Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах
Назва теми

Рівень вищої освіти *другий (магістерський)*

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Шифр ДРКІ 240245.24.02.21 ПЗ

Виконав здобувач II курсу, група КІ2м-24-2 — Підпис — Володимир МЕЛЬНИЧУК (Ім'я, ПРІЗВИЩЕ)

Керівник — канд.-екон. наук, доцент (Науковий ступінь, учене звання) — Підпис — Світлана САЧЕНКО (Ім'я, ПРІЗВИЩЕ)

Нормоконтролер — д. техн. наук, професор (Науковий ступінь, учене звання) — Підпис — Сергій ЛИСЕНКО (Ім'я, ПРІЗВИЩЕ)

До захисту допускаю:
завідувач кафедри КІІС — Підпис — Ольга ПАВЛОВА (Ім'я, ПРІЗВИЩЕ)
«01» травня 2026 р.
дата

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Рівень вищої освіти другий (магістерський)

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Завідувачка кафедри КІІС
_____ Ольга ПАВЛОВА

"12" 01 2026 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мельничук Володимир Олегович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

Керівник проекту (роботи) Саченко Світлана Іванівна, к.е.н., доцент.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 12.01.2026 р. № 6

2. Термін подання здобувачем роботи на кафедру 01.05.2026 р.

3. Вихідні дані до роботи Завдання на кваліфікаційну роботу

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

Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

Проєктування засобу захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

Програмно-апаратний засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «_12_» 01 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проєкту (роботи)	Термін виконання етапів проєкту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	12.01.2026	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	15.01.2026	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.02.2026	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	01.03.2026	виконано
5	Робота над розділом 3 – проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	29.03.2026	виконано
6	Оформлення пояснювальної записки згідно вимог	25.04.2026	виконано
7	Попередній захист ВКР	26.04.2025	виконано
8	Захист ВКР на засіданні ЕК	травень 2026 року	

Здобувач _____ Володимир МЕЛЬНИЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Світлана САЧЕНКО
Підпис Ім'я, ПРІЗВИЩЕ

РЕФЕРАТ

Тема кваліфікаційної роботи магістра: Засіб захисту даних та протокол доказу “нульового дня” у розподілених комп’ютерних системах

Автор роботи: Мельничук Володимир Олегович

Керівник роботи: Саченко Світлана Іванівна

Пояснювальна записка: 90 с., 6 рис., 2 табл., 80 джерел.

ПЕРЕЛІК КЛЮЧОВИХ СЛІВ РОЗПОДІЛЕНА КОМП’ЮТЕРНА СИСТЕМА, ЗАХИСТ ДАНИХ, ПОЛІТИКА БЕЗПЕКИ, ПРИВАТНА ВЕРИФІКАЦІЯ, ДОКАЗ НУЛЬОВОГО ЗНАННЯ, ПРОТОКОЛ, КРИПТОГРАФІЧНИЙ ЗАХИСТ, КОНТРОЛЬ ДОСТУПУ, КОНФІДЕНЦІЙНІСТЬ, ЖУРНАЛЮВАННЯ.

Об’єктом роботи є процес захисту даних у розподілених комп’ютерних системах.

Предметом роботи є методи, моделі, алгоритми та програмні засоби приватної верифікації відповідності компонента політиці безпеки з використанням протоколу доказу “нульового дня”.

Метою кваліфікаційної роботи магістра є підвищення рівня захисту даних у розподілених комп’ютерних системах шляхом розроблення засобу приватної верифікації та протоколу доказу “нульового дня”, що забезпечують підтвердження відповідності компонента політиці безпеки без надмірного розкриття його внутрішніх атрибутів.

Для розв’язання поставлених задач було використано методи аналізу розподілених комп’ютерних систем, методи формалізації політик безпеки, методи криптографічного захисту даних, методи побудови та перевірки доказів нульового знання, а також методи проєктування і реалізації програмного забезпечення.

Наукова новизна одержаних результатів полягає в розробленні методу приватної верифікації відповідності компонента політиці безпеки в розподіленому середовищі, у межах якого поєднано формалізоване подання політики безпеки, метод формування твердження про стан компонента, метод генерації та перевірки

доказу “нульового дня”, а також механізм інтеграції результату перевірки в контур контролю доступу. На відміну від відомих підходів, запропоноване рішення дозволяє приймати обґрунтоване рішення щодо доступу без безпосереднього розкриття внутрішніх атрибутів компонента.

Практичне значення одержаних результатів полягає в розробленні програмного засобу захисту даних для розподілених комп’ютерних систем, який реалізує формування твердження, генерацію доказу, перевірку доказу, прийняття рішення про доступ і журналювання результатів перевірки. Реалізований програмний засіб може бути використаний як основа для подальшого впровадження механізмів приватної верифікації в сервісних і розподілених програмних середовищах.

У вступі подано об’єкт і предмет роботи, мету, наукову новизну, практичне значення та загальну характеристику структури роботи.

У першому розділі виконано аналіз особливостей захисту даних у розподілених комп’ютерних системах, розглянуто основні загрози безпеці даних, проблему надмірного розкриття атрибутів.

У другому розділі сформовано концепцію побудови засобу захисту даних, побудовано модель приватної верифікації відповідності компонента політиці безпеки, розроблено метод формування твердження про стан безпеки компонента, метод генерації та перевірки доказу “нульового дня”.

У третьому розділі розроблено алгоритм формування доказу відповідності політиці безпеки, алгоритм перевірки доказу та прийняття рішення про доступ, визначено вимоги до програмного забезпечення, спроєктовано структуру програмного засобу та обґрунтовано вибір технологій його реалізації.

У четвертому розділі реалізовано програмний засіб захисту даних у вигляді модульного серверного застосунку, розроблено модулі формування твердження, генерації та перевірки доказу, підсистему керування політиками, конфігурацією та журналюванням подій, а також виконано перевірку працездатності розробленого рішення в межах тестових сценаріїв.

У висновках наведено основні результати виконаної роботи та підсумовано досягнення поставленої мети.

ЗМІСТ

СПИСОК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

API - програмний інтерфейс застосунку

CRUD - базові операції створення, читання, оновлення та видалення даних

DID - децентралізований ідентифікатор

DTLS - захищений транспортний протокол дейтаграмного типу

HPKE - гібридне шифрування з відкритим ключем

IAM - керування ідентифікацією та доступом

PKI - інфраструктура відкритих ключів

QUIC - сучасний транспортний протокол із вбудованим захистом з’єднання

REST - архітектурний стиль побудови мережевих сервісів

RPC - віддалений виклик процедур

SDLC - життєвий цикл розроблення програмного забезпечення

SSDF - рамка безпечного розроблення програмного забезпечення

TLS - протокол захисту транспортного рівня

UUID - універсальний унікальний ідентифікатор

VC - перевірювані облікові дані

Zero Trust - модель безпеки нульової довіри

ZKP - доказ нульового знання

ВСТУП

Розподілені комп’ютерні системи широко використовуються для обробки, передавання та зберігання даних у сервісних, корпоративних і мережевих середовищах. Їх функціонування пов’язане з постійною взаємодією між великою кількістю вузлів, сервісів і прикладних компонентів, що працюють у різних доменах довіри. У таких умовах традиційні механізми перевірки прав доступу, автентифікації та підтвердження стану компонента часто передбачають передавання значного обсягу службових або чутливих даних. Це призводить до появи додаткових ризиків, оскільки навіть технічні атрибути, конфігураційні параметри або проміжні результати перевірки можуть ставати доступними стороннім особам і використовуватися для обходу захисту або впливу на роботу системи.

Сучасні умови експлуатації розподілених комп’ютерних систем вимагають переходу до таких підходів, за яких перевірка відповідності компонента встановленим вимогам безпеки виконується без надмірного розкриття його внутрішніх характеристик. Наприклад, у процесі підтвердження права доступу, перевірки конфігурації вузла або встановлення відповідності політиці безпеки не завжди є доцільним передавати повний набір атрибутів, якщо для ухвалення рішення достатньо лише підтвердити сам факт виконання необхідних умов. Унаслідок цього виникає потреба в удосконаленні механізмів захисту даних і введенні до їх складу криптографічних засобів, здатних підтверджувати істинність певного твердження без розкриття вхідних значень, на основі яких воно сформоване.

Перспективним напрямом у цій сфері є застосування протоколів доказу нульового знання, які дозволяють реалізувати приватну верифікацію відповідності компонента політиці безпеки. Використання такого підходу дає змогу перевіряти допустимість виконання дії, коректність стану компонента або наявність необхідних прав без безпосереднього розкриття внутрішніх атрибутів системи. Для розподілених середовищ це має особливе значення, оскільки зменшує ризик витоку

службової інформації, підвищує рівень конфіденційності та дозволяє поєднати механізми контролю доступу з криптографічним підтвердженням коректності перевірки.

Актуальність роботи полягає в розробленні засобу захисту даних і протоколу доказу “нульового дня” для розподілених комп’ютерних систем, що забезпечують підтвердження відповідності компонента політиці безпеки без надмірного розкриття його внутрішніх атрибутів.

Метою кваліфікаційної роботи магістра є підвищення ефективності захисту даних у розподілених комп’ютерних системах шляхом розроблення засобу приватної верифікації та протоколу доказу “нульового дня”, що забезпечують перевірку відповідності компонента політиці безпеки без розкриття конфіденційної інформації.

Поставлена мета досягається розв’язанням таких основних завдань:

– проаналізувати особливості захисту даних у розподілених комп’ютерних системах, наявні загрози безпеці та сучасні підходи до приватної верифікації;

– розробити модель приватної верифікації відповідності компонента політиці безпеки;

– розробити метод формування твердження про стан безпеки компонента та метод генерації і перевірки доказу “нульового дня”;

Об’єктом роботи є процес захисту даних у розподілених комп’ютерних системах.

Предметом роботи є методи, моделі, алгоритми та програмні засоби приватної верифікації відповідності компонента політиці безпеки з використанням протоколу доказу “нульового дня”.

Наукова новизна одержаних результатів полягає в розробленні методу приватної верифікації відповідності компонента політиці безпеки в розподіленому середовищі, у межах якого поєднано формалізоване подання політики безпеки, метод формування твердження про стан компонента, метод генерації та перевірки доказу “нульового дня”, а також механізм інтеграції результатів перевірки в контур контролю доступу. На відміну від відомих підходів, запропоноване рішення

дозволяє приймати обґрунтоване рішення щодо доступу без безпосереднього розкриття внутрішніх атрибутів компонента.

Практичне значення одержаних результатів полягає в розробленні програмного засобу захисту даних для розподілених комп'ютерних систем, який реалізує формування твердження, генерацію доказу, перевірку доказу, прийняття рішення про доступ і журналювання результатів перевірки. Розроблений засіб може бути використаний як основа для подальшого впровадження механізмів приватної верифікації у сервісних і розподілених програмних середовищах.

