатами. Координати W_i розширеного вектора генеруються з початкового повідомлення M за такою формулою [13]:

$$W_i = \begin{cases} M_i & for\ 0 \le i < 16 \\ \delta_1(W_{i-2}) + W_{i-7} + \delta_0(W_{i-15}) + (W_{i-16}) & for\ 16 \le i < N \end{cases}$$

Взяття значення для N, відмінного від 64, призводить до покроково скороченого хеш-функції. Функції $\sigma_0(x)$ та $\sigma 1(x)$ визначаються наступним чином: $\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$ та $\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$,

де $SHR^n(x)$ – бітовий зсув X на n позицій праворуч [13].

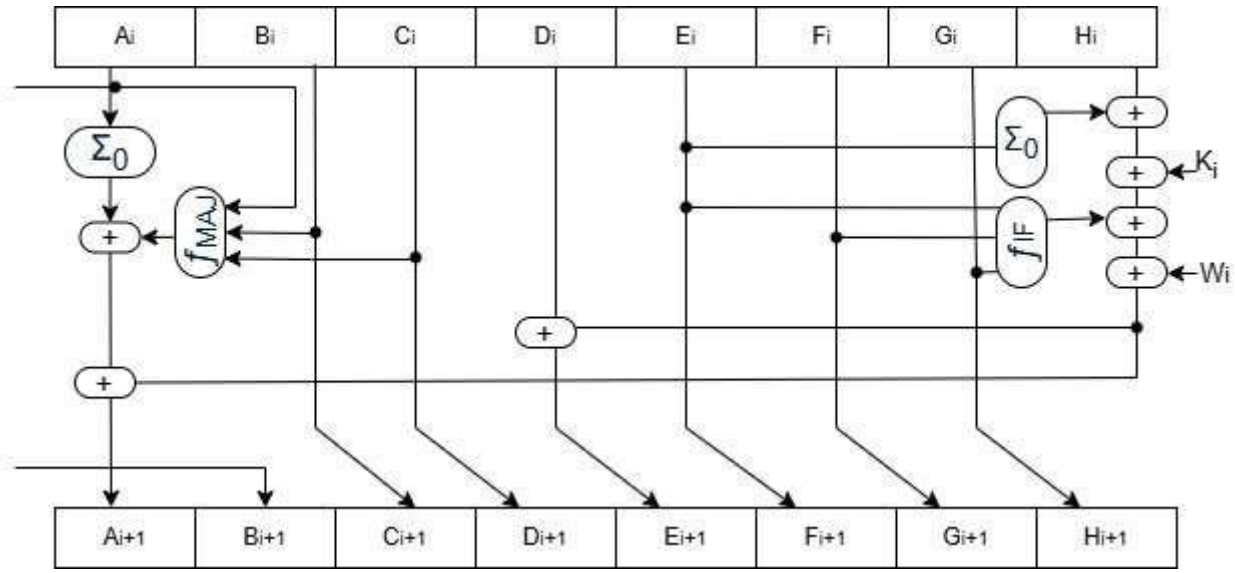


Рисунок 2.3 – Схема роботи одного кроку функції стиснення

Таким чином, алгоритм SHA-256 забезпечує стійке криптографічне хешування шляхом багаторазових ітерацій функції стиснення, яка ефективно змішує вхідні дані з використанням побітових операцій, фіксованих констант і механізму розширення повідомлення. Це гарантує високу чутливість до змін вхідного повідомлення та надійний рівень захисту даних.

2.2 Розробка механізмів конфіденційності та автентифікації з використанням Zero-Knowledge Proof та MFA

У межах розробки захищеної системи зберігання та управління корпоративними даними було впроваджено два ключові компоненти безпеки: механізм автентифікації на основі мультифакторної автентифікації (MFA) та протокол конфіденційності з використанням Zero-Knowledge Proof (ZKP) для забезпечення автентифікації без розкриття чутливих даних.

Архітектуру модулів конфіденційності та автентифікації зображено у вигляді блок-схеми на рисунку 2.3.

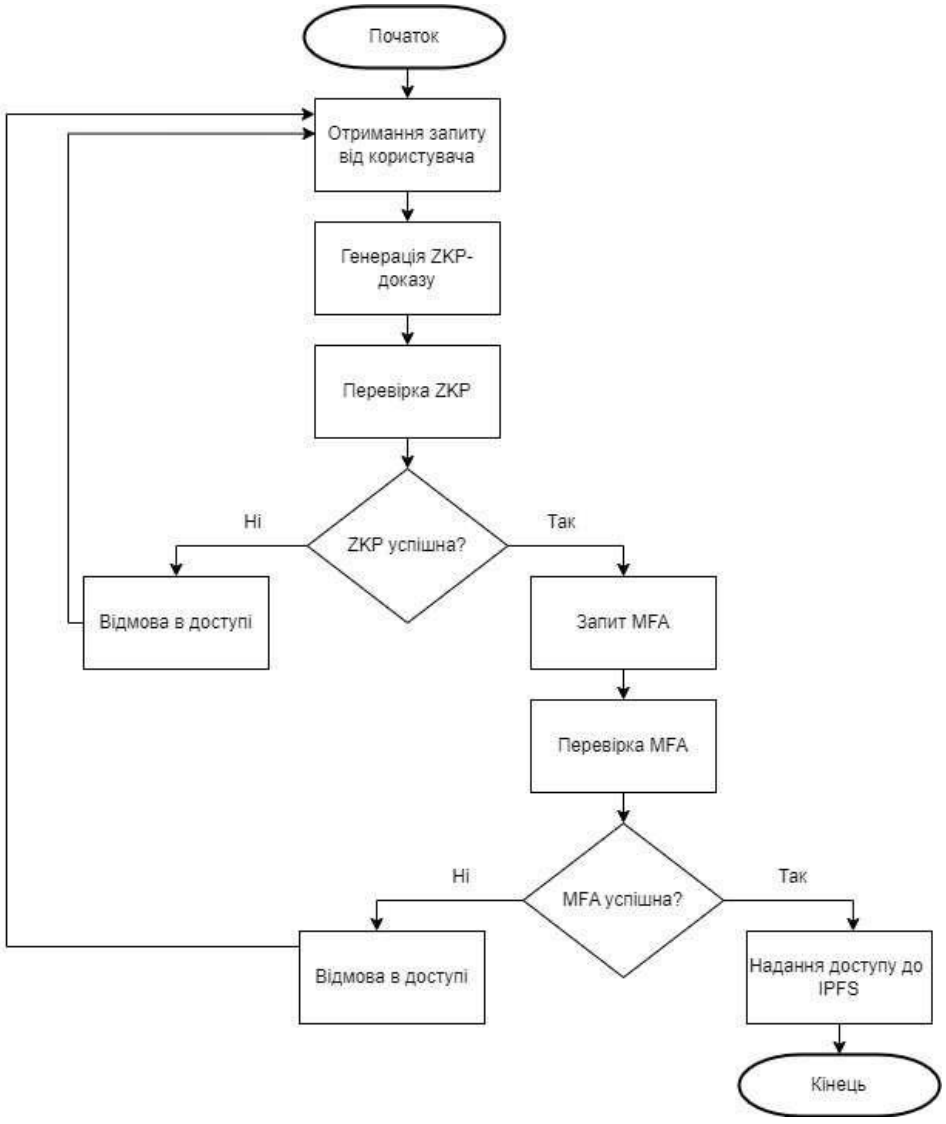


Рисунок 2.3 – Схема взаємодії модулів автентифікації з використанням ZKP та MFA

Крок 1. Користувач ініціює запит через інтерфейс системи.

Крок 2. Генерується Zero-Knowledge доказ, який перевіряється сервером.

Крок 3. У разі успішної перевірки – запитується додатковий фактор MFA.

Крок 4. У разі успішної двофакторної автентифікації доступ надається контролером доступу, інакше – відмова.

Крок 5. Користувач отримує дозволений доступ до даних в IPFS.

Zero-Knowledge Proof (ZKP) – це криптографічний протокол, який дозволяє одній стороні (доказувачу) переконати іншу сторону (верифікатора) у тому, що певне твердження є істинним, не розкриваючи при цьому жодної додаткової інформації про саме твердження або його секретні дані. Для досягнення цієї властивості ZKP використовує низку криптографічних примітивів, серед яких важливе місце займають схеми зобов'язань (commitment schemes).

Схема зобов'язань дозволяє доказувачу «закріпити» певне значення або повідомлення у вигляді спеціального криптографічного зобов'язання, яке публікується, не розкриваючи самих даних. Пізніше доказувач може розкрити це зобов'язання, надавши часткову або повну інформацію, яку верифікатор може перевірити на відповідність початковому зобов'язанню. Такий механізм гарантує цілісність і незмінність прихованої інформації, що є фундаментальним для реалізації ZKP [14].

Загалом до схем зобов'язань відносять чотири основні компоненти [14]:

1) setup (налаштування) – приймає початкові параметри і генерує параметри налаштування, які визначають тип об'єктів, що підлягають зобов'язанню та рівень криптографічного захисту, зазвичай вимірюваний у бітах безпеки. Рівень безпеки n біт означає, що для злому системи необхідно виконати приблизно O(2^n) операцій.

2) commit (приєднати) – створює зобов'язання на повідомлення відповідно до встановлених параметрів, яке публікується як доказ зобов'язання.

3) open (часткове розкриття) – генерує доказ часткового розкриття, який стосується конкретної частини повідомлення (наприклад, значення полінома для заданого аргументу), використовуючи параметри налаштування.

4) verify (перевірити) – перевіряє коректність розкритої інформації за допомогою доказу часткового відкриття і параметрів налаштування, визначаючи, чи відповідає вона початковому зобов’язанню.

Загальна архітектура ZKP-протоколу наведена на рисунку 2.4 [15].

Рисунок 2.4 – Архітектура протоколу доведення з нульовим розголошенням

Крок 1. Створення двох ключів: ключ оцінки (передається Доказувачу) та ключ перевірки (передається Верифікатору).

Крок 2. Користувач надсилає до Точки обчислення відкриті (вхідні) дані, необхідні для виконання обчислень.

Крок 3. Користувач надсилає Доказувачу приватні дані разом із копією вхідних даних. Ці дані необхідні для створення доказу правильності обчислень.

Крок 4. На основі отриманих даних та ключа оцінки Доказувач формує доказ правильності обчислень і передає його до Точки обчислення.

Крок 5. Точка обчислення виконує необхідні обчислення та формує результат разом із згенерованим доказом, який потім надсилається Користувачу.

Крок 6. Користувач надсилає Верифікатору запит на підтвердження отриманого доказу. Разом із запитом передаються вхідні дані та сам доказ.

Крок 7. Верифікатор використовує ключ перевірки для перевірки достовірності отриманого доказу. У разі успішної перевірки він надсилає Користувачу підтвердження правильності обчислень.

Таким чином, використання протоколів з нульовим розголошенням (ZKP) дозволяє гарантувати достовірність обчислень і передачі даних без необхідності розкривати їх зміст. Це особливо актуально в умовах обробки конфіденційної інформації, коли необхідно забезпечити високий рівень довіри між сторонами без порушення приватності.

Однак ефективний захист інформації вимагає також надійного контролю доступу до самих систем. Одним із найважливіших аспектів такого контролю є автентифікація користувача. У зв’язку з цим у межах даного проєкту також було реалізовано мультифакторну автентифікацію (MFA) – механізм, що поєднує декілька факторів перевірки особи з метою посилення безпеки доступу до системи.

Реалізована мультифакторна автентифікація включає основний пароль користувача та одноразовий пароль . Одноразовий пароль (OTP) – це тимчасовий код, який використовується лише один раз для автентифікації користувача, що значно підвищує безпеку системи, додаючи другий фактор автентифікації [16].

Основні етапи алгоритму [16]:

1) спільний секрет – сервер і клієнт (користувач) мають спільний секретний ключ K, який використовується як основа для генерації OTP.

2) генерація вхідного значення (час або лічильник):

– для Time-based OTP (TOTP) – використовується поточний час, розділений на фіксовані інтервали (наприклад, кожні 30 секунд).

– для HMAC-based OTP (HOTP) – використовується лічильник подій (наприклад, кількість входів чи авторизаційних запитів).

3) обчислення HMAC – на основі ключа K і значення часу або лічильника C обчислюється HMAC-функція: HMAC = HMAC_SHA256(K,C).

4) динамічне відсікання (Dynamic Truncation) – з отриманого результату HMAC-функції за допомогою останніх 4 біт визначається зсув, і на його основі виділяється 4-байтний фрагмент.

5) генерація OTP – виділене число перетворюється в десятковий код фіксованої довжини шляхом обчислення за модулем:

$OTP = (truncated\_value) mod 10^d$,

де d – кількість цифр у паролі (зазвичай 6 або 8).

6) надсилання OTP користувачу – сформований OTP-код передається користувачу, наприклад, через SMS, email або мобільний додаток.

7) перевірка OTP сервером – користувач вводить отриманий OTP-код. Сервер повторює процедуру генерації OTP на основі відомого йому секретного ключа K та відповідного значення C, і порівнює з введеним кодом. Якщо значення співпадають і OTP ще не був використаний, автентифікація вважається успішною.

Безпекові переваги запропонованого підходу, який поєднує Zero-Knowledge Proof та мультифакторну автентифікацію у вигляді паролю та одноразового паролю, є суттєвими і різноплановими. По-перше, висока стійкість до атак типу «man-in-the-middle» забезпечується тим, що облікові дані, зокрема пароль, не передаються мережею в явному вигляді. Замість цього користувач формує доказ володіння секретом за допомогою Zero-Knowledge Proof, що дозволяє серверу перевірити автентичність без розкриття самого пароля. Це унеможливлює перехоплення даних і їх повторне використання зловмисниками. По-друге, застосування ZKP виключає можливість підбору або крадіжки пароля, оскільки пароль не передається та не зберігається на сервері у вигляді, придатному для аналізу, що робить традиційні атаки типу brute-force, фішинг або словникові атаки неефективними.

Вторинним рівнем захисту є використання одноразового пароля (OTP) як другого фактора автентифікації, що суттєво ускладнює несанкціонований доступ навіть у разі компрометації основного пароля. Це забезпечує додатковий бар’єр, адже зловмиснику необхідно мати не лише знання пароля, а й тимчасовий OTP, що генерується або надсилається на довірений пристрій користувача. Такий

підхід підвищує загальну безпеку системи, не жертвуючи зручністю для кінцевого користувача.

Крім того, використання Zero-Knowledge Proof і MFA на основі пароля та OTP не прив’язане до централізованих серверів чи конкретних провайдерів, що підвищує масштабованість та гнучкість системи. Рішення легко інтегрується у різні інфраструктури, включаючи децентралізовані середовища або приватні мережі, що особливо важливо для корпоративних даних із високими вимогами до безпеки. Таким чином, поєднання ZKP із двофакторною автентифікацією створює багаторівневий, надійний захист, який відповідає сучасним стандартам безпеки.

2.3 Обґрунтування інтеграції IPFS в децентралізовану систему

Міжпланетна файлова система (IPFS) – це розподілена система для зберігання і доступу до файлів, веб-сайтів, додатків і даних. На відміну від Всесвітньої мережі (WWW), яка знаходить інформацію за її адресою/місцем розташування, IPFS знаходить інформацію за її змістом, тому її ще називають системою, що базується на змісті [17].

Для того, щоб така система працювала, IPFS побудована на багатьох модульних бібліотеках, які підтримують певні частини системи. Три основні компоненти [17]:

а) адресація контенту/ідентифікатори контенту (CID);

б) зв'язування контенту за допомогою спрямованих ациклічних графів Меркла (DAG);

в) виявлення контенту за допомогою розподілених хеш-таблиць (DHT).