Для розв'язання поставлених завдань використано методи аналізу розподілених комп'ютерних систем, методи формалізації політик безпеки, методи криптографічного захисту даних, методи побудови та перевірки доказів нульового знання, а також методи проєктування і реалізації програмного забезпечення.

За темою кваліфікаційної роботи опубліковано одну публікацію [81] у Збірнику наукових праць за матеріалами XIX Всеукраїнська науково-практична web конференція аспірантів, студентів та молодих вчених комп'ютерні інтелектуальні системи та мережі. (Кривий ріг – 2026).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ ПІДХОДІВ І ЗАСОБІВ ЗАХИСТУ ДАНИХ У РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

1.1 Аналіз існуючих підходів і рішень

Розподілені комп'ютерні системи дедалі частіше стають основою для обробки та обміну даними між незалежними учасниками - сервісами, організаціями та пристроями. Зростання кількості вузлів і каналів взаємодії підвищує вимоги до конфіденційності, цілісності та керованості доступу, адже дані циркулюють поза межами одного контрольованого периметра [9, 11, 13]. У таких умовах традиційні підходи, що спираються лише на мережеву ізоляцію або довіру до внутрішнього сегмента, демонструють обмеження, оскільки ризики виникають як на транспортному рівні, так і на рівні прикладної логіки та ланцюгів постачання програмного забезпечення [12, 15, 19].

Окрему складність створює поєднання двох вимог, які на практиці часто конфліктують: з одного боку потрібна прозора верифікація прав та станів (наприклад, підтвердження відповідності політикам або коректності транзакції), а з іншого - мінімізація розкриття чутливих атрибутів, бізнес-логіки або персональних даних [1, 3, 14]. Для розподілених середовищ ця суперечність проявляється особливо гостро: учасники взаємодіють із різними рівнями довіри, а маршрути даних можуть проходити через сторонні мережі та сервіси [24, 25]. У підсумку ключовим стає пошук механізмів, які дозволяють перевіряти твердження про дані, не розкриваючи самі дані в повному обсязі.

Одним із найбільш перспективних підходів для такого класу задач є протоколи доказу нульового знання (Zero-Knowledge Proof, ZKP). Їх використання дає змогу підтверджувати істинність твердження (наприклад, належність до групи, коректність обчислення або відповідність правилам) без передачі вихідних секретів чи повного набору атрибутів [45, 46]. У сучасних застосуваннях ZKP активно розвиваються як на рівні теорії протоколів, так і на рівні інженерних реалізацій: універсальні та масштабовані конструкції на кшталт PLONK/Marlin/Halo/Nova

дозволяють будувати докази для складних обчислень, підтримувати рекурсію та зменшувати обсяг довірених налаштувань або витрати на верифікацію [32-35]. Паралельно вдосконалюються схеми для практичних сценаріїв конфіденційних транзакцій, де важливі короткі докази та ефективність перевірки [38, 39]. Внаслідок цього ZKP поступово переходять із суто криптографічних конструкцій у прикладні компоненти захисту даних у розподілених системах.

Разом із перевагами виникають і технічні виклики. Для реальних систем критичними стають обчислювальні витрати генерації доказу, вимоги до пам’яті, параметри довіреного налаштування (за його наявності), а також стійкість допоміжних примітивів - зокрема хеш-функцій, що використовуються всередині схем [40, 41]. Під час проєктування рішення доводиться узгоджувати вимоги безпеки та продуктивності з особливостями розподіленої архітектури, мережевими затримками та потребою в масштабуванні. Додатково важливими є питання формалізації тверджень, які підлягають перевірці, а також інтерфейси інтеграції ZKP у прикладні протоколи обміну повідомленнями та ідентифікації [30, 31].

У предметній області захисту даних також суттєве значення має стандартизація та відповідність практикам керування ризиками. Міжнародні стандарти систем менеджменту інформаційної безпеки та контролів (зокрема сімейство ISO/IEC 27001/27002 і пов’язані розширення) задають рамки для політик, процесів і технічних заходів, які забезпечують цілісне управління безпекою даних [1-4]. На рівні державних та наддержавних вимог важливими є нормативні акти та стратегії кібербезпеки, що визначають пріоритети захисту критичних сервісів та вимоги до інцидент-менеджменту [8, 21]. Унаслідок цього розробка прикладного засобу захисту даних у розподілених системах має враховувати не лише криптографічну коректність, а й організаційні та архітектурні аспекти, пов’язані з життєвим циклом програмного забезпечення та контролем доступу [11, 12].

З огляду на зазначене, предметна область цієї роботи охоплює методи захисту даних у розподілених комп’ютерних системах із фокусом на інтеграцію протоколів доказу нульового знання як механізму приватної верифікації. Під час формування постановки задачі було враховано, що ефективне рішення має: (1)

забезпечувати конфіденційність чутливих атрибутів під час підтвердження прав або властивостей; (2) підтримувати верифікацію доказів у вузлах з різними рівнями довіри; (3) бути сумісним із сучасними мережевими та прикладними протоколами, а також із підходами керування безпекою [9, 24-29].

Постановка задачі полягає у розробленні засобу захисту даних та протоколу доказу нульового знання для розподілених комп'ютерних систем, який забезпечує формування формалізованого твердження (statement) про дані або обчислення, що підлягає перевірці без розкриття секретних значень, а також генерацію доказу та його перевірку з прийнятними витратами часу і ресурсів у вузлах розподіленої системи. Окремо передбачено інтеграцію з механізмами автентифікації та авторизації і транспортного захисту, зокрема у вебсервісних сценаріях та протоколах обміну повідомленнями [24, 25, 30]. Додатково враховуються вимоги безпечної розробки, керування ключами та контролів доступу на рівні організації, що дозволяє узгодити криптографічну частину рішення з практиками інформаційної безпеки [11, 12, 1].

Очікуваний результат розв'язання поставленої задачі полягає у створенні архітектурно та криптографічно узгодженого рішення, яке дозволяє перевіряти критично важливі твердження у розподіленому середовищі без надмірного розкриття даних. Це дає змогу зменшити площину атак, підвищити приватність взаємодії між учасниками та спростити доведення відповідності політикам безпеки за наявності багатьох сторін і зовнішніх інтеграцій [9, 13, 45].

1.2 Захист даних у розподілених комп'ютерних системах

Сучасний розвиток інформаційних технологій дедалі більше пов'язаний із переходом від локальних програмних рішень до розподілених середовищ, у яких дані постійно циркулюють між сервісами, організаціями та користувачами. У такій архітектурі головним викликом стає не стільки передавання інформації, скільки контроль того, що саме розкривається під час взаємодії, хто і на яких умовах може перевірити коректність операцій, а також як зберігається довіра між сторонами, які

не мають спільного периметра безпеки [9, 11]. Через це захист даних у розподілених комп'ютерних системах дедалі частіше розглядається як поєднання криптографічних механізмів, архітектурних принципів та процесних контролів, що задають правила для всього життєвого циклу даних і програмного забезпечення [1, 12].

Паралельно зростає потреба в «приватній перевірці» - ситуаціях, коли необхідно довести певну властивість даних або коректність обчислення, але без розкриття самих даних у повному обсязі. Типові приклади включають підтвердження права доступу без розкриття зайвих атрибутів, доказ відповідності транзакції правилам без демонстрації всіх внутрішніх параметрів, або перевірку того, що обчислення виконано правильно, не передаючи вхідні значення [45, 46]. У розподіленому середовищі така модель знижує обсяг довіри до вузлів-посередників і зменшує ризики витоку в разі компрометації окремих компонентів, що добре узгоджується з підходами нульової довіри (Zero Trust), де жоден сегмент не вважається апріорно безпечним [9, 15].

Одним із найбільш придатних інструментів для приватної перевірки є протоколи доказу нульового знання (Zero-Knowledge Proof, ZKP). Їх застосування дозволяє підтвердити істинність твердження про секретні або частково приховані дані без розкриття цих даних, а в практичних системах - також звести до мінімуму обсяг метаданих, що передаються між сторонами [45, 46]. За останні роки ZKP отримали значний розвиток завдяки універсальним конструкціям і системам доказів, які зменшують залежність від специфічних налаштувань, покращують масштабованість і підтримують рекурсивну верифікацію [32-35]. Це дозволяє будувати протоколи, де доказ може перевірятися багатьма вузлами, а ланцюжок перевірок узгоджується в єдину логіку без повторного розкриття первинних секретів.

Практична реалізація ZKP у розподілених системах майже завжди спирається на допоміжні криптографічні примітиви: схеми хешування, канонічне кодування даних, узгодженість серіалізації повідомлень та захищені транспортні протоколи. Для веборієнтованих і сервісних сценаріїв важливими є сучасні стандарти

передавання та захисту трафіку, зокрема QUIC і DTLS 1.3, які задають вимоги до безпечного транспорту та стійких механізмів встановлення з'єднань [24, 25]. Для прикладних протоколів, де потрібне узгоджене шифрування повідомлень або обмін ключами, корисними є стандартизовані конструкції, такі як HPKE, а для коректної перевірки підписів і хешів - узгоджені правила канонізації (наприклад JCS) та хешування до еліптичних кривих [26, 27, 29]. Це дозволяє розглядати криптографічну частину рішення не ізольовано, а як шар, що взаємодіє з транспортом і прикладними форматами даних.

Окремим питанням є представлення ідентичностей і атрибутів у системах, де потрібно підтверджувати властивості без повного розкриття. Для цього придатними є моделі перевірюваних облікових даних (Verifiable Credentials), які стандартизують структуру тверджень, підписів і перевірок, а також забезпечують основу для інтеграції з ZKP-методами у сценаріях вибіркового розкриття (selective disclosure) [30, 31]. Такі підходи дозволяють пов'язати криптографічну доказовість із прикладною логікою доступу: перевіряється не просто «секрет», а конкретна властивість, яка є значущою для політики безпеки, контракту або регламенту взаємодії.

Разом із перспективністю ZKP виникають інженерні компроміси. Генерація доказу часто є суттєво дорожчою за його перевірку, а в окремих схемах важливими стають вимоги до пам'яті, параметри довіреного налаштування або стабільність допоміжних хеш-функцій, оптимізованих під нульове знання [40, 41]. Для практичних впроваджень корисними є систематичні огляди та бенчмарки, які порівнюють сімейства zkSNARK/zkSTARK/Bulletproofs за витратами часу, розміром доказів і практичною придатністю [43, 44]. Окремий інтерес становлять модифікації Bulletproofs та наступні покоління коротких доказів, що орієнтовані на конфіденційні транзакції та компактну верифікацію [38, 39]. Внаслідок цього клас протоколу може обиратися не абстрактно, а відповідно до ресурсних обмежень вузлів і вимог до затримок у розподіленій взаємодії.