Або іншими словами: IPFS використовує багато різних, простих і усталених технологій і поєднує їх у новому та унікальному підході, в результаті чого створюється файлова система, яка намагається з'єднати всі вузли з однією системою файлів, яка не має єдиної точки відмови і де вузлам не потрібно довіряти один одному. Процес системи обміну та зберігання файлів в IPFS можна розділити на кілька етапів.

Перший етап – це етап передачі файлів, на якому власник завантажує файл в мережу IPFS за допомогою вузла IPFS. Завантажені файли шифруються, а потім розбиваються на менші частини, які потім розподіляються по мережі IPFS. Сегменти файлів реплікуються між кількома вузлами IPFS, гарантуючи, що файли завжди будуть доступні, навіть якщо один вузол вийде з ладу або перейде в офлайн.

Другий етап – це етап автентифікації, на якому користувачі, які хочуть отримати доступ до спільних файлів, повинні пройти автентифікацію. Після підтвердження користувач може отримати доступ до файлу і завантажити його з мережі IPFS. Розроблена децентралізована система перевіряє ідентифікаційні дані користувачів і гарантує, що тільки авторизовані користувачі можуть отримати доступ до спільних файлів.

Третій етап – це етап передачі файлів, на якому користувачі завантажують спільні файли з мережі IPFS. Під час процесу завантаження вузли IPFS отримують фрагменти файлів з мережі та збирають їх в оригінальний файл. Потім завантажений файл розшифровується за допомогою приватного ключа власника, так що тільки авторизовані користувачі можуть отримати доступ до оригінального файлу [17].

Взаємодію користувача з IPFS можна зобразити у вигляді діаграми послідовностей (рис. 2.5) [18].

Діаграма послідовностей графічно демонструє порядок взаємодії певних об’єктів програми у часі. Як правило, в цій діаграмі демонструється, як користувачі взаємодіють з іншими компонентами програми під час реалізації тих чи інших варіантів її використання, та як при цьому взаємодіють інші компоненти програмної системи. Діаграми послідовності є одним із способів формалізації сценаріїв використання. Її перевага заключається в тому, що на ранніх стадіях опису сценаріїв можливо з’ясувати склад взаємодіючих компонентів та описати потік повідомлень від одних компонентів до інших [18].

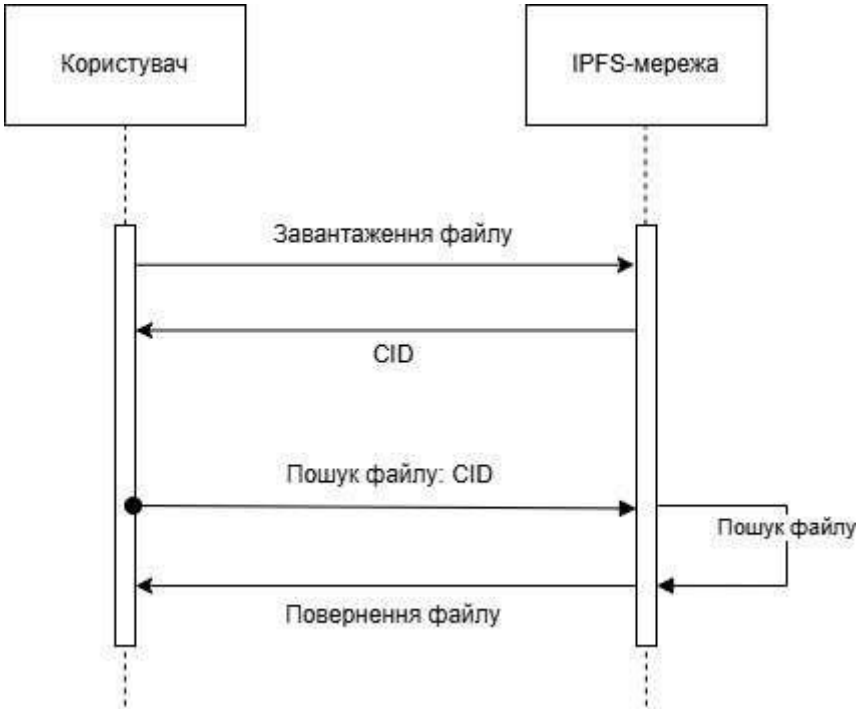


Рисунок 2.5 – Діаграма послідовностей взаємодії користувача з IPFS

Крок 1. Користувач завантажує файл до мережі IPFS.

Крок 2. У відповідь мережа IPFS генерує та повертає CID (Content Identifier) – унікальний хеш, що ідентифікує вміст файлу.

Крок 3. Користувач ініціює новий процес – отримання файлу, надсилаючи запит до IPFS, вказуючи CID, щоб знайти потрібний файл.

Крок 4. IPFS-мережа виконує пошук відповідного файлу за вказаним CID.

Крок 5. Мережа повертає знайдений файл користувачу.

Щоб забезпечити конфіденційність даних, перед збереженням у мережі IPFS файли шифруються за допомогою симетричного алгоритму Fernet, який гарантує цілісність і недоступність вмісту без відповідного ключа.

Fernet – це техніка симетричного шифрування, доступна користувачам у модулі cryptography в Python, яка використовує алгоритм Advanced Encryption Standard (AES) для кодування та декодування повідомлень. AES – один з найпоширеніших симетричних алгоритмів блокового шифрування, у якому розмір блоку 128 біт, а розмір ключ 128/192/256 біт [19].

Алгоритм Fernet забезпечує створення шифротекстів, які є безпечними для передавання через веб-протоколи, зокрема HTTP(S). Це спрощує обмін

зашифрованими даними в мережевому середовищі. Для шифрування використовується 32-байтовий буквенно-цифровий ключ, згенерований випадковим чином, що забезпечує високий рівень захисту від атак типу brute force. Fernet також підтримує механізм ротації ключів, тобто дозволяє періодично створювати нові ключі та замінювати застарілі.

Крім того, у шифротекст вбудовуються метадані, зокрема мітка часу та серіалізовані дані (тип операції, структуровані блоки даних), що дозволяє обмежувати термін дії ключа та підвищує загальний рівень безпеки системи шифрування [19].

Покроковий алгоритм роботи Fernet наведено на рисунку (рис. 2.6) [20].

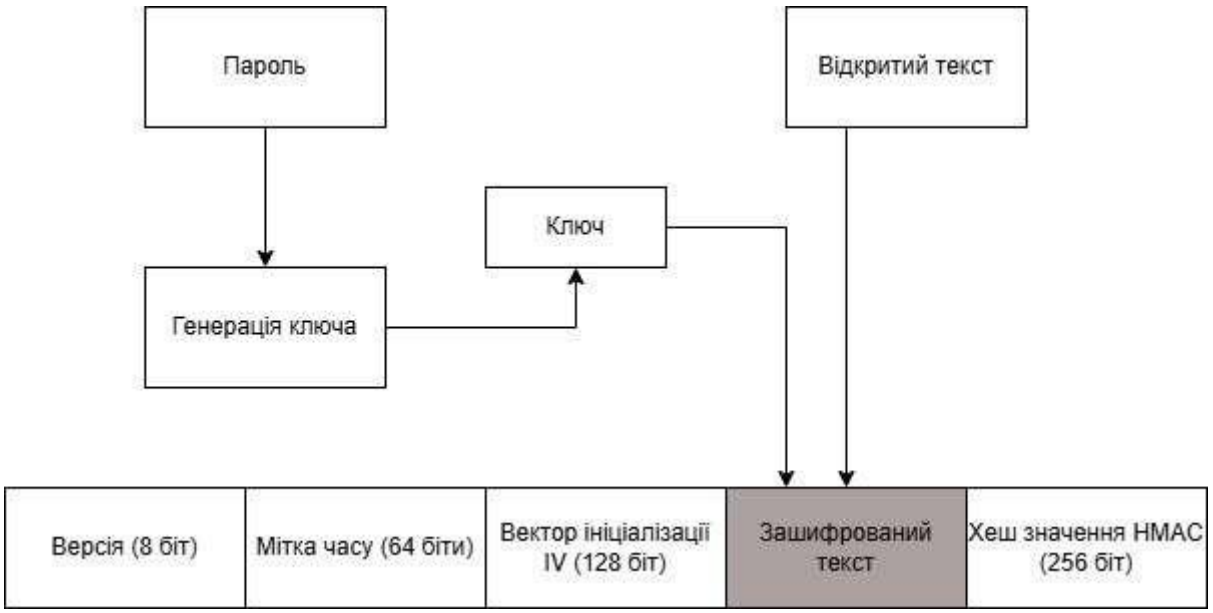


Рисунок 2.6 – Алгоритм роботи Fernet

Крок 1. Користувач вводить пароль та завантажує відкритий текст.

Крок 2. За допомогою функції виведення ключа із введеного пароля генерується секретний криптографічний ключ, який багаторазово хешується, щоб запобігти атаці brute force. Результ – ключ довжиною 256 біт (32 байти), де перші 128 біт використовуються для шифрування (AES), а інші 128 біт – для HMAC (аутентифікація).

Крок 3. Генерується випадковий вектор ініціалізації (IV) довжиною 128 біт, який забезпечує, що кожне шифрування навіть однакового тексту дає різний результат.

Крок 4. Додається мітка часу, тобто час, коли було здійснено шифрування (64 біти).

Крок 5. Здійснюється шифрування відкритого тексту за допомогою AES (128-бітний ключ) у режимі CBC (Cipher Block Chaining), використовуючи IV. В результаті цього отримується зашифрований текст.

Крок 6. До зашифрованих даних додається HMAC (код автентифікації повідомлення на основі хешу) для гарантування цілісності та достовірності повідомлення, який обчислюється з використанням алгоритму SHA-256 та іншого ключа.

Крок 7. Формування токена: фінальний шифротекст включає версію, мітку часу, вектор ініціалізації (IV), зашифровані дані та HMAC, представлені у форматі Base64.

Для дешифрування даних алгоритм Fernet виконує зворотні дії:

Крок 1. Токен розбивається на відповідні компоненти.

Крок 2. Обчислюється HMAC ще раз і порівнюється з отриманим, для того, щоб переконатись, що зашифровані дані не було змінено.

Крок 3. Шифрований текст розшифровується за допомогою AES і IV, в результаті чого отримуються відкритий текст.

Таким чином, алгоритм Fernet забезпечує не лише конфіденційність даних завдяки симетричному шифруванню, а й їхню цілісність та автентичність за допомогою HMAC.

2.4 Обґрунтування вибору засобів програмування

Для реалізації проєкту було проаналізовано найпопулярніші системи управління базами даних. В результаті аналізу було обрано SQLite.

SQLite – це компактна вбудована реляційна база даних з відкритим кодом. Замість того, щоб працювати незалежно як окремий процес, вона симбіотично

співіснує всередині програми, яку обслуговує, тобто в її просторі процесів, її код вбудований, як частина програми, яка її розміщує [21].

На відміну від серверних систем управління базами даних, таких як MySQL чи PostgreSQL, SQLite не є окремим сервером. Це вбудовувана бібліотека, яку можна інтегрувати безпосередньо в додаток. Завдяки цьому не потрібно окремо встановлювати чи налаштовувати сервер бази даних для її використання.

До недоліків SQLite відносять [22]:

1) SQLite не розрахована на велику кількість одночасних операцій запису – у таких випадках можливі блокування або конфлікти при доступі до даних.

2) дана система управління базами даних не підходить для роботи з великими обсягами інформації чи для високонавантажених застосунків через свої обмеження.

3) віддалене підключення через мережу в SQLite не підтримується, оскільки вона функціонує як локальна база даних.

4) відсутність безкоштовної офіційної технічної підтримки – у разі проблем користувачі змушені покладатися на допомогу спільноти.

5) основна сфера застосування SQLite – це базові CRUD-операції (створення, читання, оновлення, видалення), тому вона не є найкращим вибором для складної аналітики або обробки великих обсягів даних.

Серед основних переваг SQLite [21]:

1) мінімальні вимоги до системних ресурсів, що робить цю СУБД ідеальною для мобільних додатків та інших ресурсозалежних середовищ.

2) SQLite підтримує більшість стандартних SQL-операцій, що дозволяє ефективно виконувати запити та управляти даними. Оскільки вона не використовує окремого сервера, доступ до бази здійснюється безпосередньо через API бібліотеки, доступні для різних мов програмування. Це значно спрощує інтеграцію з програмою.

3) SQLite підтримує транзакції, що дозволяє забезпечити цілісність та безпеку даних під час одночасного виконання декількох операцій. Завдяки своїй простоті та ефективності, SQLite активно застосовується в мобільних, десктопних

додатках, браузерах та багатьох інших типах програм, де немає потреби у повноцінному сервері баз даних.

4) компактність – вся база зберігається в одному файлі. Ефективно використовує простір завдяки змінній довжині записів.

Важливим фактором вибору SQLite є також її інтеграція з популярними мовами програмування, такими як Python, C#, Java тощо. Це забезпечує зручність розробки та розширення функціональних можливостей системи.

Крім того, саме поєднання простоти, легкості інтеграції, достатньої функціональності та відкритого коду робить SQLite оптимальним вибором для проєктів, де важлива швидкість розробки, стабільність та мінімальні вимоги до ресурсів.

Також для вибору мови програмування було проаналізовано переваги та недоліки деяких з них (табл. 2.1) [23].

Таблиця 2.1 – Порівняння мов програмування

Критерій	Python	Java	C++	JavaScript	Ruby
Тип мови	Високий рівень, інтерпретована	Високий рівень, компільована	Середній рівень, компільована	Високий рівень, інтерпретована	Високий рівень, інтерпретована
Продуктивність	Середня	Висока	Дуже висока	Середня	Середня
Масштабованість	Середня	Висока	Дуже висока	Середня	Середня
Безпека	Висока	Висока	Середня	Середня	Середня
Швидкість	Середня	Висока	Дуже висока	Середня	Середня

Після аналізу різних критеріїв, включаючи безпеку та швидкість, було обрано Python. Це рішення базується на його простоті використання, а також достатній швидкості для більшості завдань, пов'язаних з наукою про дані, автоматизацією та веб-розробкою. Хоча Python не є найшвидшим у порівнянні з

іншими мовами, його переваги в ефективності розробки та широкому застосуванні роблять його ідеальним вибором для даного проекту [23].

Python – це високорівнева інтерпретована мова програмування, яка відома своєю простотою, читабельністю та багатою екосистемою бібліотек. Вона була створена Гвідо ван Россумом і вперше випущена в 1991 році. Завдяки своїй гнучкості та потужності, Python став однією з найпопулярніших мов програмування у світі.

Python розроблений із акцентом на читабельність коду. Його синтаксис максимально наближений до природної мови, що робить мову зрозумілою навіть для новачків. Наприклад, для оголошення змінних або написання функцій потрібно мінімум коду, що підвищує швидкість розробки. Він є інтерпретованою мовою, що означає, що код виконується безпосередньо через інтерпретатор, а не компілюється в машинний код заздалегідь. Це дозволяє швидше тестувати та розробляти програму, але може трохи знижувати продуктивність у порівнянні з компільованими мовами [23].

Python підтримує об'єктно-орієнтовану парадигму програмування (ООП), що дозволяє ефективно працювати з даними та методами в межах об'єктів. Це дає змогу створювати складні структури даних, які можна легко використовувати та розширювати. Хоча Python не є настільки швидким, як компільовані мови (наприклад, C++ чи Java), його простота та швидкість розробки компенсують цей недолік в багатьох випадках. Однак для важких обчислень або вимог до високої продуктивності можуть використовуватись спеціальні бібліотеки або інтерфейси до низькорівневих мов, наприклад, через Cython або PyPy [23].