Питання безпеки в цій роботі також розглядається з позиції керування ризиками й відповідності контролям. Міжнародні стандарти сімейства ISO/IEC

27001 та 27002 визначають вимоги до системи менеджменту інформаційної безпеки і каталогу контролів, а розширення для приватності задають рамки для обробки персональних даних у організаційних процесах [1-3]. На рівні державної політики важливими є документи, що визначають пріоритети кібербезпеки та підходи до захисту критичних сервісів, а в європейському контексті - нормативні вимоги до кіберстійкості та обміну даними, зокрема NIS2 і акти, пов’язані з управлінням даними [8, 21-23]. Це дозволяє поєднати криптографічну частину рішення з вимогами безпечної розробки, аудиту та контролю доступу, а не зводити захист лише до вибору окремої схеми [11, 12].

У межах предметної області цієї роботи центральним стає завдання побудови засобу захисту даних, який підтримує приватну верифікацію у розподіленій системі. Передбачається, що учасники можуть мати різні ролі та рівні довіри, а тому потрібні механізми, які дозволяють перевіряти коректність операцій і дотримання політик безпеки без надмірного розкриття даних. Архітектурно це може бути оформлено як сервіс генерації доказів, модуль верифікації на вузлах-перевірниках і шар керування ключами та атрибутами, який відповідає за життєвий цикл секретів і пов’язаних доказів [11, 12, 30].

Постановка задачі в цій роботі зводиться до розроблення засобу захисту даних та протоколу доказу нульового знання для розподілених комп’ютерних систем, який забезпечує формування формалізованого твердження (statement) про дані або обчислення, що підлягає перевірці без розкриття секретних значень, а також генерацію доказу та його перевірку з прийнятними витратами часу і ресурсів у вузлах системи. Окремо передбачено інтеграцію з механізмами автентифікації та авторизації і транспортного захисту, зокрема у вебсервісних сценаріях та протоколах обміну повідомленнями [24, 25, 30]. Додатково враховуються вимоги безпечної розробки, керування ключами та контролів доступу на рівні організації, що дозволяє узгодити криптографічну частину рішення з практиками інформаційної безпеки [11, 12, 1].

У підсумку очікуваний результат виконання поставленої задачі полягає в отриманні узгодженого протоколу та програмного засобу, здатного зменшити

обсяг розкриття даних під час перевірки прав і властивостей у розподіленому середовищі. Це дозволяє підвищити приватність взаємодії, знизити чутливість системи до компрометації окремих компонентів і спростити підтвердження відповідності політикам безпеки за участі багатьох сторін та зовнішніх інтеграцій [9, 13, 45].

1.3 Аналіз існуючих рішень для захисту даних у розподілених системах та їх обмеження

Наявні програмні рішення, що використовуються для захисту даних у розподілених комп'ютерних системах, умовно групуються за призначенням у три напрями: (1) інфраструктурні підходи керування доступом і мережевої взаємодії, (2) криптографічні протоколи захисту каналу та повідомлень, (3) системи приватної верифікації на основі доказів нульового знання. Кожен напрям розв'язує окремий клас задач - від політик доступу й сегментації до забезпечення конфіденційності передавання та доведення коректності операцій без розкриття чутливих значень [9, 11, 24, 25]. Водночас у реальних сценаріях ці складові часто існують як окремі шари, що потребують узгодження форматів даних, життєвого циклу ключів і правил перевірки.

До першого напряму належать архітектурні моделі нульової довіри (Zero Trust) та пов'язані практики керування доступом. Вони формують основу для контролю того, хто і за яких умов отримує доступ до ресурсів у середовищі, де не передбачається «довіреної» внутрішньої мережі [9, 15]. Подібні моделі добре описують принципи сегментації, автентифікації та авторизації, однак самі по собі не розв'язують задачу приватної перевірки: у багатьох випадках для прийняття рішення все одно доводиться передавати атрибути або контекст доступу стороні, яка виконує перевірку. Унаслідок цього зростають ризики надмірного розкриття даних і накопичення чутливих метаданих у сервісах-посередниках.

Другий напрям представлений стандартизованими транспортними та прикладними криптопротоколами, які забезпечують захист передавання

інформації. Протоколи QUIC і DTLS 1.3 задають сучасний рівень безпеки для з’єднань у мережевому середовищі, зменшуючи класичні загрози перехоплення та підміни трафіку [24, 25]. Для шифрування повідомлень і обміну ключами в прикладних сценаріях використовується HPKE, що спрощує побудову захищених схем доставляння даних між сторонами з різними ролями [26]. Ці рішення добре закривають «канальну» частину безпеки, однак не гарантують мінімізації розкриття на рівні змісту: якщо вузол має перевірити певну властивість даних, то самі дані або їх значуща частина все одно передається для валідації. Це обмеження особливо помітне в розподілених системах із багатьма учасниками, де перевірка виконується не один раз, а повторюється на різних вузлах.

Третій напрям охоплює рішення на основі доказів нульового знання (ZKP), які дають змогу підтверджувати істинність твердження про дані або коректність обчислення без розкриття секретних значень [45, 46]. У практичній площині найбільше поширення отримали системи доказів типу zkSNARK/zkSTARK та суміжні конструкції, а також схеми коротких доказів для конфіденційних транзакцій [38, 43, 44]. Універсальні та масштабовані підходи на кшталт PLONK, Marlin, Halo і Nova створюють основу для побудови доказів для складних обчислень і, за потреби, для рекурсивного складання доказів [32-35]. Унаслідок цього приватна верифікація стає технічно здійсненною навіть у системах із великою кількістю операцій і перевірок.

Разом із тим аналіз існуючих ZKP-рішень показує низку обмежень, що безпосередньо впливають на придатність до промислового застосування. По-перше, генерація доказу зазвичай є значно дорожчою за перевірку, що створює навантаження на вузли-генератори та ускладнює роботу в режимі високої інтенсивності запитів [43, 44]. По-друге, у низці схем важливими залишаються параметри налаштування та вимоги до надійності допоміжних криптопримітивів (зокрема хеш-функцій, оптимізованих під нульове знання), що впливає на вибір бібліотек і профілю загроз [40, 41]. По-третє, інтеграція ZKP у прикладні протоколи часто потребує чіткої формалізації тверджень (statement) та узгодження форматів серіалізації даних, інакше виникають неоднозначності під час перевірки [27, 30].

Такі фактори зсувають акцент із «наявності протоколу» на проєктування повного контуру - від моделі даних і ключів до політик доступу й аудиту.

Окремо слід відзначити, що на практиці рішення з нульовим знанням нерідко впроваджуються як «острівці» приватної верифікації в межах ширшої інфраструктури безпеки. У таких випадках без належного узгодження з процесними контролями та вимогами безпечної розробки виникають розриви між криптографічною коректністю та експлуатаційною безпекою [11, 12]. Стандарти та рамки керування інформаційною безпекою задають вимоги до керування ключами, доступом, інцидентами й життєвим циклом ПЗ, однак вони не визначають конкретну ZKP-схему та не знімають потреби інженерного вибору під конкретний сценарій [1, 2].

У підсумку наявні рішення забезпечують сильні можливості на окремих рівнях: Zero Trust підсилює контроль доступу, транспортні протоколи захищають канал, а ZKP надає механізм приватної верифікації [9, 24, 45]. Обмеження проявляються тоді, коли необхідно поєднати ці рівні в одному засобі: потрібні узгоджені моделі тверджень, керування ключами, оцінка витрат генерації доказів і стандартизована інтеграція в сервіси розподіленої системи [11, 30, 43]. Це створює підстави для розроблення єдиного рішення, у якому протокол доказу нульового знання та засіб захисту даних проєктуються як взаємопов’язані компоненти спільної архітектури, а не як набір ізольованих модулів [32-35, 12].

1.4 Вимоги до засобу захисту даних та протоколу доказу нульового знання у розподілених комп’ютерних системах

Засіб захисту даних у розподіленому середовищі має забезпечувати таку організацію взаємодії, за якої коректність операцій і відповідність політикам безпеки перевіряються без надмірного розкриття інформації. Однією з базових вимог у цьому випадку є підтримка моделі приватної верифікації, за якої для кожної критично важливої дії, зокрема доступу до ресурсу, підтвердження атрибутів або виконання обчислення, формується формалізоване твердження, а

його істинність підтверджується доказом нульового знання без передавання секретних значень або повних наборів атрибутів [45, 46]. Такий підхід дає змогу зменшити кількість точок накопичення чутливих даних і підвищити стійкість системи до компрометації окремих вузлів [9, 15].

Важливою вимогою є криптографічна коректність і стійкість використовуваних примітивів. Обране сімейство ZKP має відповідати цільовому сценарію за типом тверджень, моделлю довіри та ресурсними обмеженнями. Для задач, у яких необхідні універсальність і масштабованість, доцільно орієнтуватися на сучасні системи доказів та їх інженерні реалізації, що підтримують складні обчислення і, за потреби, рекурсивну композицію [32–35]. Для сценаріїв, де критичними є компактність доказу та швидкість перевірки, актуальними залишаються схеми класу Bulletproofs і підходи, що розвивають цю ідею [38, 39]. Разом із цим має враховуватися те, що генерація доказу в більшості випадків є суттєво дорожчою за його перевірку, тому вимоги до продуктивності та планування навантаження є принципово важливими [43, 44].

Окрему групу становлять вимоги до керування криптографічними ключами й секретами. У структурі такого засобу повинні бути передбачені механізми генерації, ротації, безпечного зберігання та контролю доступу до ключового матеріалу. Для прикладних сценаріїв обміну повідомленнями доцільно спиратися на стандартизовані конструкції, зокрема HPKE, що використовується для шифрування з відкритим ключем у сучасних протоколах, а також на узгоджені правила канонізації даних і серіалізації, які зменшують ризики неоднозначної інтерпретації під час верифікації [26, 27]. Для мережевої взаємодії обов’язковою є підтримка захищених транспортних протоколів, зокрема QUIC та DTLS 1.3, що забезпечують конфіденційність і цілісність каналу під час передавання доказів, публічних параметрів і службових повідомлень [24, 25].

Не менш важливою є вимога сумісності з механізмами ідентифікації та авторизації. Засіб повинен підтримувати сценарії, у яких доступ надається на основі підтверджених властивостей, наприклад належності до ролі, наявності права або відповідності певним обмеженням, причому підтвердження має виконуватися

вибірково, тобто із розкриттям лише тієї частини відомостей, яка потрібна для прийняття рішення. Для таких задач перспективним є узгодження з моделями перевірюваних облікових даних Verifiable Credentials, що стандартизують структуру тверджень і правила їх перевірки в умовах міждоменної взаємодії [30, 31]. Це спрощує інтеграцію рішення в сервіси, де учасники мають різний рівень довіри та різні політики обробки даних.