Середовищем розробки було обрано Visual Studio Code – це потужне, легке та багатофункціональне середовище для розробки, яке підтримує широкий спектр мов програмування, включаючи Python, JavaScript, C++, Java та багато інших. VS Code є відкритим програмним забезпеченням, розробленим компанією Microsoft, і на сьогодні є одним з найпопулярніших текстових редакторів для програмістів.

До основних характеристик Visual Studio Code належать [24]:

1) легкість та швидкість. VS Code має компактний розмір і працює швидко, що робить його зручним інструментом для розробки навіть на машинах з обмеженими ресурсами.

2) підтримка багатьох мов програмування. VS Code підтримує велику кількість мов програмування за допомогою плагінів та розширень, що дозволяє налаштувати середовище під конкретні потреби розробника.

3) інтеграція з Git. Однією з ключових переваг VS Code є вбудована підтримка Git, що дозволяє зручно працювати з версіями коду без необхідності використовувати зовнішні інструменти для керування репозиторіями.

4) підтримка розширень. VS Code має велику бібліотеку розширень, що дає змогу додавати підтримку нових мов, інструментів для тестування, підключення до баз даних та багато іншого. Це робить середовище гнучким і налаштовуваним під конкретні вимоги проєкту.

5) інтерактивний відладчик. VS Code надає вбудовані засоби для налагодження коду, що дозволяє запускати, тестувати та відлагоджувати програми безпосередньо з інтерфейсу редактора. Це знижує час на виправлення помилок та спрощує процес розробки.

6) кросплатформеність. VS Code доступний для Windows, MacOS та Linux, що дозволяє розробникам працювати в будь-якому зручному для них середовищі.

7) інтелектуальні підказки та автодоповнення. VS Code має вбудовану підтримку автодоповнення коду (IntelliSense), що значно покращує продуктивність, дозволяючи швидко писати код і знаходити необхідні функції та методи.

8) консоль і термінал. У VS Code є вбудований термінал, що дозволяє виконувати команди безпосередньо в середовищі редактора.

До переваг Visual Studio Code відносять легкість у використанні, тобто інтуїтивно зрозумілий інтерфейс, що підходить як для новачків, так і для досвідчених розробників, підтримка багатьох мов, фреймворків та інструментів, а також доступність для всіх користувачів безкоштовно, з можливістю внесення змін у вихідний код [24].

До недоліків можна віднести невеликий обсяг функцій у порівнянні з повними IDE. Хоча VS Code є потужним редактором, він не містить усіх можливостей, які є в більш важких IDE, як-от Visual Studio або IntelliJ IDEA.

Ще одним недоліком є залежність від плагінів – для розширення функціональності потрібно додавати плагіни, що може створити додаткові складнощі у процесі налаштування [24].

Отже, в результаті аналізу було обрано систему управління базою даних SQLite, мову програмування Python та середовище розробки Visual Studio Code.

2.5 Висновки до розділу 2

Отже, у даному розділі було розроблено архітектуру системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації. Для цього було спроєктовано блок-схему, що демонструє принцип взаємодії клієнта із системою. Особлива увага приділялась реалізації механізмів автентифікації, де поєднання пароля та одноразового пароля (OTP) у межах мультифакторної аутентифікації дозволяє значно підвищити безпеку доступу до системи. Впровадження Zero-Knowledge Proof забезпечує можливість перевірки прав доступу без розкриття конфіденційних даних, що мінімізує ризики витоку критичної інформації.

Було обґрунтовано вибір інструментів реалізації: мови програмування Python, середовища розробки Visual Studio Code та СУБД SQLite. Окрему увагу приділено IPFS як основі для зберігання даних у розподіленому середовищі. Було реалізовано шифрування файлів перед їх передачею у мережу IPFS з використанням алгоритму Fernet, що забезпечує конфіденційність та цілісність інформації. Таким чином, розроблена система поєднує децентралізоване зберігання, криптографічний захист та сучасні підходи до автентифікації, що в сукупності створює надійне середовище для зберігання й контролю доступу до корпоративних даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ДЕЦЕНТРАЛІЗОВАНОЇ СИСТЕМИ ДЛЯ ЗБЕРІГАННЯ КОРПОРАТИВНИХ ДАНИХ

У цьому розділі наведено опис програмної реалізації децентралізованої системи для зберігання та управління корпоративними даними. Розроблено та детально описано графічний користувацький інтерфейс і програмну реалізацію системи, також надано інструкцію для користувача щодо роботи з системою. Проект був здійснений у середовищі Visual Studio Code з використанням СУБД SQLite.

3.1 Програмна реалізація системи для децентралізованого зберігання корпоративних даних

Основною метою даного підрозділу є опис програмної реалізації системи, що забезпечує надійне та безпечне децентралізоване зберігання корпоративних даних. Наведено структуру програмного забезпечення, розглянуто функціональні модулі та інтеграцію з технологіями Zero-Knowledge Proof і мультифакторною аутентифікацією.

Для початку було здійснено ініціалізацію головного вікна програми. Для цього було застосовано метод init(), підключено базу даних, ініціалізовано менеджер шифрування, а також задано початковий стан – без активного користувача.

```
def __init__(self, root, db):
    self.root = root
    self.root.title("Захищена система зберігання даних")
    self.root.geometry("800x800")
    self.root.resizable(False, False)

    self.db = db
    self.current_user = None
    self.encryption_manager = EncryptionManager()
```

Далі було реалізовано інтерфейс з вкладками для зручної взаємодії користувача з основними функціями програми. Спочатку створюється елемент

інтерфейсу типу Tabview (записник), який дозволяє організувати кілька екранів у вигляді вкладок в одному вікні. Далі створюються дві вкладки: «Вхід» та «Реєстрація», кожна з яких містить окремий інтерфейс для відповідних дій користувача. Вміст кожної вкладки налаштовується викликами окремих методів – setup_login_ui() для інтерфейсу входу та setup_register_ui() для інтерфейсу реєстрації. Після цього ініціалізується змінна app_frame, яка в майбутньому буде використовуватись для відображення основного інтерфейсу програми після успішної автентифікації користувача.

```
self.notebook = ctk.CTkTabview(root)
self.notebook.pack(fill='both', expand=True, padx=10, pady=10)

self.notebook.add("Вхід")
self.notebook.add("Реєстрація")

self.setup_login_ui()

self.setup_register_ui()

self.app_frame = None
```

Далі було реалізовано функцію setup_login_ui(), що відповідає за створення графічного інтерфейсу для входу користувача в систему. Спочатку створюється контейнер – фрейм, який розміщується у відповідній вкладці та займає всю доступну область з невеликими відступами. У верхній частині форми виводиться заголовок з назвою системи, далі послідовно додаються текстові мітки та поля для введення імені користувача, пароль (з маскуванням введених символів для безпеки) та коду автентифікації (згенерованного через застосунок Google Authenticator). Наприкінці розміщується кнопка «Увійти», яка при натисканні викликає метод self.login, що відповідає за обробку введених даних та перевірку автентичності користувача.

```
def setup_login_ui(self):
    login_frame = ctk.CTkFrame(self.notebook.tab("Вхід"))
    login_frame.pack(fill='both', expand=True, padx=20, pady=20)

    ctk.CTkLabel(login_frame, text="Захищена система зберігання даних", font=("Arial",
20)).pack(pady=20)
```

```
        ctk.CTkLabel(login_frame, text="Ім'я користувача:").pack(pady=5)
        self.login_username = ctk.CTkEntry(login_frame, width=300)
        self.login_username.pack(pady=5)

        ctk.CTkLabel(login_frame, text="Пароль:").pack(pady=5)
        self.login_password = ctk.CTkEntry(login_frame, width=300, show="*")
        self.login_password.pack(pady=5)

        ctk.CTkLabel(login_frame, text="Код автентифікації:").pack(pady=5)
        self.login_totp = ctk.CTkEntry(login_frame, width=300)
        self.login_totp.pack(pady=5)

        ctk.CTkButton(login_frame, text="Увійти", command=self.login,
    width=200).pack(pady=20)
```

Після цього було реалізовано функцію setup_register_ui(), щовідповідає за створення інтерфейсу реєстрації нового користувача у вкладці «Реєстрація». На початку ідентично до вікна «Вхід» створюється окремий фрейм, у якому послідовно додаються елементи управління: поле для введення імені користувача, поле для введення пароля з прихованим відображенням символів та поле для підтвердження пароля (щоб уникнути помилок при введенні). Після цього розміщується кнопка «Зареєструватися», яка при натисканні викликає метод self.register, що реалізує логіку перевірки введених даних та створення нового облікового запису.

Окремо створюється фрейм self.qr_frame, який буде використовуватись для відображення QR-коду при реєстрації, що є частиною налаштування двофакторної автентифікації.

```
def setup_register_ui(self):
        register_frame = ctk.CTkFrame(self.notebook.tab("Реєстрація"))
        register_frame.pack(fill='both', expand=True, padx=20, pady=20)

        ctk.CTkLabel(register_frame, text="Реєстрація нового користувача", font=("Arial",
    20)).pack(pady=20)

        ctk.CTkLabel(register_frame, text="Ім'я користувача:").pack(pady=5)
        self.reg_username = ctk.CTkEntry(register_frame, width=300)
        self.reg_username.pack(pady=5)

        ctk.CTkLabel(register_frame, text="Пароль:").pack(pady=5)
        self.reg_password = ctk.CTkEntry(register_frame, width=300, show="*")
        self.reg_password.pack(pady=5)
```

```
        ctk.CTkLabel(register_frame, text="Підтвердження пароля:").pack(pady=5)
        self.reg_confirm_password = ctk.CTkEntry(register_frame, width=300, show="*")
        self.reg_confirm_password.pack(pady=5)

        ctk.CTkButton(register_frame, text="Зареєструватися", command=self.register,
    width=200).pack(pady=20)

        self.qr_frame = ctk.CTkFrame(register_frame)
        self.qr_frame.pack(pady=10, fill='both', expand=True)
```

Було реалізовано функцію display_qr_code(), який відповідає за виведення QR-коду після успішної реєстрації користувача. Спочатку очищується фрейм, у якому відображається код, щоб прибрати попередній вміст. Потім генерується QR-код на основі імені користувача та секретного ключа TOTP, після чого зображення конвертується у формат, придатний для відображення в інтерфейсі. У фреймі з'являється інструкція для сканування, сам QR-код та відповідний секретний ключ. Наприкінці користувачу показується повідомлення з підтвердженням успішної реєстрації та вказівкою просканувати код у Google Authenticator. Це необхідно для налаштування двофакторної автентифікації.

```
def display_qr_code(self, username, totp_secret):
        for widget in self.qr_frame.winfo_children():
            widget.destroy()

        qr_img, _ = generate_qr_code(username, totp_secret)
        self.qr_photoimage = ImageTk.PhotoImage(qr_img)

        ctk.CTkLabel(self.qr_frame, text="Відскануйте QR-код за допомогою Google
    Authenticator:").pack(pady=5)
        qr_label = ctk.CTkLabel(self.qr_frame, text="")
        qr_label.pack(pady=5)
        qr_label.configure(image=self.qr_photoimage)

        ctk.CTkLabel(self.qr_frame, text=f"Секретний ключ: {totp_secret}").pack(pady=5)
        ctk.CTkLabel(self.qr_frame, text="Будь ласка, збережіть цей QR-код або секретний
    ключ надійно!").pack(pady=5)

        messagebox.showinfo("Успішна реєстрація",
                            "Реєстрація успішна! Відскануйте QR-код за допомогою додатку Google
    Authenticator.")
    def register(self):
```

Після цього було реалізовано логіку обробки реєстрації нового користувача. Спочатку з відповідних полів інтерфейсу зчитуються введені дані: ім'я

користувача, пароль і підтвердження паролю. Далі виконується перевірка на наявність обов’язкових полів – якщо будь-яке з них порожнє, система виводить повідомлення про помилку. Після цього здійснюється перевірка відповідності пароля та його підтвердження. Якщо вони не співпадають, реєстрація не продовжується. Нарешті, виконується перевірка, чи існує вже користувач із таким ім’ям у базі даних. Якщо так, користувач отримує повідомлення про помилку, і процес припиняється. Дані зареєстрованих користувачів хешуються та передаються у базу даних.

```
username = self.reg_username.get()
password = self.reg_password.get()
confirm_password = self.reg_confirm_password.get()
    if not username or not password:
  messagebox.showerror("Помилка", "Ім'я користувача та пароль є обов'язковими")
  return
    if password != confirm_password:
  messagebox.showerror("Помилка", "Паролі не співпадають")
  return

   if self.db.get_user_by_username(username):
  messagebox.showerror("Помилка", "Користувач з таким ім'ям вже існує")
  return
   self.db.add_user(username, hashed_password, totp_secret)
```

Далі було реалізовано логіку входу користувача. Спочатку зчитуються введені дані: ім’я користувача, пароль і код автентифікації (TOTP). Якщо будь-яке поле порожнє, з’являється повідомлення про помилку. Далі пароль хешується, і система перевіряє, чи існує користувач із такими обліковими даними. Якщо ні – вхід блокується. Після цього перевіряється правильність одноразового TOTP-коду. Якщо код недійсний, користувач також не може ввійти. Якщо всі перевірки проходять успішно, система зберігає дані авторизованого користувача і повідомляє про успішний вхід.

```
def login(self):
    username = self.login_username.get()
    password = self.login_password.get()
    totp_code = self.login_totp.get()

    if not username or not password or not totp_code:
      messagebox.showerror("Помилка", "Всі поля обов'язкові для заповнення")
```

```
            return

        hashed_password = hash_password(password)
        user = self.db.get_user_by_credentials(username, hashed_password)

        if not user:
            messagebox.showerror("Помилка", "Невірне ім'я користувача або пароль")
            return

        if not verify_totp(user[3], totp_code):  # totp_secret знаходиться в 4-му стовпці
            messagebox.showerror("Помилка", "Невірний код автентифікації")
            return

        self.current_user = user
        messagebox.showinfo("Успіх", "Вхід успішний!")
```

Далі було реалізовано функцію generate_totp_secret() для генерації секрету TOTP та функція verify_totp() для перевірки коду TOTP.

```
def generate_totp_secret():
    return pyotp.random_base32()

def verify_totp(secret, code):
    totp = pyotp.TOTP(secret)
    return totp.verify(code)
```

Для генерації QR-коду для Google Authenticator було застосовано функцію generate_qr_code().

```
def generate_qr_code(username, totp_secret, issuer_name="Захищене сховище даних"):
    totp_uri = pyotp.totp.TOTP(totp_secret).provisioning_uri(
        name=username,
        issuer_name=issuer_name
    )

    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    qr.add_data(totp_uri)
    qr.make(fit=True)
    qr_img = qr.make_image(fill_color="black", back_color="white")
    qr_img = qr_img.resize((200, 200))

    return qr_img, totp_uri
```

Далі було створено клас «ZKProof», який реалізує механізм доведення з нульовим розголошенням (Zero-Knowledge Proof) для перевірки прав доступу користувача до файлу без розкриття деталей про рівень доступу. При створенні об’єкта ініціалізується підключення до бази даних, ідентифікатори користувача та файлу, а також курсор для роботи з базою.