Вимоги до збереження даних і забезпечення їх цілісності охоплюють журналювання подій, контроль незмінності критичних записів і можливість відтворення ланцюжка перевірок. Журнали повинні фіксувати факт перевірки доказу, ідентифікатор твердження, версію параметрів і результат валідації без включення секретних значень. Такий підхід підвищує придатність рішення до аудиту та розслідування інцидентів, не створюючи додаткових каналів витоку інформації. У прикладній площині ці вимоги узгоджуються з контролями систем керування інформаційною безпекою, зокрема з підходами ISO/IEC 27001 та каталогом контролів ISO/IEC 27002 [1, 2]. Для систем, у яких обробляються персональні дані, додатково враховуються вимоги мінімізації даних і керування приватністю [3, 14].

До нефункціональних вимог належить масштабованість. Система повинна зберігати працездатність у разі зростання кількості вузлів, частоти запитів на генерацію доказів і обсягів метаданих, пов’язаних із верифікацією. Досягнення цієї властивості пов’язується з модульною архітектурою, розподілом ролей між компонентами, зокрема між генераторами доказів, перевірниками та службами керування ключами, а також із можливістю горизонтального масштабування критичних сервісів, що узгоджується з практиками Zero Trust і сегментації довіри [9, 15]. Вимога продуктивності, своєю чергою, передбачає прийнятний час генерації доказу, швидку перевірку на стороні сервісів, які приймають рішення, а також ефективне кешування публічних параметрів і версійних артефактів для уникнення повторних дорогих операцій [43, 44].

Вимога надійності охоплює стійкість до відмов, резервування критичних сервісів і відновлення після збоїв без втрати узгодженості стану. Для програмної

реалізації також важливою є відповідність підходам безпечної розробки, включно з контролем ланцюгів постачання, тестуванням, управлінням вразливостями та повторюваністю збірок, що відображено в рамках на кшталт SSDF [12]. Це дозволяє зменшити ризики появи уразливостей у криптографічних модулях і сервісних компонентах, від яких безпосередньо залежить безпечність усієї системи.

Важливою вимогою до такого засобу є адаптивність до змін політик безпеки та умов функціонування розподіленого середовища. У реальних системах склад учасників, набір доступних сервісів, правила авторизації та перелік перевірюваних атрибутів не залишаються незмінними, тому архітектура рішення повинна підтримувати оновлення правил перевірки без повного перегляду всієї системи. Це стосується як зміни набору допустимих тверджень, так і перегляду логіки формування доказів для нових сценаріїв взаємодії. За такої організації засіб захисту даних зберігає придатність до тривалої експлуатації та не втрачає ефективності в умовах розвитку прикладних сервісів і зміни вимог до безпеки [9, 15].

Не менш суттєвою є вимога сумісність між різними програмними платформами та сервісами. Оскільки розподілені комп’ютерні системи часто формуються з компонентів, розроблених із використанням різних технологічних стеків, засіб захисту даних має бути придатним до інтеграції в неоднорідне середовище без суттєвого ускладнення обміну повідомленнями та процедур перевірки. У підсумку забезпечується можливість впровадження протоколу доказу нульового знання не лише в окремому сервісі, а й у ширшому контурі взаємопов’язаних інформаційних систем [24, 26, 30].

Окремої уваги потребує вимога керованості та спостережуваності системи в процесі експлуатації. Для практичного застосування недостатньо лише забезпечити криптографічну коректність протоколу, оскільки важливим залишається і контроль фактичної роботи сервісних компонентів, своєчасне виявлення перевантажень, помилок верифікації, збоїв у роботі служб ключів або відхилень у поведінці окремих вузлів. У зв’язку з цим до архітектури доцільно включати механізми моніторингу стану компонентів, збирання технічних метрик, фіксації подій безпеки

та централізованого аналізу журналів. Це дозволяє підтримувати не лише конфіденційність і цілісність даних, а й загальну керованість системи як програмного засобу, що функціонує в умовах постійної мережевої взаємодії та змінного навантаження [1, 2, 12].

У підсумку засіб захисту даних і протокол доказу нульового знання для розподіленого середовища мають поєднувати механізми приватної верифікації, криптографічну стійкість, захищений транспорт, узгоджені правила серіалізації, керування ключами, аудит без розкриття секретних значень, а також масштабовану й надійну архітектуру для розподіленої експлуатації [1, 24–27, 45]. Сукупність наведених вимог формує основу для подальшого проєктування засобу, здатного забезпечити підтвердження коректності дій і властивостей даних без порушення конфіденційності в практичних сценаріях функціонування розподілених комп’ютерних систем [9, 11].

1.5 Постановка задачі розробки засобу захисту даних та протоколу доказу нульового знання у розподілених комп’ютерних системах

Зростання обсягів даних і ускладнення взаємодії між сервісами в розподілених комп’ютерних системах підсилюють потребу в рішеннях, які одночасно забезпечують конфіденційність, цілісність і керованість доступу. Дані дедалі частіше проходять через кілька доменів довіри, обробляються в різних вузлах і можуть бути предметом перевірки з боку багатьох учасників. У таких умовах слабкою ланкою стає не лише транспорт, а й спосіб підтвердження прав, властивостей і коректності обчислень: традиційна перевірка часто вимагає передавання надлишкової інформації, що збільшує ризик витоків і накопичення чутливих метаданих у проміжних компонентах [9, 31, 54]. З огляду на це зростає цінність підходів, де перевірка виконується за принципом мінімального розкриття, а довіра зменшується до рівня формально перевірюваних тверджень [15, 45].

У сучасних умовах особливого значення набувають протоколи доказу нульового знання (Zero-Knowledge Proof, ZKP), які дозволяють підтверджувати

істинність твердження про дані або обчислення без розкриття секретних значень. Це створює основу для приватної верифікації в розподілених системах, де учасники можуть бути незалежними, а перевірка повинна виконуватися багаторазово в різних вузлах [45, 46]. Разом із тим аналіз наукових і прикладних напрацювань показує, що ефективність ZKP-рішень визначається не лише вибором схеми, а й узгодженістю протоколу з мережею, форматами даних, керуванням ключами та архітектурою сервісів. Для практичних впроваджень важливими є сучасні системи доказів і підходи до побудови універсальних та масштабованих доказів, зокрема конструкції сімейства PLONK/Marlin/Halo/Nova, а також порівняльні оцінки продуктивності різних класів схем [32-35, 43, 44].

Проблема, що потребує розв’язання, полягає у відсутності цілісного програмного рішення, яке поєднувало б засіб захисту даних, протокол приватної верифікації та узгоджені механізми інтеграції в розподілену архітектуру. На практиці контроль доступу, транспортний захист і перевірка коректності часто реалізуються як окремі модулі, а «приватність перевірки» або відсутня, або впроваджується фрагментарно, що ускладнює супровід, підвищує вимоги до налаштування і не гарантує мінімізації розкриття даних у всіх критичних точках [9, 11, 12]. Унаслідок цього зростає потреба в такій постановці задачі, де ZKP-протокол і засіб захисту даних проєктуються як взаємопов’язані елементи єдиного контуру безпеки.

У межах роботи ставиться задача розроблення засобу захисту даних та протоколу доказу нульового знання для розподілених комп’ютерних систем. Розроблюване рішення передбачає, по-перше, формування формалізованих тверджень (statement) про дані або обчислення, які мають бути перевірені без розкриття секретних значень, та визначення правил їх інтерпретації у вузлах-перевірниках [45, 46]. По-друге, має бути реалізовано процес генерації доказу і його перевірки з прийнятними витратами часу та ресурсів, з урахуванням того, що навантаження на генерацію може бути суттєвим і потребує інженерного планування та оптимізації [43, 44]. По-третє, передбачається інтеграція протоколу з механізмами автентифікації та авторизації, щоб рішення про доступ або

виконання операції приймалося на основі підтверджених властивостей, а не на основі повного розкриття атрибутів [9, 30, 41]. По-четверте, має бути забезпечено транспортний захист взаємодії між вузлами, зокрема на основі сучасних протоколів (QUIC, DTLS 1.3) та узгоджених криптографічних конструкцій для шифрування повідомлень, що дозволяє передавати докази й службові дані без ризику перехоплення або модифікації [24-26].

Важливою складовою поставленої задачі є організація керування ключами та артефактами протоколу (публічними параметрами, версіями схем, ідентифікаторами тверджень). Рішення має підтримувати безпечне зберігання, ротацію й контроль доступу до ключового матеріалу, а також передбачати журналювання подій верифікації без включення секретних значень, що підвищує придатність до аудиту та розслідування інцидентів [11, 62]. Для узгодження криптографічної частини з практиками управління безпекою враховуються рамки та контролі систем менеджменту інформаційної безпеки, зокрема підходи ISO/IEC 27001 та каталог контролів ISO/IEC 27002 [1, 2].

Реалізація поставленої задачі передбачає формування логічної структури рішення, у межах якої компоненти виконують чітко визначені функції: модуль формування тверджень і генерації доказів, модуль перевірки доказів у сервісах-споживачах, підсистема керування ключами та параметрами, а також інтерфейс інтеграції з сервісами автентифікації/авторизації і транспортним рівнем [24, 30]. Розмежування ролей дозволяє зберегти прозорість архітектури, спростити тестування та ізолювати критичні криптографічні частини від прикладної логіки. Додатково враховується масштабованість: система має зберігати працездатність при зростанні кількості вузлів і частоти перевірок, що потребує модульності та можливості горизонтального масштабування компонентів [39, 55].

У підсумку розроблюване рішення має забезпечити узгоджену взаємодію між компонентами розподіленої системи, у якій перевірка прав і коректності операцій виконується без надмірного розкриття даних [45, 46, 9].

2 МОДЕЛІ ТА МЕТОДИ ПОБУДОВИ ЗАСОБУ ЗАХИСТУ ДАНИХ І ПРОТОКОЛУ ПРИВАТНОЇ ВЕРИФІКАЦІЇ

2.1 Концепція побудови засобу захисту даних у розподілених комп’ютерних системах

Побудова засобу захисту даних у розподілених комп’ютерних системах потребує такого підходу, за якого безпека розглядається як невід’ємна складова всієї системної взаємодії, а не як окремий допоміжний механізм. Це зумовлено тим, що в розподіленому середовищі обробка й передавання даних відбуваються між великою кількістю вузлів, сервісів і програмних компонентів, які можуть функціонувати в різних доменах довіри, використовувати різні протоколи обміну та мати неоднакові вимоги до збереження конфіденційності інформації. За таких умов традиційні моделі контролю доступу не завжди забезпечують належний рівень захисту, оскільки рішення часто приймається на основі передачі надлишкових атрибутів або розкриття службових характеристик вузла, що само по собі створює додаткові ризики.