Функція check_direct_access() перевіряє наявність прямого права доступу у користувача до конкретного файлу. Якщо доступ є, generate_challenge() створює випадкове число – виклик, що використовується в протоколі доказу.

```
def __init__(self, conn, user_id, file_id):
    # Ініціалізація об'єкту доведення з нульовим розголошенням
    self.conn = conn
    self.user_id = user_id
    self.file_id = file_id
    self.cursor = conn.cursor()

  def check_direct_access(self):
    self.cursor.execute(
      "SELECT access_level FROM access_rights WHERE user_id = ? AND file_id = ?",
      (self.user_id, self.file_id)
    )
    result = self.cursor.fetchone()
    return result is not None

  def generate_challenge(self):
    return random.randint(1000000, 9999999)
```

Далі create_proof () формує доказ, який ґрунтується на хешуванні інформації про користувача, файл, рівень доступу та виклик. Таким чином доказ підтверджує право користувача без розкриття конкретних деталей.

Функція verify_proof () повторює створення хешу з тих же даних і порівнює його з отриманим доказом, що дозволяє перевірити достовірність доказу.

Функція verify_access () використовує цей клас для комплексної перевірки: спочатку визначає, чи має користувач прямий доступ, потім створює і перевіряє доказ. Якщо всі перевірки пройдені, вона повертає True, інакше – False.

```
def create_proof(self, challenge):
    self.cursor.execute(
      "SELECT access_level FROM access_rights WHERE user_id = ? AND file_id = ?",
      (self.user_id, self.file_id) )
```

```
        access_info = self.cursor.fetchone()

        if not access_info:
            return None

        commitment = f"{self.user_id}:{self.file_id}:{access_info[0]}:{challenge}"
        hashed_commitment = hashlib.sha256(commitment.encode()).hexdigest()

        return {
            "challenge": challenge,
            "commitment": hashed_commitment
        }
    def verify_proof(self, proof):
        if not proof:
            return False
        self.cursor.execute(
            "SELECT access_level FROM access_rights WHERE user_id = ? AND file_id = ?",
            (self.user_id, self.file_id)
        )
        access_info = self.cursor.fetchone()
        if not access_info:
            return False

        commitment = f"{self.user_id}:{self.file_id}:{access_info[0]}:{proof['challenge']}"
        hashed_commitment = hashlib.sha256(commitment.encode()).hexdigest()

        return hashed_commitment == proof["commitment"]

def verify_access(user_id, file_id, conn):
    zkp = ZKProof(conn, user_id, file_id)

    if zkp.check_direct_access():
        challenge = zkp.generate_challenge()
        proof = zkp.create_proof(challenge)
        return zkp.verify_proof(proof)
    return False
```

Також було реалізовано клас «Database» для інтеграції бази даних у систему. Для початку було ініціалізовано саму базу даних та створено три таблиці «users», «files» та «access_rights».

```
def __init__(self, db_path='secure_storage.db'):
        self.conn = sqlite3.connect(db_path)
        self.cursor = self.conn.cursor()
        self.setup_database()

    def setup_database(self):
        self.cursor.execute('''
```

```
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password TEXT NOT NULL,
            totp_secret TEXT NOT NULL
        )
        ''')

        self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS files (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            filename TEXT NOT NULL,
            cid TEXT NOT NULL,
            encryption_key TEXT NOT NULL,
            metadata TEXT,
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
        ''')

        self.cursor.execute('''
        CREATE TABLE IF NOT EXISTS access_rights (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            file_id INTEGER NOT NULL,
            user_id INTEGER NOT NULL,
            access_level TEXT NOT NULL,
            FOREIGN KEY (file_id) REFERENCES files (id),
            FOREIGN KEY (user_id) REFERENCES users (id)
        )
        ''')
```

Далі було реалізовано функцію get_user_by_credentials () для пошуку користувача в базі даних за заданим іменем користувача та хешем пароля, а також функцію get_user_by_username () для пошуку користувача за іменем.

```
    def get_user_by_credentials(self, username, hashed_password):
        self.cursor.execute(
            "SELECT * FROM users WHERE username = ? AND password = ?",
            (username, hashed_password)
        )
        return self.cursor.fetchone()

    def get_user_by_username(self, username):
        self.cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
        return self.cursor.fetchone()
```

Також було застосовано функції add_user, add_file та add_access_right для додавання нових записів про користувача, файл та права доступу у базу даних.

```
def add_user(self, username, hashed_password, totp_secret):
        self.cursor.execute(
          "INSERT INTO users (username, password, totp_secret) VALUES (?, ?, ?)",
          (username, hashed_password, totp_secret)
      )
      self.conn.commit()
      return self.cursor.lastrowid

    def add_file(self, user_id, filename, cid, encryption_key):
      self.cursor.execute(
        "INSERT INTO files (user_id, filename, cid, encryption_key) VALUES (?, ?, ?, ?)",
        (user_id, filename, cid, encryption_key)
      )
      self.conn.commit()
      return self.cursor.lastrowid

    def add_access_right(self, file_id, user_id, access_level):
      self.cursor.execute(
        "INSERT INTO access_rights (file_id, user_id, access_level) VALUES (?, ?, ?)",
        (file_id, user_id, access_level)
      )
      self.conn.commit()
```

Крім цього було реалізовано функції, які забезпечують отримання інформації про файли користувача: get_user_files повертає список файлів, що належать безпосередньо користувачу за його ідентифікатором; get_accessible_files отримує всі файли, до яких користувач має доступ, включно з рівнем цього доступу та ключем шифрування; get_owned_files повертає лише ті файли, якими користувач володіє як власник, тобто має найвищий рівень доступу.

```
def get_user_files(self, user_id):
      self.cursor.execute(
        "SELECT id, filename, cid FROM files WHERE user_id = ?",
        (user_id,)
      )
      return self.cursor.fetchall()

    def get_accessible_files(self, user_id):
      self.cursor.execute("""
        SELECT f.id, f.filename, f.cid, f.encryption_key, a.access_level
        FROM files f
        JOIN access_rights a ON f.id = a.file_id
        WHERE a.user_id = ?
      """, (user_id,))
      return self.cursor.fetchall()
```

```
def get_owned_files(self, user_id):
    self.cursor.execute("""
        SELECT f.id, f.filename FROM files f
        JOIN access_rights a ON f.id = a.file_id
        WHERE a.user_id = ? AND a.access_level = 'owner'
    """, (user_id,))
    return self.cursor.fetchall()
```

Отже, у цьому підрозділі було представлено основні фрагменти коду, які були реалізовані для створення системи для децентралізованого зберігання корпоративних даних, включаючи логіку реєстрації та входу, мультифакторну автентифікацію та базу даних.

3.2 Програмна реалізація впровадження IPFS у систему

Для забезпечення децентралізованості розробленої системи, було інтегровано IPFS Pinata, як допоміжний інструмент для зберігання зашифрованих файлів. При цьому сама Pinata не має доступу до змісту файлів, оскільки їх шифрування відбувається локально на стороні користувача перед завантаженням. Таким чином, навіть у разі компрометації хмарного сервісу, сторонні особи не зможуть отримати доступ до даних, що гарантує максимальний рівень конфіденційності без використання сторонніх готових рішень у критично важливих частинах функціоналу.

Спочатку було реалізовано функціонал завантаження файлу до системи. Користувач за допомогою діалогового вікна обирає файл. Якщо файл не обрано – процес припиняється. Далі зчитується вміст файлу у двійковому форматі, після чого відбувається його шифрування з використанням реалізованого менеджера шифрування. Зашифрований файл завантажується у IPFS, і якщо отримано CID (унікальний ідентифікатор файлу в IPFS), у базі даних зберігається інформація про файл, включно з CID та ключем шифрування. Користувачу призначається право власника файлу, а в інтерфейсі з’являється повідомлення про успішне завантаження. У разі помилки виводиться відповідне повідомлення.

```
def upload_file(self):
    file_path = filedialog.askopenfilename(title="Виберіть файл для завантаження")
    if not file_path:
```

```
            return
        file_name = os.path.basename(file_path)

        with open(file_path, "rb") as file:
            file_data = file.read()

        encrypted_data, file_key = self.encryption_manager.encrypt_file(file_data)

        try:
            cid = upload_to_ipfs(encrypted_data)

            if cid:
                file_id = self.db.add_file(
                    self.current_user[0],
                    file_name,
                    cid,
                    file_key.decode()
                )

                self.db.add_access_right(file_id, self.current_user[0], "owner")
                messagebox.showinfo("Успіх", f"Файл успішно завантажено!\nCID: {cid}")
                self.refresh_files_list()
            else:
                messagebox.showerror("Помилка", "Не вдалося завантажити файл в IPFS")
        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка завантаження: {str(e)}")
```

Функція download_file відповідає за завантаження обраного користувачем файлу з IPFS. Перед початком процесу перевіряється право доступу до файлу за допомогою протоколу нульового розголошення (ZKP). Якщо користувач не має дозволу на доступ, відображається повідомлення про помилку і завантаження не відбувається.

```
    def download_file(self, file_data):
        file_id, filename, cid, encryption_key, _ = file_data

        if not verify_access(self.current_user[0], int(file_id), self.db.conn):
            messagebox.showerror("Помилка", "Доступ заборонено")
            return
```

Далі було реалізовано логіку отримання зашифрованого файлу з IPFS за його CID, подальше його розшифрування локально за допомогою збереженого ключа, та збереження результату у файл, обраний користувачем. У разі помилки на будь-якому етапі виводиться відповідне повідомлення.

```
        try:
            encrypted_data = retrieve_from_ipfs(cid)

            if not encrypted_data:
                messagebox.showerror("Помилка", "Не вдалося отримати файл з IPFS")
                return

            decrypted_data = self.encryption_manager.decrypt_file(encrypted_data,
encryption_key.encode())

            save_path = filedialog.asksaveasfilename(
                defaultextension=".*",
                initialfile=filename
            )

            if save_path:
                with open(save_path, "wb") as f:
                    f.write(decrypted_data)
                messagebox.showinfo("Успіх", f"Файл успішно завантажено та розшифровано в
{save_path}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка завантаження: {str(e)}")
```

Для взаємодії системи з IPFS спочатку було додано ключі сховища у файл pinata_key.txt. Далі було реалізовано функцію read_pinata_credentials(), яка відкриває файл pinata_key.txt, читає його вміст і шукає рядки, які містять ключ API (API Key:), секретний ключ (API Secret:) та токен (JWT:).

```
def read_pinata_credentials():
    try:
        with open("./pinata_key.txt", "r") as file:
            content = file.read()
            api_key = None
            api_secret = None
            jwt = None

            for line in content.split("\n"):
                if "API Key:" in line:
                    api_key = line.split(":")[1].strip()
                elif "API Secret:" in line:
                    api_secret = line.split(":")[1].strip()
                elif "JWT:" in line:
                    jwt = line.split(":")[1].strip()

            return api_key, api_secret, jwt
    except Exception as e:
        print(f"Помилка зчитування облікових даних Pinata: {e}")
```

```
    return None, None, None
```

Функція upload_to_ipfs(data) виконує завантаження даних до IPFS. Спочатку зчитуються облікові дані з файлу за допомогою read_pinata_credentials(). Якщо не виявлено JWT-токен (ключ доступу), піднімається виняток, оскільки без нього взаємодія з API неможлива.

```
def upload_to_ipfs(data):
  api_key, api_secret, jwt = read_pinata_credentials()

  if not jwt:
    raise Exception("Не знайдено JWT токен ")

  with tempfile.NamedTemporaryFile(delete=False) as temp_file:
    temp_file.write(data)
    temp_file_path = temp_file.name
```

Функція retrieve_from_ipfs(cid) завантажує зашифровані дані з IPFS за вказаним CID. Спочатку намагаючись отримати файл через кілька публічних шлюзів. Якщо це не вдається, використовується Pinata API з JWT-авторизацією. У разі успіху повертаються байти файлу, інакше – None. Такий підхід забезпечує надійний та безпечний доступ до файлів у децентралізованій мережі.

```
def retrieve_from_ipfs(cid):
  try:
    for gateway in ["https://gateway.pinata.cloud", "https://cloudflare-ipfs.com",
"https://ipfs.io"]:
      try:
        response = requests.get(f"{gateway}/ipfs/{cid}", timeout=10)
        if response.status_code == 200:
          return response.content
      except requests.exceptions.RequestException:
        continue
    api_key, api_secret, jwt = read_pinata_credentials()
    if not jwt:
      raise Exception("Не знайдено JWT токен ")
    headers = {
      "Authorization": f"Bearer {jwt}"
    }

    url = f"https://api.pinata.cloud/pinning/pinJSONToIPFS/{cid}"
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
      response = requests.get(f"https://gateway.pinata.cloud/ipfs/{cid}")
```

```
            return response.content
        else:
            return None
    except Exception as e:
        print(f"Помилка отримання з IPFS: {e}")
        return None
```

Для шифрування та дешифрування файлів перед завантаженням до IPFS і після їх отримання було створено клас «EncryptionManager». У ньому було створено наступні функції: self.load_or_create_key() для завантаження або створення ключа шифрування; generate_file_key() – для генерації нового ключа для файлу; encrypt_file() – для шифрування даних файлу та decrypt_file() – для дешифрування.

```
class EncryptionManager:
    def __init__(self):
        self.load_or_create_key()

    def load_or_create_key(self):
        key_path = "encryption_key.key"
        if os.path.exists(key_path):
            with open(key_path, "rb") as key_file:
                self.encryption_key = key_file.read()
        else:
            self.encryption_key = Fernet.generate_key()
            with open(key_path, "wb") as key_file:
                key_file.write(self.encryption_key)

        self.cipher = Fernet(self.encryption_key)

    def generate_file_key(self):
        return Fernet.generate_key()

    def encrypt_file(self, file_data, file_key=None):
        if file_key is None:
            file_key = self.generate_file_key()
        file_cipher = Fernet(file_key)
        encrypted_data = file_cipher.encrypt(file_data)
        return encrypted_data, file_key

    def decrypt_file(self, encrypted_data, file_key):
        file_cipher = Fernet(file_key)
        decrypted_data = file_cipher.decrypt(encrypted_data)
        return decrypted_data
```

Таким чином, запропонований підхід забезпечує надійне симетричне шифрування файлів перед їх збереженням у децентралізованому сховищі. Завдяки локальній генерації та зберіганню ключів, кожен файл має окремий ключ, що значно підвищує рівень безпеки. Навіть у разі компрометації CID або доступу до IPFS, сторонній користувач не зможе розшифрувати файл без відповідного ключа, який зберігається лише локально у захищеному вигляді. Це унеможливлює несанкціонований доступ до вмісту файлів і забезпечує надійний захист конфіденційної інформації в системі.

3.3 Інструкція користувача по роботі з розробленою системою

При запуску програми відображається основні форма, яка містить вкладки «Вхід» та «Реєстрація». Для реєстрації нового користувача необхідно ввести ім’я користувача та пароль (рис. 3.1).

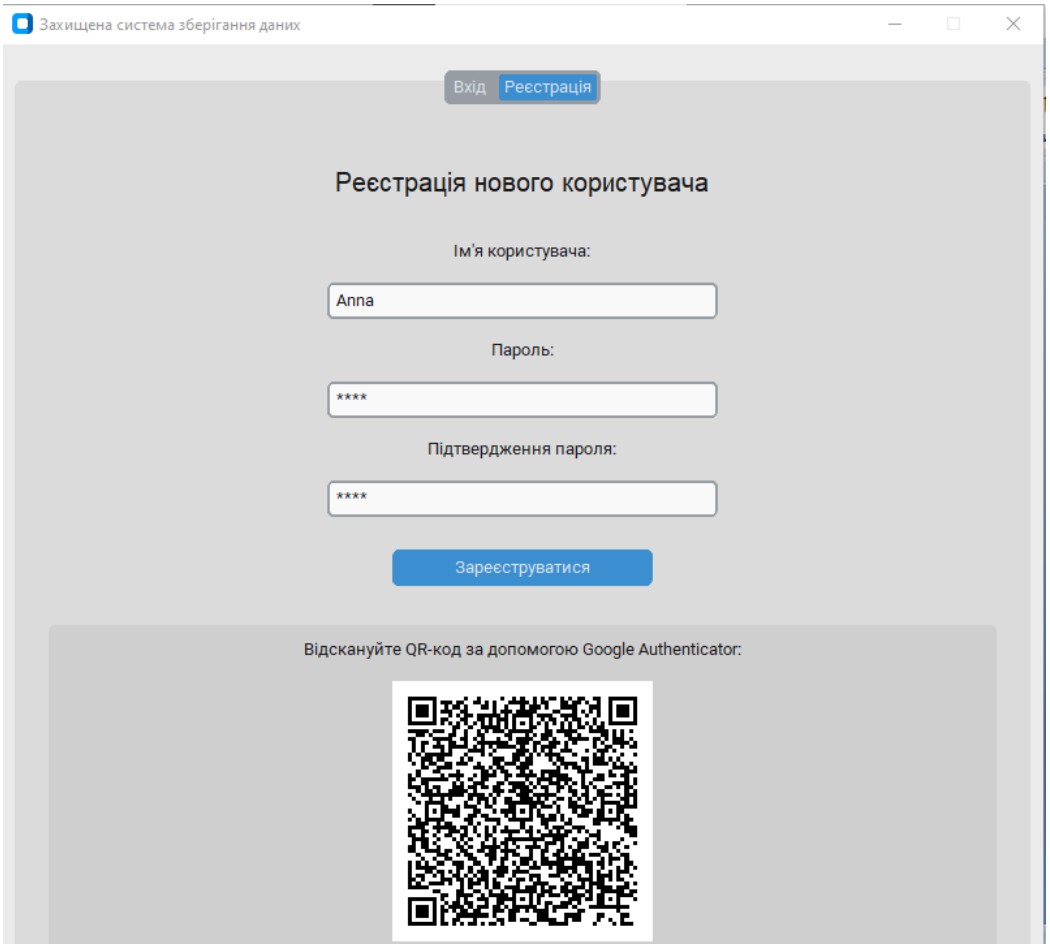


Рисунок 3.1 – Інтерфейс вікна реєстрації

Після натиснення кнопки «Зареєструватись» відображається повідомлення про успішну реєстрацію та QR-код, який необхідно відсканувати за допомогою додатку Google Authenticator.

Після успішної реєстрації необхідно перекти у вкладку «Вхід», ввести ім’я користувача, пароль, та одноразовий пароль, який генерується у додатку Google Authenticator. Ключовим моментом є те, що одноразові паролі дійсні протягом декількох секунд та постійно оновлюються, що запобігає несанкціонованому доступу до системи. Після успішного входу виводиться відповідне повідомлення (рис. 3.2).

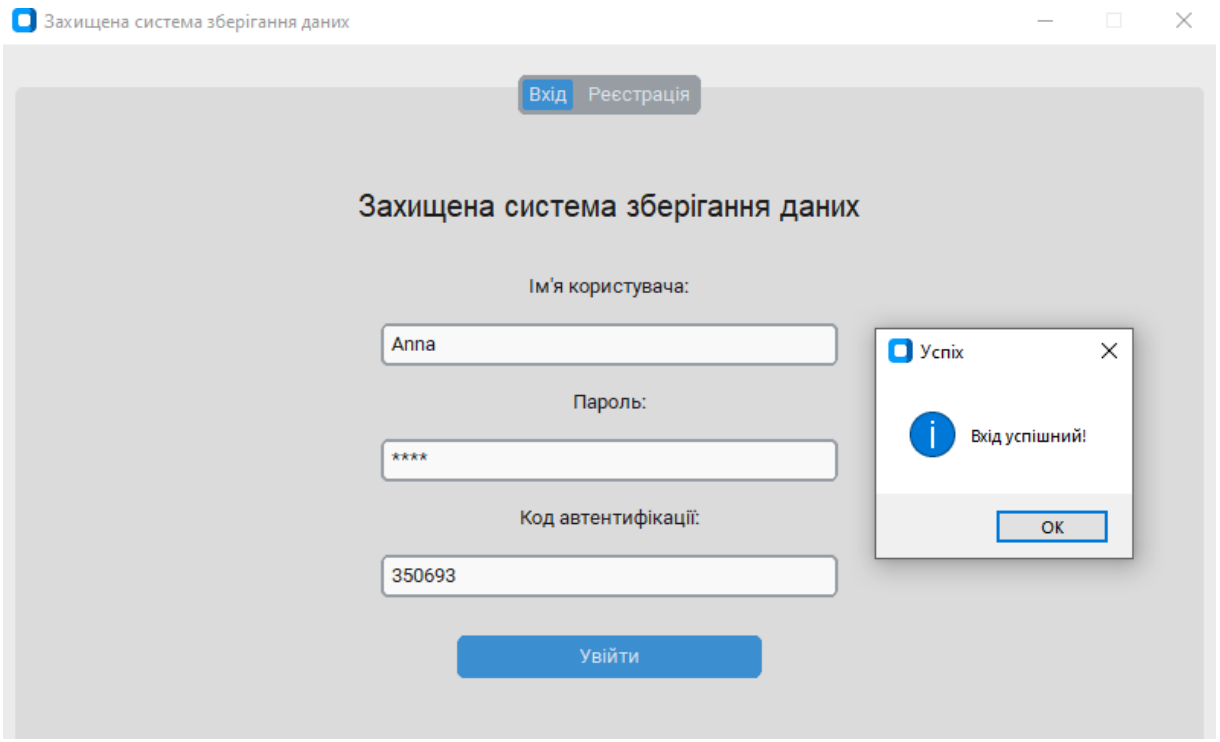


Рисунок 3.2 – Інтерфейс вікна входу

У разі неправильного введення одного з компонентів вікна входу, виводиться відповідне повідомлення про помилку, після чого користувачу необхідно повторити спробу (рис. 3.3).

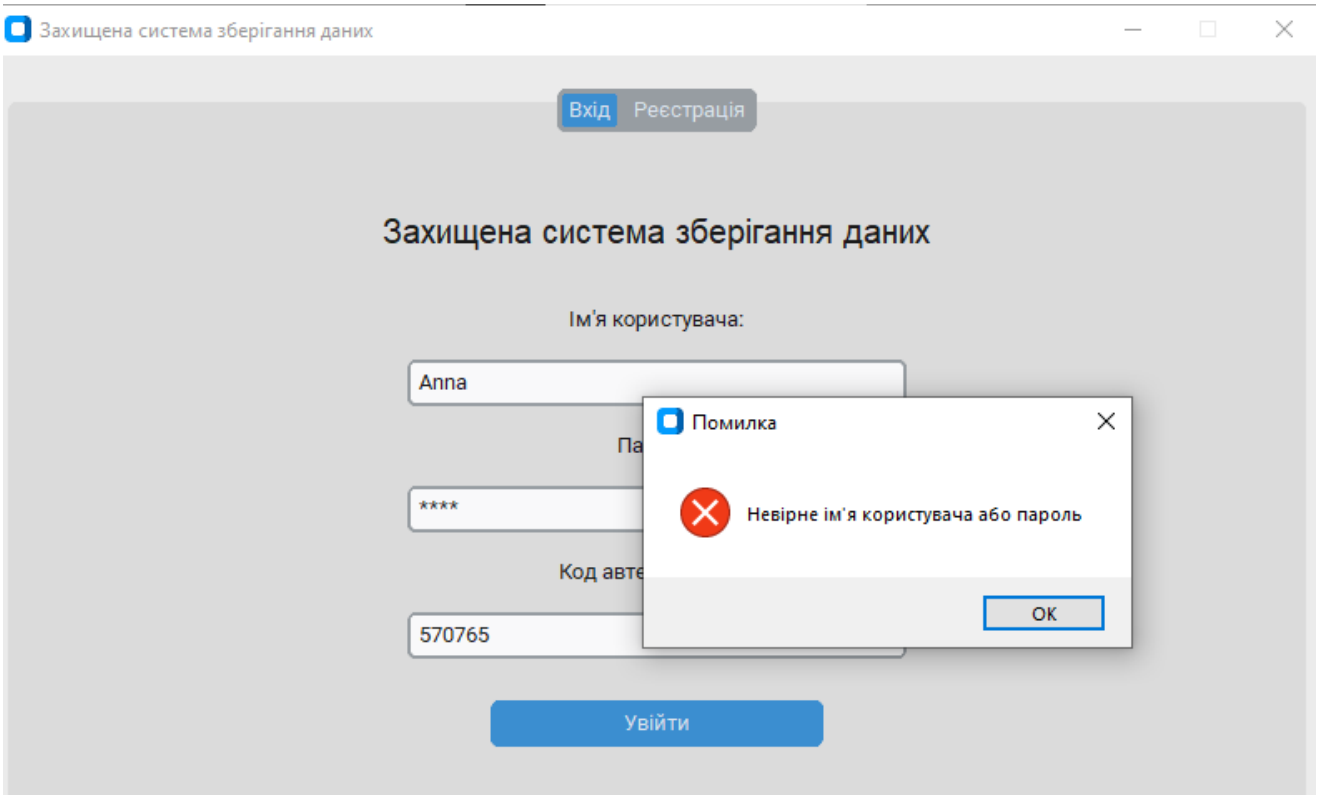


Рисунок 3.3 – Результат введення хибних облікових даних

Після успішного входу в систему відображається головне вікно (рис. 3.4).

Рисунок 3.4 – Інтерфейс головного вікна програми

Для завантаження файлу необхідно натиснути кнопку «Завантажити файл», далі обрати необхідний файл. Після успішного завантаження виводиться відповідне повідомлення та CID файлу, а завантажений файл відображається на головній сторінці (рис. 3.5).

Рисунок 3.5 – Результат завантаження файлу в систему

Щоб скачати файл, необхідно натиснути на кнопку «Скачати файл» та обрати необхідний файл у відповідному вікні (рис. 3.6).

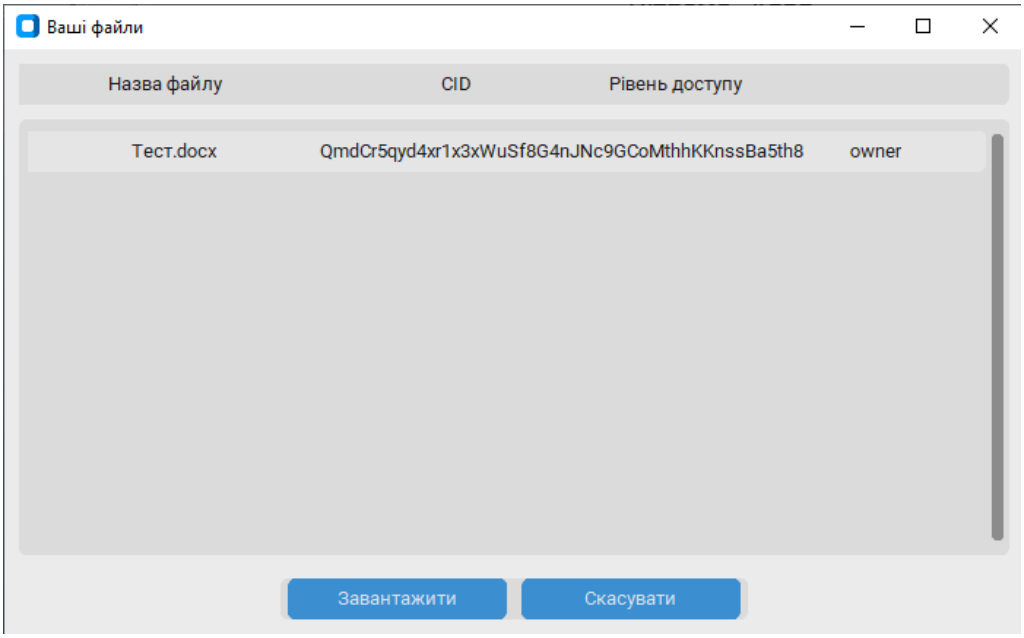


Рисунок 3.6 – Інтерфейс вікна вибору файлів для завантаження

Після натиснення кнопки «Завантажити» необхідно обрати місце для зберігання.

Окрім цього було реалізовано можливість поширення файлу між користувачами. Для цього необхідно натиснути на кнопку «Поділитися файлом» та у відповідному вікні обрати файл, яким користувач бажає поділитися, ввести ім’я користувача для спільного доступу та надати рівень доступу – тільки читання або читання і запис (рис. 3.7).

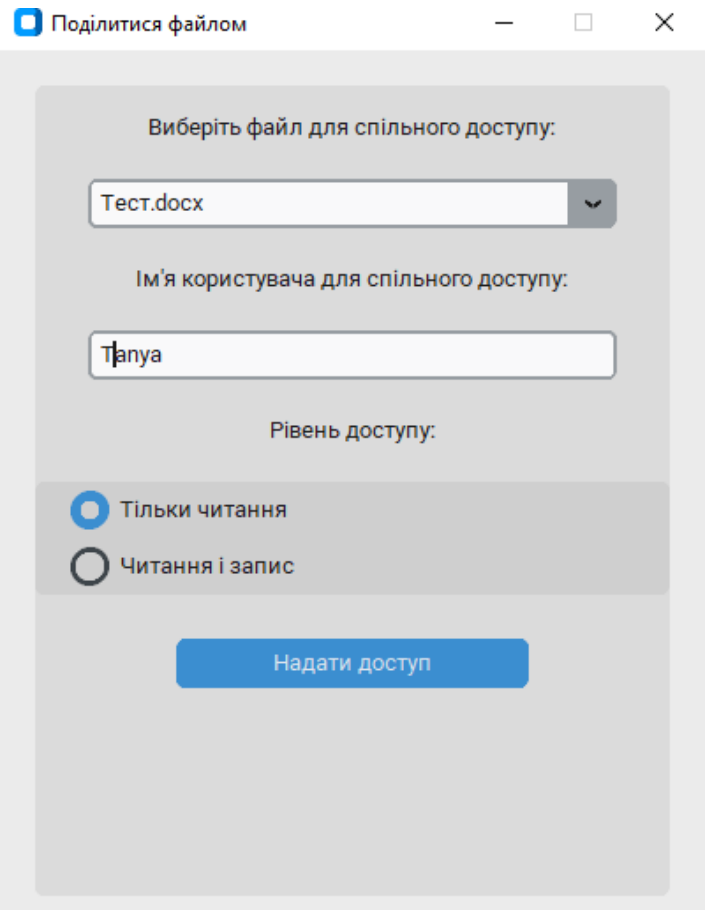


Рисунок 3.7 – Інтерфейс вкладки «Поділитися доступом»

Після цього у користувача, якому надали спільний доступ відповідний файл стає доступним для завантаження (рис. 3.8).