У звичайних підходах перевірка права доступу або допустимості певної операції переважно спирається на факт автентифікації суб’єкта, наявність ролі чи подання встановленого набору атрибутів. Проте для розподілених комп’ютерних систем цього часто недостатньо, оскільки важливим є не лише те, хто ініціює дію, а й те, у якому стані перебуває компонент, чи відповідає середовище встановленим вимогам безпеки, чи дотримано політики доступу та чи може бути підтверджена коректність певних властивостей без розкриття конфіденційних даних. Саме тому доцільним є перехід до підходу, за якого перевіряється не повний набір відомостей про вузол або користувача, а лише істинність наперед визначеного твердження про його стан або право на виконання дії.

Концепція побудови такого засобу захисту даних має спиратися на поєднання трьох основних складових: формалізованої політики безпеки, механізму приватної верифікації та програмної архітектури, здатної забезпечити практичну реалізацію цього підходу. Політика безпеки задає умови, за яких дія, запит або доступ можуть

вважатися допустимими. Механізм приватної верифікації забезпечує підтвердження відповідності цим умовам без розкриття секретних значень або повного набору атрибутів. Програмна архітектура повинна підтримувати взаємодію між джерелами даних, модулями формування тверджень, генераторами доказів, перевірниками, засобами журналювання та службами керування ключами.

Ключовою ідеєю запропонованої концепції є подання умови безпеки у вигляді формалізованого твердження, істинність якого підтверджується за допомогою протоколу доказу нульового знання. Таке твердження може описувати наявність права доступу, відповідність конфігурації встановленому профілю, дотримання певних обмежень або коректність окремого обчислення. У цьому випадку сторона, що доводить, формує доказ на основі секретних даних, а сторона, що перевіряє, отримує лише результат верифікації без доступу до внутрішніх значень, які стали підставою для підтвердження. Це дозволяє зменшити обсяг інформації, що розкривається під час взаємодії, та підвищити стійкість системи до компрометації окремих вузлів.

Структурно такий засіб доцільно розглядати як багаторівневу систему (рисунок 2.1). На нижньому рівні розміщуються джерела атрибутів, параметрів стану та службових даних. На середньому рівні виконується формування тверджень і генерація доказів. На верхньому рівні здійснюються перевірка доказів, прийняття рішень, застосування політик і журналювання результатів. Така організація дає змогу відокремити процес підготовки даних від процедур криптографічного підтвердження та процедур прийняття рішень, що є важливим для масштабованості й подальшої інтеграції рішення в реальні програмні системи.

Рисунок 2.1 – Концептуальна схема побудови засобу захисту даних у розподіленій комп’ютерній системі.

Важливою характеристикою концепції є її сумісність із наявними механізмами ідентифікації, авторизації та аудиту. У запропонованому підході протокол доказу нульового знання не замінює повністю ці механізми, а доповнює їх, дозволяючи перейти від моделі повного розкриття атрибутів до моделі вибіркового підтвердження окремих властивостей. Це особливо важливо в тих системах, де рішення про доступ або виконання операції повинно прийматися в умовах обмеженої довіри між учасниками взаємодії. Такий підхід добре узгоджується з принципами Zero Trust, відповідно до яких жоден учасник не

вважається безумовно надійним, а кожна дія має супроводжуватися перевіркою необхідних умов.

Для практичної реалізації концепції важливо також враховувати вимоги до мережевої взаємодії, захищеного транспорту, серіалізації даних і керування ключами. Навіть за умови криптографічної стійкості самого доказу система не може вважатися достатньо надійною, якщо допоміжні дані, публічні параметри або результати перевірки передаються незахищеними каналами або обробляються в неоднозначних форматах. У зв'язку з цим засіб захисту даних повинен розглядатися не лише як криптографічний модуль, а як цілісна програмна система, у якій безпека забезпечується на рівні логіки перевірки, транспортної взаємодії, збереження результатів і контролю критичних компонентів.

Узагальнено концепцію побудови засобу захисту даних можна подати як залежність між політикою безпеки, набором атрибутів, процедурою формування твердження, генерацією доказу та його перевіркою. У спрощеному вигляді це виглядатиме так:

$$R = V(Prf(S, W)), \tag{2.1}$$

де R - результат перевірки та підстава для прийняття рішення;

S - формалізоване твердження про властивість безпеки;

W - секретні дані, що використовуються для побудови доказу;

Prf - процедура генерації доказу;

V - процедура перевірки доказу.

Наведене подання відображає загальну логіку функціонування запропонованого підходу: рішення в системі повинно базуватися не на безпосередньому розкритті даних, а на перевірці доказу істинності твердження про безпечний стан або допустимість дії. Це створює основу для подальшого формування моделі приватної верифікації відповідності компонента політиці безпеки, яка розглядатиметься в наступному підрозділі.

2.2 Модель приватної верифікації відповідності вузла політиці безпеки

Побудова моделі приватної верифікації відповідності компонента політиці безпеки ґрунтується на переході від прямої перевірки набору атрибутів до перевірки істинності формалізованого твердження про стан компонента. У звичайних системах рішення про доступ або виконання операції часто приймається після передавання значної кількості службових відомостей: конфігураційних параметрів, ролей, сертифікатів, ознак середовища виконання або інших атрибутів, що характеризують вузол чи користувача. Такий підхід дає змогу виконати перевірку, однак одночасно збільшує обсяг інформації, яка стає доступною іншій стороні. У розподіленому середовищі це створює додаткові ризики, оскільки навіть допоміжні технічні дані можуть становити цінність для порушника або порушувати вимоги мінімізації розкриття інформації.

Запропонована модель виходить із того, що для прийняття рішення в більшості випадків достатньо не знати всі вхідні параметри компонента, а лише отримати підтвердження, що ці параметри задовольняють встановлену політику безпеки. Отже, центральним елементом моделі стає твердження про відповідність, сформоване в узагальненому й придатному до перевірки вигляді. Воно повинно відображати конкретну безпекову умову, яка має значення для системи. Це може бути належність компонента до дозволеного класу, відповідність конфігурації затвердженому профілю, чинність права доступу, дотримання часових або контекстних обмежень, а також коректність окремих параметрів середовища виконання. У такому підході рішення приймається не на основі розкриття всіх атрибутів, а на основі доведення істинності сформульованої умови.

Модель приватної верифікації передбачає наявність щонайменше трьох логічно пов’язаних складових: політики безпеки, набору атрибутів компонента та механізму формування і перевірки доказу. Політика безпеки визначає, які саме властивості повинні бути підтверджені для дозволу певної дії. Набір атрибутів містить вхідні значення, що характеризують стан компонента. Механізм формування доказу забезпечує криптографічне підтвердження того, що реальні

значення атрибутів відповідають вимогам політики, але без передавання цих значень стороні, яка виконує перевірку. У результаті модель поєднує логіку контролю доступу, вимоги до конфіденційності та процедури математично коректної верифікації в межах єдиної схеми функціонування системи.

У загальному вигляді політику безпеки доцільно подати як множину умов, яким повинен відповідати компонент для виконання запитуваної дії. Нехай компонент характеризується набором атрибутів.

Умову відповідності компонента політиці безпеки виглядає так:

$$S = \begin{cases} 1, \text{якщо } A = C \\ 0, \text{якщо } A \neq C' \end{cases} \quad (2.2)$$

де S - результат логічної оцінки твердження про відповідність;

A - набір атрибутів компонента;

C - множина умов політики безпеки.

Наведений запис відображає базову ідею моделі: якщо реальні атрибути компонента задовольняють усі визначені обмеження, твердження про відповідність вважається істинним. Проте в межах приватної верифікації стороні, що перевіряє, не повинні передаватися самі атрибути A. Замість цього формується доказ того, що для деякого прихованого набору значень умова $A \vDash C$ є істинною. Саме цей перехід від відкритої перевірки до прихованого доведення і становить зміст приватної верифікації. У підсумку зберігається можливість обґрунтовано прийняти рішення, але без надмірного розкриття внутрішнього стану компонента.

З погляду структурної організації моделі доцільно виділити дві сторони взаємодії. Перша сторона є власником атрибутів і виконує побудову доказу. Друга сторона отримує доказ, виконує його перевірку та приймає рішення щодо дозволу, обмеження або відхилення дії. Крім цього, у моделі можуть бути присутні допоміжні компоненти, зокрема сховище політик, служба керування ключами, джерело публічних параметрів і підсистема журналювання результатів перевірки. Така побудова є доцільною для розподілених комп’ютерних систем, оскільки

дозволяє рознести навантаження між окремими сервісами та забезпечити незалежність генерації доказу від процедури прийняття рішення.

Особливістю цієї моделі є те, що політика безпеки повинна бути придатною до формалізації. Це означає, що вона має бути описана не лише мовою організаційних правил, а й у такому вигляді, який можна безпосередньо використати в процедурі побудови твердження. Для цього політика повинна задавати чіткі логічні, порогові, часові або структурні обмеження, що можуть бути перевірені в межах протоколу. Чим точніше визначено умови політики, тим меншою є ймовірність неоднозначної інтерпретації під час перевірки. Унаслідок цього сама модель приватної верифікації стає не лише засобом захисту даних, а й способом формального узгодження політики безпеки з логікою роботи програмної системи.

Ще однією важливою властивістю моделі є вибірковість підтвердження. У багатьох практичних сценаріях системі не потрібно знати точне значення певного параметра, якщо достатньо підтвердити, що це значення належить допустимому діапазону або відповідає встановленому правилу. Наприклад, для прийняття рішення може бути достатньо підтвердити, що версія компонента не нижча за мінімально допустиму, що атрибут чинний у встановлений момент часу або що конфігурація відповідає еталонному профілю. Це дозволяє суттєво зменшити обсяг передаваних даних і зробити перевірку більш приватною без втрати функціонального сенсу.

Для практичної придатності моделі важливою є її сумісність із наявними механізмами ідентифікації та контролю доступу. Результат приватної верифікації повинен бути таким, щоб його можна було використати як підставу для подальшого рішення в прикладній системі. Тобто перевірка доказу має завершуватися не лише логічним значенням істинності, а й придатним для подальшої обробки результатом: дозволом доступу, підтвердженням відповідності, ініціюванням додаткової перевірки або фіксацією відмови. За такого підходу модель не існує окремо від прикладної системи, а безпосередньо вбудовується в її контур безпеки.

Узагальнюючи наведене, модель приватної верифікації відповідності компонента політиці безпеки може розглядатися як формальний механізм, у межах якого істинність твердження про безпечний стан компонента підтверджується без розкриття його внутрішніх атрибутів. Основу цієї моделі становлять політика безпеки, набір прихованих атрибутів, формалізоване твердження, процедура генерації доказу та процедура його перевірки. Такий підхід створює основу для подальшого розроблення методу формування твердження про стан безпеки компонента, що є наступним етапом побудови засобу захисту даних у розподілених комп’ютерних системах.

2.3 Метод формування формалізованих тверджень для перевірки стану безпеки компонента

Формування твердження про стан безпеки компонента є одним із ключових етапів побудови засобу захисту даних у розподілених комп’ютерних системах, оскільки саме на цьому етапі виконується перехід від загальних вимог політики безпеки до формалізованого опису умови, яка надалі може бути перевірена криптографічними засобами. Якщо на попередньому підрозділі було визначено загальну модель приватної верифікації, то в межах цього підпункту увага зосереджується на способі побудови самого твердження, що виступає основою для генерації доказу. Без коректного формування такого твердження неможливо забезпечити ні математичну правильність перевірки, ні її відповідність прикладному змісту політики безпеки.

У загальному випадку твердження про стан безпеки компонента повинно відображати факт відповідності певного набору атрибутів установленим умовам. Однак для використання в засобі приватної верифікації цього недостатньо, оскільки таке твердження має бути придатним до машинної обробки, однозначним з логічного погляду та формально пов’язаним із політикою безпеки, яка діє в системі. Через це метод формування твердження доцільно будувати як послідовність дій, у межах якої виконується відбір релевантних атрибутів,

інтерпретація вимог політики, побудова логічної структури умови та перетворення її у форму, придатну для генерації доказу нульового знання.

Першим етапом методу є виділення множини атрибутів, що реально впливають на прийняття рішення про безпечність компонента. У практичних умовах компонент розподіленої системи може характеризуватися значною кількістю параметрів: версією програмного забезпечення, конфігураційними налаштуваннями, статусом сертифікатів, роллю у системі, належністю до певного домену довіри, часовими характеристиками, ознаками середовища виконання та іншими службовими показниками. Проте не всі ці параметри повинні ставати частиною твердження. У нього включаються лише ті атрибути, які безпосередньо пов’язані з конкретною політикою безпеки та є значущими для прийняття рішення. Такий відбір дозволяє зменшити складність подальшої формалізації та зберегти принцип мінімізації даних.

Другим етапом є формалізація політики безпеки у вигляді системи умов, що можуть бути безпосередньо перевірені. У вихідному вигляді політика безпеки часто подається як набір правил організаційного або технічного характеру, які описують допустиму поведінку, вимоги до конфігурації або умови надання доступу. Проте для побудови твердження ці правила повинні бути переведені у логічні предикати, порогові умови, часові обмеження або співвідношення між значеннями атрибутів. Наприклад, правило про допустимість доступу може бути подано як належність ролі до дозволеної множини, вимога до середовища виконання - як відповідність визначеному профілю, а вимога до технічного стану компонента - як виконання сукупності порогових умов. У підсумку політика безпеки набуває форми, придатної до точного машинного трактування.

На наступному етапі виконується побудова самого твердження як логічного висловлювання про те, що вибраний набір атрибутів задовольняє множину обмежень. У найпростішому випадку це може бути кон’юнкція кількох умов, які мають виконуватися одночасно. В інших випадках твердження може мати складнішу структуру і включати альтернативні варіанти, часові перевірки, вкладені залежності або поєднання кількох груп правил. Головною вимогою на цьому етапі

є однозначність формулювання: твердження не повинно допускати різних тлумачень, оскільки будь-яка логічна невизначеність на етапі побудови автоматично переноситься на етап генерації доказу й може призвести до некоректної перевірки.

Після побудови загальної структури твердження виконується його нормалізація, тобто приведення до стандартизованої форми, придатної для використання в межах протоколу. Цей етап є важливим через те, що одна й та сама умова безпеки може бути виражена в різний спосіб, але для криптографічної верифікації потрібне однозначне подання без структурних суперечностей і неоднозначностей у серіалізації. Нормалізація може включати впорядкування атрибутів, фіксацію структури запису, уніфікацію формату значень, задання єдиних правил кодування і приведення складних умов до канонічного логічного вигляду (рисунок 2.2). Це зменшує ризик логічних помилок на стику між прикладною логікою та криптографічною реалізацією.

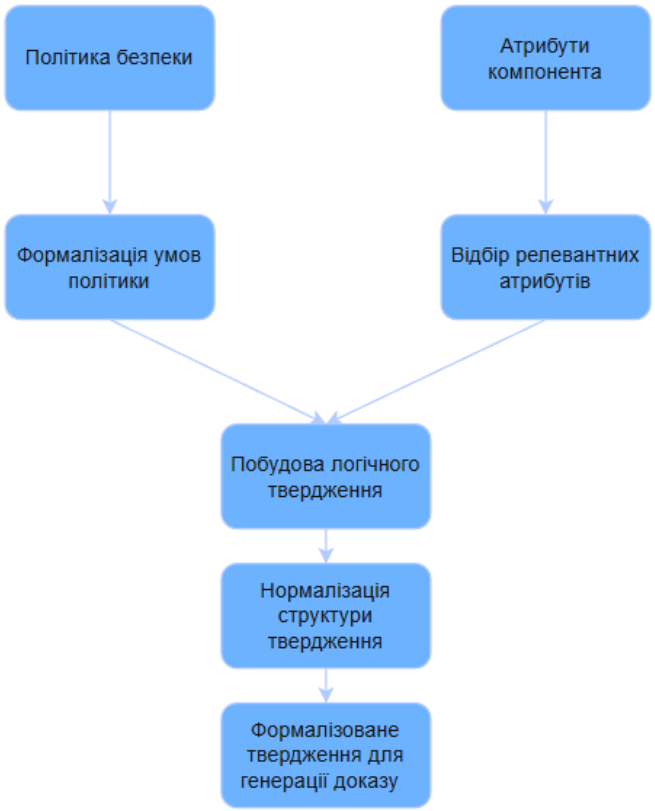


Рисунок 2.2 – Послідовність формування твердження про стан безпеки компонента.

Окрему увагу в межах методу слід приділити питанню розмежування відкритих і прихованих складових твердження. Не всі параметри, пов’язані з перевіркою, повинні бути однаково захищеними. Частина з них може бути віднесена до відкритих службових параметрів, наприклад ідентифікатор політики, версія шаблону перевірки, часовий маркер або тип запитуваної операції. Інша частина, навпаки, повинна залишатися прихованою, оскільки містить значення атрибутів, службові характеристики компонента або інші чутливі відомості. Метод формування твердження повинен чітко розмежовувати ці дві групи, щоб у процесі побудови доказу зберігалася приватність, але водночас залишалася можливість коректної перевірки загального змісту твердження.

Ще однією важливою властивістю методу є повторюваність результату. За однакових умов безпеки, однакової структури політики й однакового набору правил формування система повинна будувати твердження в узгодженому вигляді. Це має значення не лише для криптографічної коректності, а й для подальшого аудиту, журналювання та супроводу програмного засобу. Якщо одна й та сама безпекова умова в різних випадках формулюватиметься по-різному, це ускладнить і перевірку, і аналіз журналів, і підтримку системи в процесі експлуатації. Саме тому метод повинен передбачати єдині правила побудови й оформлення тверджень для всіх типових сценаріїв.

У прикладному аспекті метод формування твердження повинен забезпечувати баланс між точністю перевірки та складністю її реалізації. Надто просте твердження може не охоплювати суттєвих умов політики безпеки, через що перевірка виявиться формально коректною, але практично недостатньою. Надто складне твердження, навпаки, може призвести до збільшення обчислювального навантаження, ускладнення структури доказу та зниження придатності рішення до реального використання. Через це побудова твердження повинна здійснюватися з урахуванням не лише логічної повноти, а й подальшої можливості ефективної генерації та перевірки доказу в межах розподіленого середовища.

У підсумку метод формування твердження про стан безпеки компонента полягає в послідовному переході від політики безпеки та множини релевантних

атрибутів до формалізованого, нормалізованого й придатного до криптографічної верифікації висловлювання. Основне призначення цього методу полягає в тому, щоб забезпечити однозначний опис властивості, істинність якої надалі підтверджується без розкриття внутрішніх даних компонента. Це створює основу для наступного етапу - розроблення методу генерації та перевірки доказу “нульового дня” у межах розподіленої взаємодії.

2.4 Метод генерації та перевірки доказу нульового знання в умовах розподіленої взаємодії

Метод генерації та перевірки доказу “нульового дня” є логічним продовженням побудованої моделі приватної верифікації та методу формування твердження про стан безпеки компонента. Якщо в попередньому підрозділі увага зосереджувалася на тому, яким чином умова політики безпеки переводиться у формалізоване твердження, то на цьому етапі розглядається спосіб підтвердження істинності такого твердження без розкриття внутрішніх атрибутів компонента. Саме цей етап забезпечує практичну реалізацію принципу, за якого система одержує достатню підставу для прийняття рішення, але не отримує надлишкових відомостей про джерело запиту, його конфігурацію чи службові параметри.

Під доказом “нульового дня” у межах цієї роботи доцільно розуміти криптографічне підтвердження того, що певний компонент у момент перевірки відповідає встановленій політиці безпеки, причому така відповідність доводиться без розкриття чутливих характеристик, на основі яких вона була встановлена. Основна ідея полягає в тому, що сторона, яка володіє необхідними атрибутами та службовими параметрами, формує доказ істинності твердження, а сторона, яка приймає рішення, виконує лише процедуру перевірки. У результаті перевіряється не сам набір секретних значень, а факт того, що вони задовольняють задані умови. Це дозволяє використовувати механізм доказу як засіб підтвердження безпечного стану компонента в середовищі, де повна довіра між учасниками відсутня.

Запропонований метод доцільно розглядати як послідовність взаємопов’язаних етапів, що починаються з підготовки вхідних даних і завершуються прийняттям рішення за результатами перевірки. На першому етапі визначається структура твердження, яке повинно бути доведене. Це твердження формується на основі політики безпеки та набору релевантних атрибутів компонента. На другому етапі виконується розмежування відкритих і прихованих параметрів. До відкритої частини можуть належати службові ідентифікатори, версія політики, контекст перевірки або тип операції, тоді як до прихованої частини відносяться реальні атрибути компонента, що не повинні розкриватися в процесі взаємодії. На третьому етапі на основі цих даних виконується генерація доказу. Далі доказ передається стороні, яка перевіряє, після чого виконується процедура верифікації, а її результат використовується як підстава для прийняття системного рішення.

У загальному вигляді процес генерації доказу можна подати як перетворення формалізованого твердження та прихованих значень у криптографічний артефакт, придатний до незалежної перевірки. Нехай S є твердженням про відповідність компонента політиці безпеки, а W - множиною прихованих значень, що підтверджують істинність цього твердження. Тоді процедура генерації доказу подана так:

$$Prf = G(S, W, PP), \tag{2.3}$$

де Prf - сформований доказ;

G - процедура генерації доказу;

S - формалізоване твердження;

W - множина прихованих значень;

PP - публічні параметри протоколу.