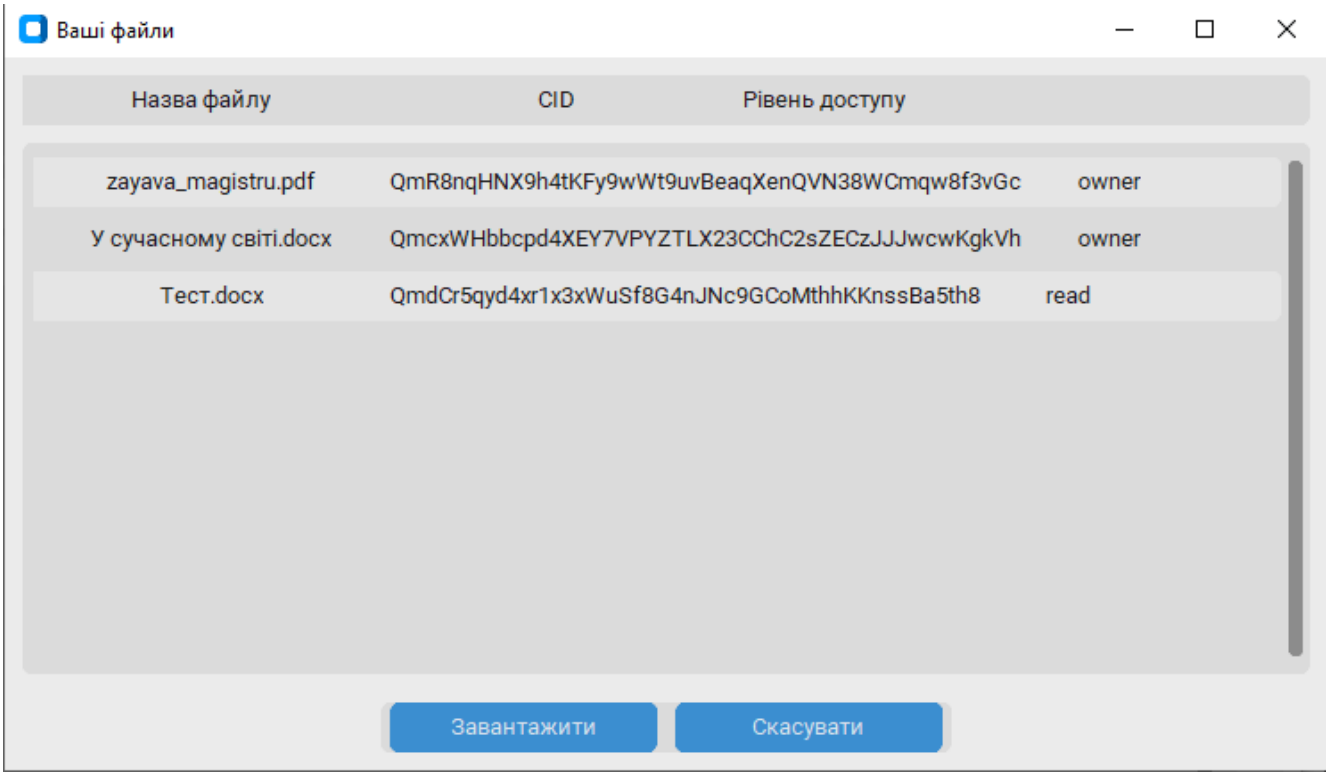


Рисунок 3.8 – Інтерфейс вкладки «Скачати» у користувача, якому було надано спільний доступ

Для виходу з облікового запису необхідно натиснути кнопку «Вийти».

3.4 Висновки до розділу 3

У даному розділі було здійснено розробку захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації. Основну увагу приділено реалізації механізму зберігання файлів у мережі IPFS з попереднім локальним шифруванням, що забезпечує високий рівень конфіденційності. Перевірка прав доступу реалізована за допомогою протоколу нульового розголошення, що гарантує безпеку автентифікації без розкриття критичної інформації. Також було розроблено зрозумілий та інтуїтивний інтерфейс. Для зручності роботи з системою було створено інструкцію для користувачів. Запропоноване рішення забезпечує надійний захист корпоративних даних навіть у разі несанкціонованого доступу до інфраструктури сховища.

ВИСНОВКИ

У даній дипломній роботі було розроблено захищену децентралізовану систему для зберігання та управління корпоративними даними із застосуванням IPFS, протоколу Zero-Knowledge Proof та мультифакторної автентифікації. Проведено комплексний аналіз сучасних підходів до зберігання даних, у тому числі централізованих та децентралізованих рішень, а також існуючих методів автентифікації, що дозволило виявити недоліки традиційних моделей безпеки та визначити доцільність використання децентралізованих технологій.

У другому розділі розроблено архітектуру захищеної системи, в якій були запропоновані механізми забезпечення конфіденційності та безпечної автентифікації. Інтеграція IPFS як базового компонента децентралізованого зберігання дозволила забезпечити масштабованість, відмовостійкість та незалежність від централізованих провайдерів. Застосування Zero-Knowledge Proof забезпечує перевірку прав доступу без розкриття додаткової інформації, а MFA зміцнює надійність автентифікації.

У третьому розділі було реалізовано повноцінний функціонал системи: завантаження файлів з попереднім симетричним шифруванням, їх збереження в IPFS, контроль прав доступу та завантаження з подальшим розшифруванням. Збереження ключів шифрування здійснюється локально, що унеможливлює компрометацію вмісту навіть при доступі до CID. Забезпечено зручний інтерфейс користувача та передбачено обробку помилок на ключових етапах взаємодії з системою.

Таким чином, реалізована система відповідає сучасним вимогам до захищеного управління корпоративними даними, демонструє переваги децентралізації та забезпечує високий рівень безпеки без залежності від сторонніх провайдерів зберігання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Main Issues in Big Data Security. *MDPI:* веб-сайт. URL: https://www.mdpi.com/1999-5903/8/3/44 (дата звернення: 05.05.2025).

2. Big Security for Big Data: Addressing Security Challenges for the Big Data Infrastructure. *Springer nature:* веб-сайт. URL: https://link.springer.com/chapter/10.1007/978-3-319-06811-4_13 (дата звернення: 05.05.2025).

3. Centralized vs. Decentralized Data: How to Choose One Over the Other. *Linkedin:* веб-сайт. URL: https://www.linkedin.com/pulse/centralized-vs-decentralized-data-how-choose-one-over-laurence-adam/ (дата звернення: 05.05.2025).

4. Centralized vs. Decentralized Data: Unveiling the Great Debate. *inclusioncloud:* веб-сайт. URL: https://inclusioncloud.com/insights/blog/centralized-decentralized-data/ (дата звернення: 05.05.2025).

5. Laydip Sen. Computer and Network Security. London, 2020, 176 p.

6. Monther Aldwairi, Saoud Aldhanhani. Multi-Factor Authentication System. Conference: International Conference on Research and Innovation in Computer Engineering and Computer Sciences (RICCES). At: Langkawi, Malaysia, 2017, 8 p.

7. Rashad Mahmood Saqib, Adnan Shahid Khan1, Yasir Javed, Shakil Ahmad, Kashif Nisar, Irshad A. Abbasi, Muhammad Reazul Haque, Azlina Ahmadi Julaihi. Analysis and Intellectual Structure of the Multi-Factor Authentication in Information Security. City of the Samaritans, Malaysia, 2021, P. 1633-1647.

8. Top Decentralized Storage Solutions for Data Security and Privacy. *Hivenet.* URL: https://www.hivenet.com/post/top-decentralized-storage-solutions-for-data-security-and-privacy (дата звернення 07.05.2025).

9. Barbara Guidi, Andrea Michienzi, Laura Ricci. Evaluating the decentralisation of filecoin. Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good, 2022, P. 13-18.

10. Sammy de Figueiredo, Akash Madhusudan, Vincent Reniers, Svetla Nikova, Bart Preneel. Exploring the storj network: a security analysis. Proceedings of the 36th Annual ACM Symposium on Applied Computing, 2021, P. 257-264.

11. Що таке блок-схема? *Dropbox:* веб-сайт. URL: https://experience.dropbox.com/uk-ua/resources/flowcharts (дата звернення: 08.05.2025).

12. John Edward Silva. An Overview of Cryptographic Hash Functions and Their Uses. SANS Institute, 2003, 11 p.

13. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Vincent Rijmen. Analysis of Step-Reduced SHA-256. Graz University of Technology, Austria, 2011, P. 245-260.

14. Zero-knowledge proofs explained in 3 examples. *Circularise:* веб-сайт. URL: https://www.circularise.com/blogs/zero-knowledge-proofs-explained-in-3-examples (дата звернення: 10.05.2025).

15. Скидан Т. М., Грицак А. В. Побудова довірчих відносин у децентралізованих системах за допомогою ZKP та MFA. *Молодь в науці: дослідження, проблеми, перспективи*: матеріали Міжнар. наук.-практ. конф., м. Вінниця, 2025р. 4с.

16. OTP-Based Two-Factor Authentication Using Mobile Phones. *IEEE Xplore:* веб-сайт. URL: https://ieeexplore.ieee.org/abstract/document/5945255 (дата звернення: 15.05.2025).

17. Neeta P Patil, Yogita D Mane, Anil Vasoya, Akshay Agrawal, Sanketi Raut. Secure File Sharing Using Blockchain and IPFS with Smart Contract-Based Access Control. Journal of Information Systems Engineering & Management, 2025, P. 184-194.

18. Застосування UML. Діаграма послідовності – Sequence Diagram. *DUT.EDU*: веб-сайт. URL: https://dut.edu.ua/ua/news-1-626-7897-zastosuvannya-uml-chastina-2-diagrama-poslidovnosti---sequence-diagram_kafedra-kompyuternih-nauk-ta-informaciynih-tehnologiy (дата звернення: 15.05.2025).

19. Fernet Symmetric Encryption Using a Cryptography Module in Python. *tutorialspoint:* веб-сайт. URL: https://www.tutorialspoint.com/fernet-symmetric-encryption-using-a-cryptography-module-in-python (дата звернення: 16.05.2025).

20. S.S.Tyagi. Enhancing Security of Cloud Data through Encryption with AES and Fernet Algorithm through Convolutional-Neural-Networks (CNN). International Journal of Computer Networks and Applications (IJCNA), India, 2021, 12 p.

21. Grant Allen and Mike Owens. The Definitive Guide to SQLite, Second Edition. Apress, 2010, 369 p.

22. ІЩо таке SQlite? *freehost.ua:* веб-сайт. URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-sqlite/ (дата звернення: 17.05.2025).

23. Comparison of 10 Programming Languages. *medium:* веб-сайт. URL: https://reubenrochesingh.medium.com/comparison-of-10-programming-languages-f43b0ac337a4 (дата звернення: 17.05.2025).

24. Why did we build Visual Studio Code? *Visual Studio Code:* веб-сайт. URL: https://code.visualstudio.com/docs/editor/whyvscode (дата звернення: 14.05.2025).

ДОДАТКИ

Додаток А. Технічне завдання

Додаток А. Технічне завдання

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

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ
до бакалаврської кваліфікаційної роботи на тему:
Розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації

08-72.БКР.016.00.069.ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент кафедри МБІС
_____ Грищак А. В.
"____"_____

Вінниця – 2025 р.

1. Найменування та область застосування

Програмний засіб захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації. Область застосування: інформаційна безпека корпоративних інформаційних систем, що потребують захищеного зберігання та контролю доступу до конфіденційних даних у децентралізованому середовищі.

2.Підстава для розробки

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

3. Мета та призначення розробки

3.1 Створення програмного засобу, що забезпечує безпечне зберігання та управління корпоративними даними в децентралізованому середовищі шляхом інтеграції IPFS для розподіленого зберігання, Zero-Knowledge Proof для захисту конфіденційності авторизації та мультифакторної автентифікації для підвищення рівня контролю доступу.

3.2 Призначення: розроблений програмний засіб забезпечує децентралізоване зберігання файлів з попереднім шифруванням, захист доступу до них за допомогою мультифакторної автентифікації та перевірки прав доступу без розкриття конфіденційної інформації за допомогою Zero-Knowledge Proof.

4. Джерела розробки

4.1. John Edward Silva. An Overview of Cryptographic Hash Functions and Their Uses. SANS Institute, 2003, 11 p.

4.2. Florian Mendel, Norbert Pramstaller, Christian Rechberger, Vincent Rijmen. Analysis of Step-Reduced SHA-256. Graz University of Technology, Austria, 2011, P. 245-260.

4.3. John Edward Silva. An Overview of Cryptographic Hash Functions and Their Uses. SANS Institute, 2003, 11 p.

4.4. S.S.Tyagi. Enhancing Security of Cloud Data through Encryption with AES and Fernet Algorithm through Convolutional-Neural-Networks (CNN). International Journal of Computer Networks and Applications (IJCNA), India, 2021, 12 p.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний та інтуїтивно зрозумілий інтерфейс користувача для завантаження, зберігання та доступу до файлів.

5.1.2 Реалізація програмного засобу повинна бути можливою без використання спеціалізованого ліцензійного програмного забезпечення.

5.1.3 Доступ до файлів повинен відбуватись лише після проходження перевірки на основі Zero-Knowledge Proof.

5.1.4 Програмний засіб має забезпечувати завантаження файлів у розподілену мережу IPFS та зберігати відповідні хеш-ідентифікатори.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати стабільно без критичних збоїв.

5.2.2 Програмний засіб повинен забезпечувати шифрування даних перед зберіганням у IPFS та збереження криптографічних ключів у захищеному середовищі.

5.2.3 Програмний засіб повинен забезпечувати збереження цілісності та конфіденційності даних при взаємодії з IPFS.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– відповідність чинним вимогам та стандартам щодо роботи з комп’ютерною технікою та програмними засобами.

6. Вимоги до програмної документації

6.1 Обов’язкова наявність поетапної інструкції користувача щодо роботи з системою, наведена у підрозділі 3.3 дипломної роботи.

7. Вимоги до технічного захисту інформації

7.1 Програмний засіб повинен бути захищений від несанкціонованого доступу шляхом впровадження автентифікації з використанням MFA.

7.2 Доступ до інформаційних ресурсів мають отримувати лише зареєстровані та автентифіковані користувачі.

7.3 Дані мають бути зашифровані перед передачею у IPFS, що унеможливлює їх прочитання третіми особами.

8. Техніко-економічні показники

8.1 Очікувана користь від впровадження системи перевищує витрати на її реалізацію.

8.2 Розроблений програмний засіб має бути простим в установці, що забезпечує можливість його використання малими та середніми підприємствами без необхідності в складній інфраструктурі.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Початок	Закінчення
1.	Визначення напрямку бакалаврської роботи, формулювання теми	21.03.2025	24.03.2025
2.	Аналіз предметної області обраної теми	26.03.2025	27.03.2025
3.	Розробка алгоритму роботи	29.03.2025	11.04.2025
4.	Написання бакалаврської роботи на основі розробленої теми	12.04.2025	22.04.2025
5.	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6.	Виправлення, уточнення, корегування бакалаврської кваліфікаційної роботи	28.05.2025	03.06.2025
7.	Захист бакалаврської кваліфікаційної роботи	11.06.2025	12.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

– демонстрація результату бакалаврської кваліфікаційної роботи;

– презентація;

– відзив керівника роботи;

– відзив рецензента.

Технічне завдання до виконання прийняв _____Скидан Т.М.