У наведеному записі відображено, що результат генерації залежить не лише від змісту самого твердження і прихованих даних, а й від публічних параметрів

протоколу, які забезпечують узгодженість між стороною, що формує доказ, і стороною, що його перевіряє. Це можуть бути параметри схеми доказу, ідентифікатори версій, правила серіалізації або інші службові величини, однаково відомі всім учасникам перевірки.

Після формування доказу виконується процедура його перевірки. Її зміст полягає у визначенні того, чи є отриманий доказ коректним відносно поданого твердження та публічних параметрів. Якщо доказ проходить перевірку, система вважає, що істинність твердження підтверджено, і може використовувати це як підставу для подальшого рішення. Якщо перевірка завершується негативно, дія не повинна бути дозволена без додаткового підтвердження або повторної оцінки стану компонента. Цю процедуру доцільно подати так:

$$R = V(Prf, S, PP), \tag{2.4}$$

де R - результат перевірки;

V - процедура верифікації доказу;

Prf - отриманий доказ;

S - формалізоване твердження;

PP - публічні параметри протоколу.

Такий запис підкреслює, що перевірка не потребує доступу до прихованих значень, оскільки рішення формується лише на основі самого доказу, твердження та узгоджених публічних параметрів. Саме в цьому і полягає ключова перевага підходу: система може перевірити наявність необхідної властивості без розкриття чутливих даних.

З погляду програмної організації метод генерації та перевірки доказу повинен включати кілька функціональних блоків. Перший блок відповідає за підготовку твердження та приведення вхідних даних до канонічного вигляду. Другий блок реалізує власне криптографічну процедуру генерації доказу. Третій блок забезпечує захищене передавання доказу та службових параметрів. Четвертий блок

виконує перевірку коректності отриманого доказу. П’ятий блок інтерпретує результат перевірки в термінах прикладної системи, тобто переводить криптографічний результат у рішення про допуск, обмеження або відмову. Така побудова дозволяє зробити метод придатним до інтеграції в реальне розподілене програмне середовище.

Окремої уваги потребує часовий аспект перевірки. Для систем безпеки важливо не лише підтвердити істинність твердження, а й забезпечити, щоб доказ був актуальним на момент прийняття рішення. Через це до складу відкритих параметрів доцільно включати ознаки контексту перевірки, наприклад часову мітку, ідентифікатор сесії, версію політики або ідентифікатор запиту. Це ускладнює повторне використання застарілих доказів і дозволяє жорсткіше пов’язати результат перевірки з конкретною ситуацією доступу або взаємодії. У підсумку доказ набуває не абстрактного, а контекстно прив’язаного характеру, що є важливим для практичного застосування в розподілених комп’ютерних системах.

Ще однією суттєвою характеристикою методу є баланс між криптографічною стійкістю та обчислювальною придатністю. Генерація доказу в більшості сучасних схем є більш ресурсоємною процедурою, ніж його перевірка, тому навантаження на сторону, що доводить, повинно враховуватися вже на етапі проєктування засобу захисту даних. Це означає, що структура твердження, обсяг прихованих параметрів і складність логічних зв’язків між ними мають бути достатніми для коректної перевірки, але не надмірними з погляду обчислювальних витрат. Такий підхід дозволяє зробити метод не лише теоретично обґрунтованим, а й придатним до використання в системах із реальними обмеженнями на швидкодію та навантаження.

Для практичної експлуатації також важливою є можливість журналювання результатів перевірки без порушення принципу конфіденційності. Журнал не повинен містити прихованих значень або службових деталей, які можуть розкрити внутрішній стан компонента. Водночас у ньому доцільно фіксувати факт перевірки, ідентифікатор твердження, версію політики, часову мітку та результат верифікації. Це забезпечує відтворюваність подій і придатність системи до аудиту без

створення додаткових каналів витоку інформації. Такий підхід особливо важливий для систем, у яких криптографічна перевірка є частиною ширшого контуру контролю доступу та реагування на інциденти.

У підсумку метод генерації та перевірки доказу “нульового дня” полягає в послідовному перетворенні формалізованого твердження та прихованих атрибутів компонента у доказ, що може бути незалежно перевірений без розкриття внутрішніх даних. Основу методу становлять підготовка вхідних параметрів, генерація доказу, його захищене передавання, верифікація та інтерпретація результату в прикладному контексті. Така організація створює основу для подальшого включення результату приватної перевірки в механізми контролю доступу, що розглядатиметься в наступному підрозділі.

2.5 Метод інтеграції результатів приватної верифікації в механізми авторизації та контролю доступу

Інтеграція результатів приватної верифікації в механізми контролю доступу є завершальним етапом побудови засобу захисту даних у розподілених комп’ютерних системах. Якщо на попередніх етапах було визначено політику безпеки, побудовано твердження про стан компонента та описано метод генерації і перевірки доказу “нульового дня”, то на цьому етапі результат криптографічної перевірки повинен бути безпосередньо пов’язаний із прикладною логікою системи. Саме цей зв’язок перетворює доказ нульового знання з окремого криптографічного інструмента на практичний механізм, здатний впливати на надання доступу, запуск сервісних операцій, обмін даними між компонентами або підтвердження права виконання певної дії.

У традиційних системах контролю доступу рішення переважно формується на основі ролі користувача, наявності облікового запису, сертифіката, токена або набору відкрито переданих атрибутів. Такий підхід є достатньо поширеним, проте в розподіленому середовищі він має низку обмежень. По-перше, він часто вимагає передавання значного обсягу службових відомостей, які не є необхідними для

самого факту прийняття рішення. По-друге, він не завжди дозволяє врахувати поточний стан компонента або контекст виконання операції. По-третє, він ускладнює реалізацію принципу мінімізації даних, оскільки перевірка доступу фактично супроводжується частковим розкриттям внутрішньої інформації. Через це доцільним є такий підхід, за якого рішення про доступ приймається не на основі безпосереднього ознайомлення з атрибутами, а на основі результату приватної верифікації їх відповідності політиці безпеки.

Запропонований метод інтеграції ґрунтується на тому, що результат перевірки доказу розглядається як одна з форм підтвердження допустимості дії в системі. У цьому випадку криптографічна верифікація виступає проміжною ланкою між фактичними атрибутами компонента та підсистемою контролю доступу. Компонент, який ініціює дію, не передає повний набір своїх характеристик, а формує доказ істинності твердження про відповідність політиці безпеки. Після перевірки цього доказу система отримує формально обґрунтований результат, який може бути використаний для подальшого ухвалення рішення. За такого підходу доступ надається не через безпосереднє порівняння відкритих атрибутів, а через підтвердження того, що приховані атрибути задовольняють встановлені умови.

Метод доцільно розглядати як послідовність кількох етапів. На першому етапі підсистема контролю доступу отримує запит на виконання певної операції. На другому етапі для цього запиту визначається політика безпеки, яка встановлює перелік умов, що повинні бути підтверджені. На третьому етапі ініціатор запиту формує твердження про свою відповідність цій політиці та генерує доказ “нульового дня”. На четвертому етапі доказ перевіряється відповідним сервісом або модулем верифікації. На п’ятому етапі результат перевірки передається в механізм контролю доступу, який на його основі приймає рішення про дозвіл, відмову або необхідність додаткової перевірки. У підсумку криптографічна перевірка вбудовується в стандартний контур прикладної авторизації, але без вимоги надмірного розкриття атрибутів.

У формалізованому вигляді це можна подати як залежність рішення про доступ від результату перевірки доказу та контексту запиту:

Наведене подання показує, що результат перевірки доказу не існує ізольовано, а інтерпретується в межах конкретного контексту запиту. Це має суттєве значення, оскільки одна й та сама властивість може бути достатньою для дозволу однієї дії, але недостатньою для іншої. Саме тому інтеграція в механізм контролю доступу повинна враховувати не лише факт успішної верифікації, а й тип ресурсу, характер операції, часові обмеження, рівень критичності дії та інші параметри прикладного середовища.

Із погляду архітектури системи метод інтеграції може бути реалізований у кількох варіантах (рисунок 2.3). У першому випадку модуль перевірки доказу виступає окремим сервісом, до якого звертається підсистема контролю доступу перед ухваленням рішення. У другому випадку функції перевірки можуть бути вбудовані безпосередньо в логіку авторизаційного компонента. У третьому випадку перевірка може виконуватися зовнішнім довіреним посередником, який повертає системі лише результат верифікації без передачі самого доказу всім компонентам. Вибір конкретного варіанта залежить від структури розподіленої системи, вимог до продуктивності, рівня централізації та допустимого обсягу довіри до окремих сервісів.

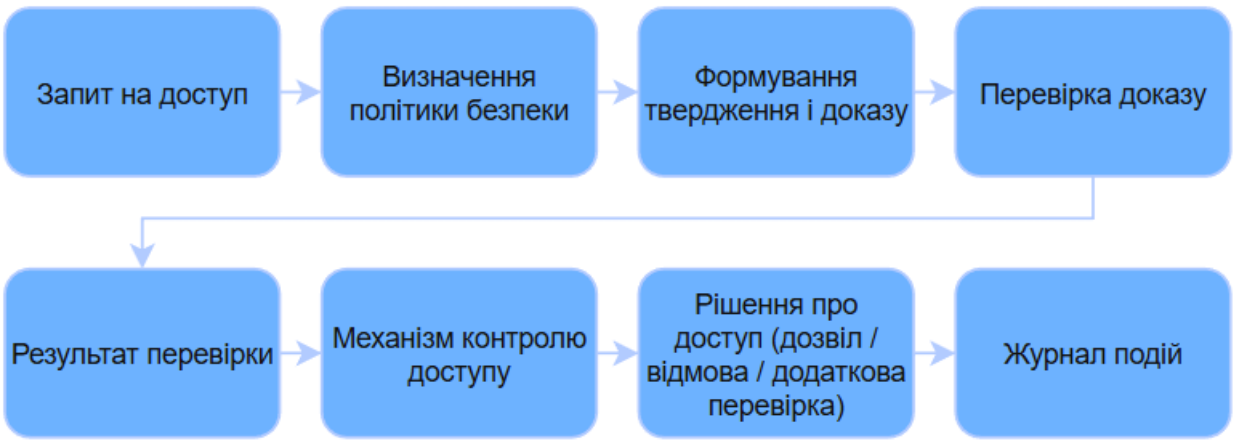


Рисунок 2.3 – Інтеграція результату перевірки доказу в механізм контролю доступу.

Однією з переваг такого підходу є можливість поєднання приватної верифікації з наявними моделями авторизації, а не їх повна заміна. Результат перевірки доказу може виступати як додатковий критерій у ролеорієнтованих, атрибутивних або контекстно-залежних механізмах контролю доступу.