Додаток Б. Лістинг програми

```
class SecureDataApp:
    def __init__(self, root, db):
        # Ініціалізація головного вікна додатку
        self.root = root
        self.root.title("Захищена система зберігання даних")
        self.root.geometry("800x800")
        self.root.resizable(False, False)

        self.db = db
        self.current_user = None
        self.encryption_manager = EncryptionManager()

        # Створення записника для різних екранів
        self.notebook = ctk.CTkTabview(root)
        self.notebook.pack(fill='both', expand=True, padx=10, pady=10)

        # Створення вкладок
        self.notebook.add("Вхід")
        self.notebook.add("Реєстрація")

        # Налаштування UI для входу
        self.setup_login_ui()

        # Налаштування UI для реєстрації
        self.setup_register_ui()

        # Основний фрейм додатку (спочатку прихований)
        self.app_frame = None

    def setup_login_ui(self):
        # Налаштування інтерфейсу для входу
        login_frame = ctk.CTkFrame(self.notebook.tab("Вхід"))
        login_frame.pack(fill='both', expand=True, padx=20, pady=20)

        ctk.CTkLabel(login_frame, text="Захищена система зберігання даних", font=("Arial",
20)).pack(pady=20)

        ctk.CTkLabel(login_frame, text="Ім'я користувача:").pack(pady=5)
        self.login_username = ctk.CTkEntry(login_frame, width=300)
        self.login_username.pack(pady=5)

        ctk.CTkLabel(login_frame, text="Пароль:").pack(pady=5)
        self.login_password = ctk.CTkEntry(login_frame, width=300, show="*")
        self.login_password.pack(pady=5)

        ctk.CTkLabel(login_frame, text="Код автентифікації:").pack(pady=5)
```

```
        self.login_totp = ctk.CTkEntry(login_frame, width=300)
        self.login_totp.pack(pady=5)

        ctk.CTkButton(login_frame, text="Увійти", command=self.login,
width=200).pack(pady=20)

    def setup_register_ui(self):
        # Налаштування інтерфейсу для реєстрації
        register_frame = ctk.CTkFrame(self.notebook.tab("Реєстрація"))
        register_frame.pack(fill='both', expand=True, padx=20, pady=20)

        ctk.CTkLabel(register_frame, text="Реєстрація нового користувача", font=("Arial",
20)).pack(pady=20)

        ctk.CTkLabel(register_frame, text="Ім'я користувача:").pack(pady=5)
        self.reg_username = ctk.CTkEntry(register_frame, width=300)
        self.reg_username.pack(pady=5)

        ctk.CTkLabel(register_frame, text="Пароль:").pack(pady=5)
        self.reg_password = ctk.CTkEntry(register_frame, width=300, show="*")
        self.reg_password.pack(pady=5)

        ctk.CTkLabel(register_frame, text="Підтвердження пароля:").pack(pady=5)
        self.reg_confirm_password = ctk.CTkEntry(register_frame, width=300, show="*")
        self.reg_confirm_password.pack(pady=5)

        ctk.CTkButton(register_frame, text="Зареєструватися", command=self.register,
width=200).pack(pady=20)

        # Фрейм для відображення QR-коду
        self.qr_frame = ctk.CTkFrame(register_frame)
        self.qr_frame.pack(pady=10, fill='both', expand=True)

    def setup_app_ui(self):
        # Видалення вкладок логіну та реєстрації
        self.notebook.delete("Вхід")
        self.notebook.delete("Реєстрація")

        # Додавання вкладки додатку
        self.notebook.add("Додаток")
        app_tab = self.notebook.tab("Додаток")

        self.app_frame = ctk.CTkFrame(app_tab)
        self.app_frame.pack(fill='both', expand=True, padx=20, pady=20)

        ctk.CTkLabel(self.app_frame, text=f"Вітаємо, {self.current_user[1]}", font=("Arial",
20)).pack(pady=10)
```

```
        # Створення фрейму для кнопок
        buttons_frame = ctk.CTkFrame(self.app_frame)
        buttons_frame.pack(pady=20)

        # Створення кнопок з відповідними функціями
        ctk.CTkButton(buttons_frame, text="Завантажити файл", command=self.upload_file,
width=150).pack(side='left', padx=10)
        ctk.CTkButton(buttons_frame, text="скачати файл", command=self.show_files,
width=150).pack(side='left', padx=10)
        ctk.CTkButton(buttons_frame, text="Поділитися файлом",
command=self.show_sharing_options, width=150).pack(side='left', padx=10)
        ctk.CTkButton(buttons_frame, text="Вийти", command=self.logout,
width=150).pack(side='left', padx=10)

        # Фрейм зі списком файлів
        self.files_frame = ctk.CTkFrame(self.app_frame)
        self.files_frame.pack(pady=10, fill='both', expand=True)
        ctk.CTkLabel(self.files_frame, text="Ваші файли:", font=("Arial", 16)).pack(pady=5)

        # Створення фрейму для таблиці файлів
        files_tree_frame = ctk.CTkFrame(self.files_frame)
        files_tree_frame.pack(fill='both', expand=True, padx=10, pady=10)

        # Створення заголовків таблиці
        header_frame = ctk.CTkFrame(files_tree_frame)
        header_frame.pack(fill='x', pady=(0, 5))
        ctk.CTkLabel(header_frame, text="Назва файлу", width=300).pack(side='left', padx=(5,
0))
        ctk.CTkLabel(header_frame, text="CID", width=300).pack(side='left', padx=(5, 0))

        # Створення прокручуваного фрейму для списку файлів
        self.files_list_frame = ctk.CTkScrollableFrame(files_tree_frame, height=300)
        self.files_list_frame.pack(fill='both', expand=True)

        self.refresh_files_list()

    def refresh_files_list(self):
        # Очищення існуючих елементів
        for widget in self.files_list_frame.winfo_children():
            widget.destroy()

        # Отримання файлів для поточного користувача
        files = self.db.get_user_files(self.current_user[0])

        if not files:
            ctk.CTkLabel(self.files_list_frame, text="Файли не знайдено").pack(pady=10)
            return
```

```
        # Відображення файлів у списку
        for i, file in enumerate(files):
            file_id, filename, cid = file
            file_frame = ctk.CTkFrame(self.files_list_frame)
            file_frame.pack(fill='x', pady=2)

            ctk.CTkLabel(file_frame, text=filename, width=300).pack(side='left')
            ctk.CTkLabel(file_frame, text=cid, width=300).pack(side='left')

            # Для зберігання file_id з кожним рядком
            file_frame.file_id = file_id

            # Додавання кольору смугами для кращої читабельності
            if i % 2 == 0:
                file_frame.configure(fg_color=("gray90",  "gray20"))

    def display_qr_code(self, username, totp_secret):
        # Очищення QR фрейму
        for widget in self.qr_frame.winfo_children():
            widget.destroy()

        # Генерація QR-коду
        qr_img, _ = generate_qr_code(username, totp_secret)

        # Конвертація до PhotoImage
        self.qr_photoimage = ImageTk.PhotoImage(qr_img)

        # Відображення QR-коду
        ctk.CTkLabel(self.qr_frame, text="Відскануйте QR-код за допомогою Google
Authenticator:").pack(pady=5)
        qr_label = ctk.CTkLabel(self.qr_frame, text="")
        qr_label.pack(pady=5)
        # Встановлення зображення на мітку
        qr_label.configure(image=self.qr_photoimage)

        ctk.CTkLabel(self.qr_frame, text=f"Секретний ключ: {totp_secret}").pack(pady=5)
        ctk.CTkLabel(self.qr_frame, text="Будь ласка, збережіть цей QR-код або секретний
ключ надійно!").pack(pady=5)

        messagebox.showinfo("Успішна реєстрація",
                            "Реєстрація успішна! Відскануйте QR-код за допомогою додатку Google
Authenticator.")

    def register(self):
        # Обробка реєстрації користувача
        username = self.reg_username.get()
        password = self.reg_password.get()
        confirm_password = self.reg_confirm_password.get()
```

```
    if not username or not password:
        messagebox.showerror("Помилка", "Ім'я користувача та пароль є обов'язковими")
        return

    if password != confirm_password:
        messagebox.showerror("Помилка", "Паролі не співпадають")
        return

    # Перевірка чи користувач вже існує
    if self.db.get_user_by_username(username):
        messagebox.showerror("Помилка", "Користувач з таким ім'ям вже існує")
        return

    # Генерація секрету TOTP
    totp_secret = generate_totp_secret()
    hashed_password = hash_password(password)

    # Вставка користувача в базу даних
    self.db.add_user(username, hashed_password, totp_secret)

    # Відображення QR-коду для Google Authenticator
    self.display_qr_code(username, totp_secret)

def login(self):
    # Обробка входу користувача
    username = self.login_username.get()
    password = self.login_password.get()
    totp_code = self.login_totp.get()

    if not username or not password or not totp_code:
        messagebox.showerror("Помилка", "Всі поля обов'язкові для заповнення")
        return

    # Перевірка облікових даних користувача
    hashed_password = hash_password(password)
    user = self.db.get_user_by_credentials(username, hashed_password)

    if not user:
        messagebox.showerror("Помилка", "Невірне ім'я користувача або пароль")
        return

    # Перевірка коду TOTP
    if not verify_totp(user[3], totp_code):  # totp_secret знаходиться в 4-му стовпці
        messagebox.showerror("Помилка", "Невірний код автентифікації")
        return

    # Успішний вхід
```

```
        self.current_user = user
        messagebox.showinfo("Успіх", "Вхід успішний!")

        # Налаштування інтерфейсу основного додатку
        self.setup_app_ui()

    def logout(self):
        # Вихід користувача
        self.current_user = None

        # Видалення вкладки додатку і додавання вкладок входу та реєстрації
        self.notebook.delete("Додаток")
        self.notebook.add("Вхід")
        self.notebook.add("Реєстрація")

        # Очищення полів входу
        self.setup_login_ui()
        self.setup_register_ui()

    def upload_file(self):
        # Завантаження файлу
        file_path = filedialog.askopenfilename(title="Виберіть файл для завантаження")
        if not file_path:
            return

        file_name = os.path.basename(file_path)

        # Зчитування файлу та його шифрування
        with open(file_path, "rb") as file:
            file_data = file.read()

        # Шифрування файлу
        encrypted_data, file_key = self.encryption_manager.encrypt_file(file_data)

        # Завантаження в IPFS
        try:
            cid = upload_to_ipfs(encrypted_data)

            if cid:
                # Зберігання інформації про файл у базі даних
                file_id = self.db.add_file(
                    self.current_user[0],
                    file_name,
                    cid,
                    file_key.decode()
                )

                # Додавання права доступу для власника
```

```
            self.db.add_access_right(file_id, self.current_user[0], "owner")

            messagebox.showinfo("Успіх", f"Файл успішно завантажено!\nCID: {cid}")
            self.refresh_files_list()
        else:
            messagebox.showerror("Помилка", "Не вдалося завантажити файл в IPFS")
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка завантаження: {str(e)}")

def show_files(self):
    # Відображення файлів користувача
    if not self.current_user:
        return

    # Отримання файлів, до яких користувач має доступ (включаючи спільні файли)
    files = self.db.get_accessible_files(self.current_user[0])

    if not files:
        messagebox.showinfo("Інформація", "Файли не знайдено")
        return

    # Створення нового вікна для відображення файлів
    file_window = ctk.CTkToplevel(self.root)
    file_window.title("Ваші файли")
    file_window.geometry("600x400")

    # Створення заголовка таблиці
    header_frame = ctk.CTkFrame(file_window)
    header_frame.pack(fill='x', pady=(10, 5), padx=10)
    ctk.CTkLabel(header_frame, text="Назва файлу", width=200).pack(side='left')
    ctk.CTkLabel(header_frame, text="CID", width=200).pack(side='left')
    ctk.CTkLabel(header_frame, text="Рівень доступу", width=100).pack(side='left')

    # Створення прокручуваного фрейму для списку файлів
    files_list = ctk.CTkScrollableFrame(file_window)
    files_list.pack(fill='both', expand=True, padx=10, pady=5)

    # Заповнення списку файлів
    selected_file = {"value": None}  # Використання словника для зберігання посилання

    for i, file in enumerate(files):
        file_id, filename, cid, key, access = file
        file_frame = ctk.CTkFrame(files_list)
        file_frame.pack(fill='x', pady=2)
        file_frame.file_data = file  # Зберігання даних файлу

        ctk.CTkLabel(file_frame, text=filename, width=200).pack(side='left')
        ctk.CTkLabel(file_frame, text=cid, width=200).pack(side='left')
```

```
            ctk.CTkLabel(file_frame, text=access, width=100).pack(side='left')

            # Додавання смугастого кольору для кращої читабельності
            if i % 2 == 0:
                file_frame.configure(fg_color=("gray90",  "gray20"))

            # Додавання обробника кліку для вибору файлу
            def select_file(frame=file_frame):
                selected_file["value"] = frame.file_data

                # Виділення вибраного файлу
                for child in files_list.winfo_children():
                    child.configure(border_width=0)
                frame.configure(border_width=2, border_color="blue")

            file_frame.bind("<Button-1>", lambda event, f=file_frame: select_file(f))

        # Кнопки керування
        btn_frame = ctk.CTkFrame(file_window)
        btn_frame.pack(pady=10)

        def download_selected():
            if selected_file["value"]:
                self.download_file(selected_file["value"])
                file_window.destroy()
            else:
                messagebox.showinfo("Інформація", "Спочатку виберіть файл")

        ctk.CTkButton(btn_frame, text="Завантажити", command=download_selected,
width=150).pack(side='left', padx=5)
        ctk.CTkButton(btn_frame, text="Скасувати", command=file_window.destroy,
width=150).pack(side='left', padx=5)

    def download_file(self, file_data):
        # Завантаження обраного файлу
        file_id, filename, cid, encryption_key, _ = file_data

        # Перевірка доступу за допомогою ZKP
        if not verify_access(self.current_user[0], int(file_id), self.db.conn):
            messagebox.showerror("Помилка", "Доступ заборонено")
            return

        # Отримання з IPFS
        try:
            encrypted_data = retrieve_from_ipfs(cid)

            if not encrypted_data:
                messagebox.showerror("Помилка", "Не вдалося отримати файл з IPFS")
```

```
            return

        # Розшифрування файлу
        decrypted_data = self.encryption_manager.decrypt_file(encrypted_data,
encryption_key.encode())

        # Збереження файлу
        save_path = filedialog.asksaveasfilename(
            defaultextension=".*",
            initialfile=filename
        )

        if save_path:
            with open(save_path, "wb") as f:
                f.write(decrypted_data)
            messagebox.showinfo("Успіх", f"Файл успішно завантажено та розшифровано в
{save_path}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Помилка завантаження: {str(e)}")

  def show_sharing_options(self):
    # Відображення опцій для спільного доступу до файлів
    if not self.current_user:
        return

    # Отримання файлів, якими володіє поточний користувач
    files = self.db.get_owned_files(self.current_user[0])

    if not files:
        messagebox.showinfo("Інформація", "Немає файлів, якими можна поділитися")
        return

    # Створення вікна для спільного доступу
    share_window = ctk.CTkToplevel(self.root)
    share_window.title("Поділитися файлом")
    share_window.geometry("400x500")
    share_window.resizable(False, False)

    share_frame = ctk.CTkFrame(share_window)
    share_frame.pack(fill='both', expand=True, padx=20, pady=20)

    ctk.CTkLabel(share_frame, text="Виберіть файл для спільного доступу:").pack(pady=10)

    # Список файлів
    file_var = ctk.StringVar()
    file_names = [f[1] for f in files]
    file_combobox = ctk.CTkComboBox(share_frame, values=file_names, variable=file_var,
width=300)
```

```
        file_combobox.pack(pady=5)

        ctk.CTkLabel(share_frame, text="Ім'я користувача для спільного
доступу:").pack(pady=10)
        username_entry = ctk.CTkEntry(share_frame, width=300)
        username_entry.pack(pady=5)

        ctk.CTkLabel(share_frame, text="Рівень доступу:").pack(pady=10)

        # Створення радіо-кнопок для вибору рівня доступу
        access_var = ctk.StringVar(value="read")
        access_frame = ctk.CTkFrame(share_frame)
        access_frame.pack(pady=5, fill='x')

        ctk.CTkRadioButton(access_frame, text="Тільки читання", variable=access_var,
value="read").pack(anchor='w', padx=20, pady=5)
        ctk.CTkRadioButton(access_frame, text="Читання і запис", variable=access_var,
value="write").pack(anchor='w', padx=20, pady=5)

        def share_with_user():
            # Обробка надання доступу користувачу
            if not file_var.get() or not username_entry.get():
                messagebox.showwarning("Попередження", "Виберіть файл і введіть ім'я
користувача")
                return

            # Знайдення file_id за іменем файлу
            file_id = None
            for f in files:
                if f[1] == file_var.get():
                    file_id = f[0]
                    break

            if not file_id:
                messagebox.showerror("Помилка", "Файл не знайдено")
                return

            username = username_entry.get()
            access_level = access_var.get()

            # Перевірка чи користувач існує
            target_user = self.db.get_user_by_username(username)

            if not target_user:
                messagebox.showerror("Помилка", "Користувача не знайдено")
                return
            target_user_id = target_user[0]
             # Перевірка чи доступ вже надано
```

```
if self.db.check_access_right(file_id, target_user_id):
    # Оновлення існуючого доступу
    self.db.update_access_right(file_id, target_user_id, access_level)
else:
    # Створення нового доступу
    self.db.add_access_right(file_id, target_user_id, access_level)
messagebox.showinfo("Успіх", f"Спільний доступ надано користувачу {username}")
share_window.destroy()
```

Додаток В. Ілюстративний матеріал

Розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної автентифікації

Виконала ст. групи 2КІТС-21б Скидан Т.М.
Керівник: к.т.н., доцент кафедри МБІС Грицак А.В.

Актуальність

Актуальність теми дослідження обумовлена стрімким розвитком цифрових технологій, зростанням обсягів даних, що обробляються у корпоративному середовищі, та посиленням загроз інформаційній безпеці. У сучасних умовах цифровізації бізнес-процесів організації стикаються з проблемами забезпечення конфіденційності, цілісності та доступності корпоративної інформації, що зберігається та передається через інформаційні системи.

Наукова новизна роботи полягає в поєднанні децентралізованої файлової системи IPFS з інноваційними методами автентифікації та криптографічного захисту даних.

Мета

Метою даної розробки є створення захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної автентифікації. Дослідження спрямоване на створення масштабованої та стійкої до загроз архітектури, здатної забезпечити надійний захист даних в умовах зростаючої цифровізації.

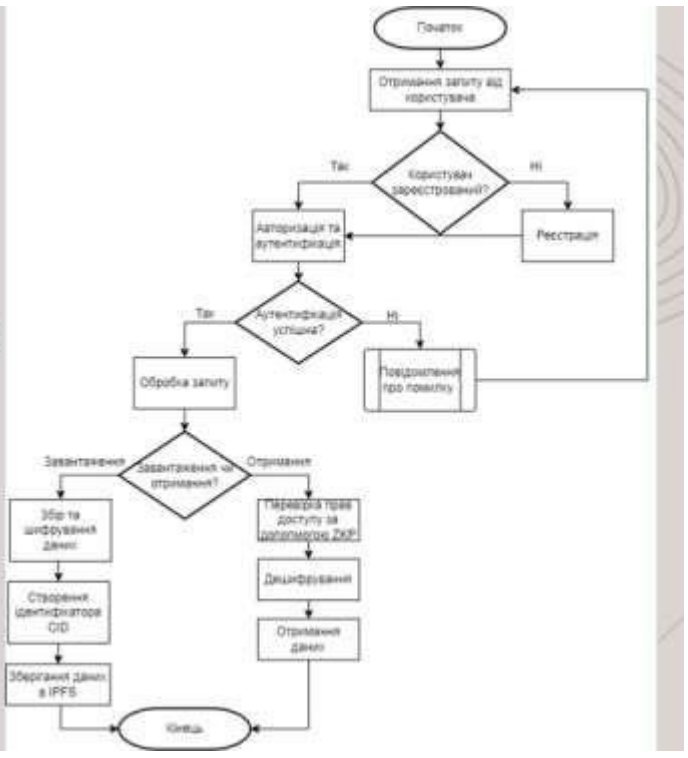
Об’єкт дослідження

Об'єктом дослідження є інформаційна інфраструктура корпоративних систем, в межах якої здійснюється зберігання, передача та обробка конфіденційних даних.

Предмет дослідження

Предметом дослідження є методи, технології та засоби забезпечення безпеки даних у децентралізованих системах.

Схема взаємодії користувача із розробленою системою



ER-модель реалізованої бази даних

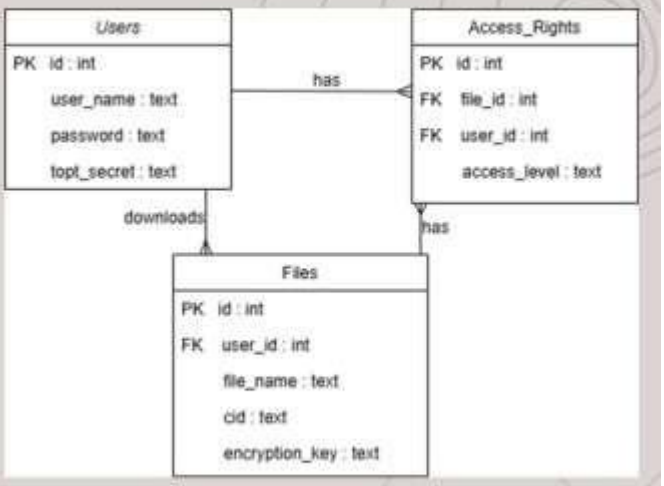
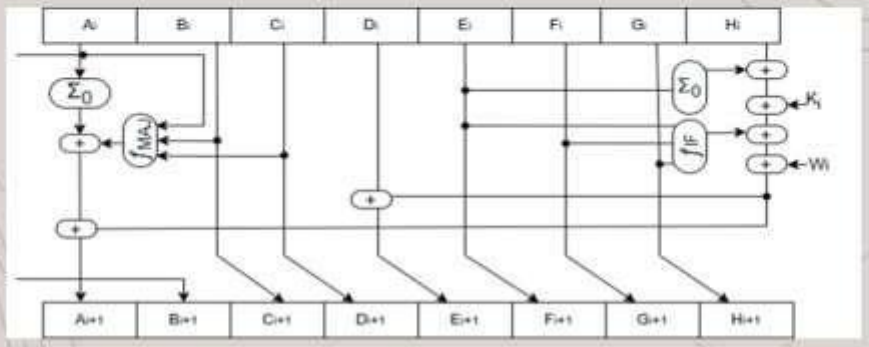
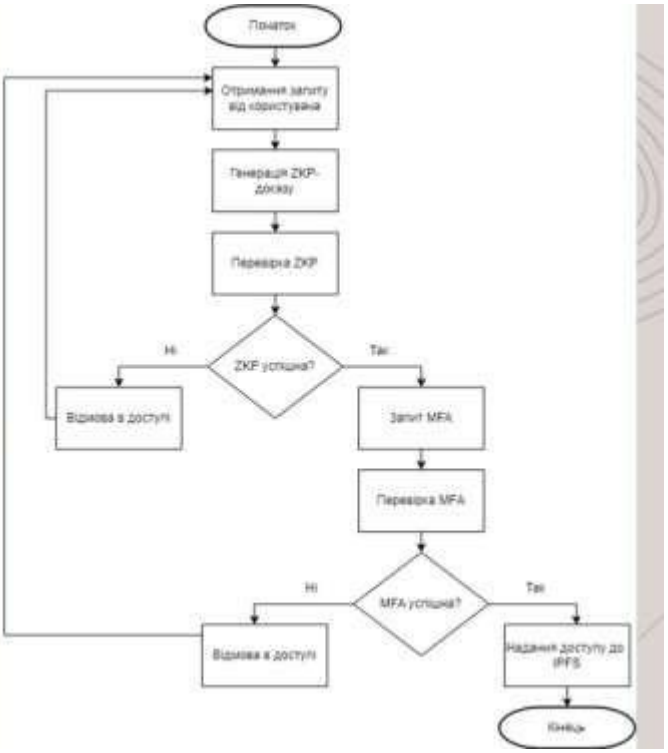


Схема роботи одного кроку функції стиснення SHA-256

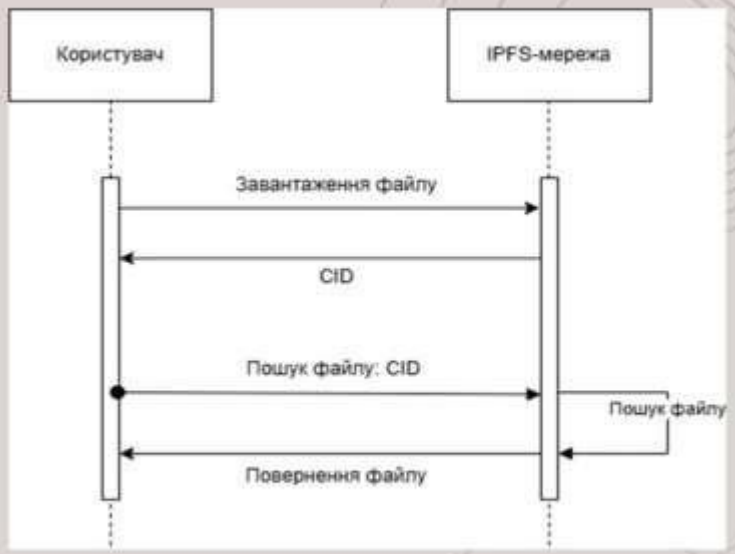


Блок-схема взаємодії модулів автентифікації з використанням ZKP та MFA

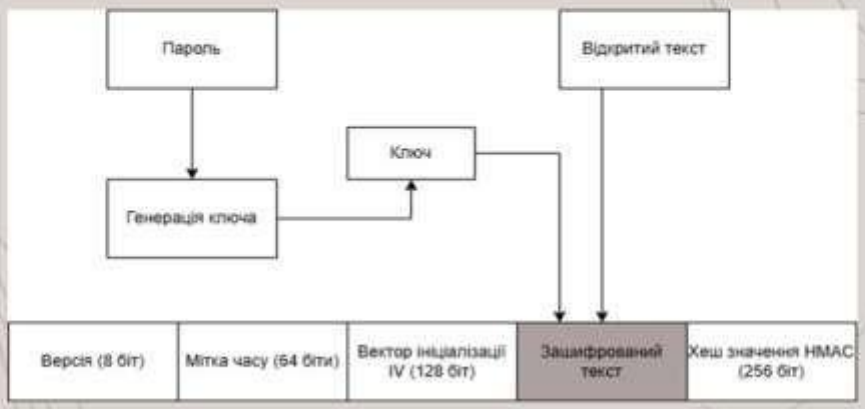


Архітектура протоколу доведення з нульовим розголошенням (ZKP)

Діаграма послідовностей взаємодії користувача з IPFS



Принцип роботи шифрування даних за допомогою Fernet

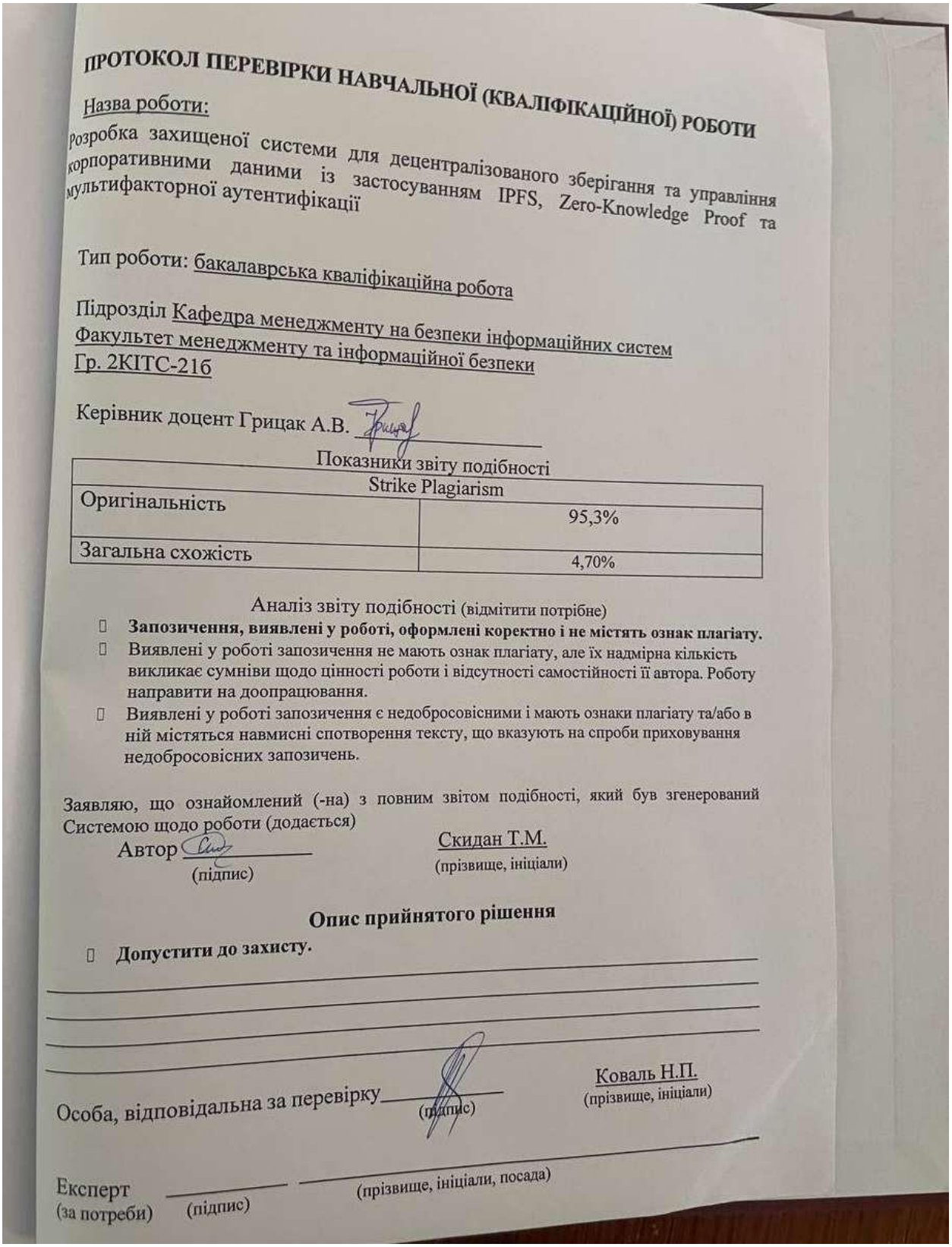


Висновки

- Проаналізовано сучасні підходи до зберігання даних та автентифікації.
- Обґрунтовано вибір децентралізованих технологій для підвищення безпеки.
- Розроблено архітектуру захищеної системи управління даними.
- Інтегровано IPFS для децентралізованого зберігання.
- Реалізовано шифрування файлів перед збереженням.
- Впроваджено Zero-Knowledge Proof для перевірки доступу.
- Реалізовано мультифакторну автентифікацію.
- Реалізовано зручний та інтуїтивно зрозумілий інтерфейс.

Дякую за увагу!

Додаток Г. Протокол перевірки на антиплагіат



ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка захищеної системи для децентралізованого зберігання та управління корпоративними даними із застосуванням IPFS, Zero-Knowledge Proof та мультифакторної аутентифікації

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 2КІТС-21б

Керівник доцент Грицак А.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	95,3%
Загальна схожість	4,70%

Аналіз звіту подібності (відмітити потрібне)

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

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

Автор _____ Скидан Т.М.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до захисту.**

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

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