Ще однією важливою властивістю методу є підтримка контекстної перевірки. У розподілених системах рішення про доступ не завжди є статичним. Воно може залежати від часу, типу запиту, попереднього стану компонента, середовища виконання або рівня ризику операції. Саме тому результат доказу має бути пов’язаний із конкретним контекстом, а не існувати як універсальне підтвердження без часової чи ситуаційної прив’язки. Це досягається включенням у відкриті параметри ідентифікатора запиту, часової мітки, версії політики або іншої службової інформації, яка пов’язує перевірку з конкретною операцією. Унаслідок цього зменшується ризик повторного використання доказу в іншому, несанкціонованому контексті.

Для прикладної експлуатації також важливим є журналювання результатів інтеграції. Система повинна фіксувати не лише факт звернення до механізму контролю доступу, а й те, що рішення було прийнято на основі успішної або неуспішної приватної верифікації. При цьому до журналу доцільно вносити ідентифікатор політики, ідентифікатор запиту, часову мітку, результат перевірки та тип ухваленого рішення. Водночас приховані атрибути компонента, внутрішні значення свідка або криптографічні секрети не повинні потрапляти до журналів. Такий підхід забезпечує придатність системи до аудиту й аналізу інцидентів без порушення основного принципу конфіденційності.

Не менш важливим є і питання продуктивності. Оскільки перевірка доказу стає частиною процесу авторизації, вона не повинна створювати надмірної затримки для критичних прикладних операцій. Це означає, що інтеграція повинна враховувати особливості розподілу навантаження між сервісами, можливість кешування публічних параметрів, оптимізацію маршрутів перевірки та відокремлення високочастотних запитів від більш складних сценаріїв із

підвищеними вимогами до безпеки. У підсумку метод інтеграції повинен забезпечувати не лише формальну коректність, а й практичну придатність до використання в системах із реальним робочим навантаженням.

У підсумку метод інтеграції результатів перевірки в механізми контролю доступу полягає у включенні криптографічно підтвердженого результату приватної верифікації до прикладної логіки ухвалення рішень про доступ. Основу цього методу становлять зв’язок між політикою безпеки, контекстом запиту, результатом перевірки доказу та підсистемою авторизації. Такий підхід дозволяє поєднати вимоги конфіденційності, формальної коректності перевірки та практичної керованості доступом у межах єдиної програмної архітектури розподіленої комп’ютерної системи.

2.6 Висновки до розділу 2

У другому розділі було сформовано теоретико-методичну основу побудови засобу захисту даних у розподілених комп’ютерних системах із використанням механізмів приватної верифікації. Розгляд розпочато з визначення загальної концепції побудови такого засобу, відповідно до якої безпека розглядається не як ізольований модуль, а як невід’ємна складова всієї системної взаємодії між компонентами розподіленого середовища. Показано, що в умовах багатовузлової архітектури, різнорівневої довіри між учасниками та потреби у захисті службових і персональних відомостей традиційні підходи до контролю доступу не завжди забезпечують належний баланс між перевіркою та конфіденційністю. У зв’язку з цим обґрунтовано доцільність переходу до моделі, у якій рішення про допустимість дії приймається на основі підтвердження окремих властивостей компонента без розкриття повного набору його атрибутів.

У межах розділу побудовано модель приватної верифікації відповідності компонента політиці безпеки. Встановлено, що центральним елементом такої моделі повинно бути формалізоване твердження про безпечний стан компонента або про допустимість виконання певної дії. На відміну від традиційної перевірки,

за якої іншій стороні передається набір відкритих атрибутів, запропонований підхід передбачає використання доказу, що підтверджує істинність цього твердження без розкриття внутрішніх значень, на основі яких воно сформоване. Це дозволяє розглядати приватну верифікацію як механізм, що одночасно підтримує вимоги конфіденційності, коректності перевірки та придатності до інтеграції у прикладні програмні системи.

На основі сформованого твердження розроблено метод генерації та перевірки доказу “нульового дня”, у межах якого підтверджується відповідність компонента встановленій політиці безпеки без розкриття його прихованих атрибутів. Визначено, що цей метод повинен охоплювати підготовку вхідних даних, розмежування відкритих і прихованих параметрів, генерацію доказу, його передавання захищеним каналом, верифікацію та подальшу інтерпретацію результату перевірки. Показано, що така послідовність дій дозволяє пов’язати криптографічну перевірку з конкретним контекстом запиту, зменшити ризики повторного використання доказів та забезпечити придатність підходу до практичного застосування в умовах розподіленої взаємодії.

Завершальним етапом другого розділу стало формування методу інтеграції результатів перевірки в механізми контролю доступу. Обґрунтовано, що результат криптографічної верифікації повинен використовуватися не ізольовано, а як складова прикладної логіки ухвалення рішень у системі. Це дозволяє включити приватну верифікацію до контуру авторизації, не руйнуючи вже наявні механізми ідентифікації та контролю доступу, а розширюючи їх можливості. Унаслідок цього створено цілісну методичну основу для побудови засобу захисту даних, у якому політика безпеки, формування твердження, генерація доказу, перевірка та прийняття рішення про доступ функціонально пов’язані між собою в межах єдиної програмної архітектури. Це створює основу для переходу до наступного розділу, де розглядатимуться алгоритмічне та технологічне забезпечення розроблюваного програмного засобу.

3 АЛГОРИТМІЧНЕ ТА ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ РОЗРОБЛЮВАНОГО ПРОГРАМНОГО ЗАСОБУ

3.1 Алгоритм формування доказу відповідності політиці безпеки

Алгоритм формування доказу відповідності політиці безпеки є ключовою складовою розроблюваного програмного засобу, оскільки саме на цьому етапі виконується перехід від формалізованого опису умови безпеки до криптографічного підтвердження її істинності. Якщо в другому розділі було обґрунтовано концепцію побудови засобу, визначено модель приватної верифікації, описано метод формування твердження та загальний порядок генерації доказу, то в межах цього підрозділу увага зосереджується вже на алгоритмічній організації відповідного процесу. Основне призначення алгоритму полягає в тому, щоб забезпечити послідовне, однозначне і придатне до програмної реалізації формування доказу на основі політики безпеки, набору атрибутів компонента та відкритих параметрів перевірки.

Вхідними даними для алгоритму є політика безпеки, набір атрибутів компонента, контекст запиту та публічні параметри протоколу. Політика безпеки задає умови, яким повинен відповідати компонент. Атрибути відображають його фактичний стан на момент перевірки. Контекст запиту включає службові дані, що характеризують конкретну операцію, наприклад тип дії, часову мітку, ідентифікатор запиту або версію політики. Публічні параметри визначають умови роботи криптографічної схеми та забезпечують узгодженість між стороною, що формує доказ, і стороною, що його перевіряє. У сукупності ці дані створюють основу для побудови доказу, однак перед початком генерації вони повинні бути приведені до єдиного, несуперечливого формату.

Першим кроком алгоритму є ініціалізація процедури формування доказу. На цьому етапі виконується отримання політики безпеки, вибір відповідного шаблону формування твердження та завантаження актуальної версії публічних параметрів. Після цього виконується перевірка наявності всіх обов’язкових атрибутів компонента, необхідних для побудови твердження. Якщо частина атрибутів

відсутня, пошкоджена або втратила актуальність, подальше формування доказу повинно бути припинене, оскільки в такому випадку неможливо гарантувати коректність результату. Уже на цьому етапі доцільно виконувати базову валідацію структури вхідних даних, щоб уникнути некоректної роботи наступних програмних модулів.

Другим кроком є відбір релевантних атрибутів відповідно до конкретної політики безпеки. Компонент може мати значно більше внутрішніх характеристик, ніж потрібно для конкретної перевірки, тому включення всіх доступних параметрів до процесу генерації доказу є недоцільним. Алгоритм повинен використовувати лише ті значення, які безпосередньо впливають на істинність твердження. Це дозволяє не лише зменшити складність подальших обчислень, а й дотриматися принципу мінімізації даних.

Третім кроком є побудова формалізованого твердження про відповідність компонента політиці безпеки. На цьому етапі виконується зіставлення вибраних атрибутів з умовами політики та формування логічної моделі, яка описує допустимість дії або безпечний стан компонента. Якщо політика містить кілька незалежних або вкладених умов, алгоритм повинен зберігати правильну структуру логічних зв’язків між ними.

Четвертим кроком є поділ параметрів на відкриті та приховані. До відкритих параметрів доцільно відносити ті значення, які необхідні для правильної інтерпретації доказу, але не створюють критичного ризику для конфіденційності. Це можуть бути ідентифікатор політики, ідентифікатор запиту, версія схеми перевірки, часовий маркер або тип виконуваної операції. До прихованих параметрів відносяться атрибути, що характеризують внутрішній стан компонента і не повинні розкриватися стороні, яка виконує перевірку.

П’ятим кроком є нормалізація вхідних структур. На цьому етапі всі відкриті та приховані параметри приводяться до стандартного формату, уніфікується порядок їх розміщення, фіксується структура серіалізації та перевіряється цілісність підготовленого набору даних. Необхідність цього етапу пояснюється тим, що навіть правильно вибрані атрибути можуть бути некоректно

інтерпретовані, якщо їх подання в різних компонентах системи не буде однаковим. Нормалізація забезпечує повторюваність результату та виключає неоднозначності на стику між логічним рівнем побудови твердження та криптографічним рівнем генерації доказу.

Шостим кроком є формування свідка, тобто внутрішнього набору прихованих значень, що використовуються під час побудови доказу. У межах запропонованого алгоритму свідок утворюється на основі відфільтрованих атрибутів компонента, які необхідні для підтвердження істинності твердження. Важливо, щоб до свідка не потрапляли надлишкові значення, не пов'язані з конкретною перевіркою, оскільки це ускладнює алгоритм і збільшує обчислювальне навантаження.

Сьомим кроком є власне генерація доказу. На цьому етапі алгоритм передає сформоване твердження, свідок і публічні параметри в криптографічний модуль, який повертає доказ, придатний до незалежної перевірки. З погляду програмної реалізації цей етап доцільно ізолювати в окремий функціональний блок, щоб забезпечити гнучкість системи у випадку зміни криптографічної бібліотеки або схеми доказу. Результат генерації повинен містити не лише сам доказ, а й супровідну службову інформацію, потрібну для його коректної подальшої інтерпретації в системі.

Узагальнено алгоритм можна подати як послідовність перетворення вхідних параметрів у доказ відповідності:

$$Prf = GenProof(P, A, Q, PP), \tag{3.1}$$

де Prf - сформований доказ відповідності політиці безпеки;

$GenProof$ - процедура формування доказу;

P - політика безпеки;

A - набір атрибутів компонента;

Q - контекст запиту;

PP - публічні параметри протоколу.

У більш деталізованому вигляді логіку роботи алгоритму можна описати так: спочатку з політики безпеки та контексту запиту визначаються необхідні умови перевірки, далі відбираються релевантні атрибути компонента, формується твердження, виділяється свідок, виконується нормалізація структури даних і лише після цього генерується доказ (рисунок 3.1).

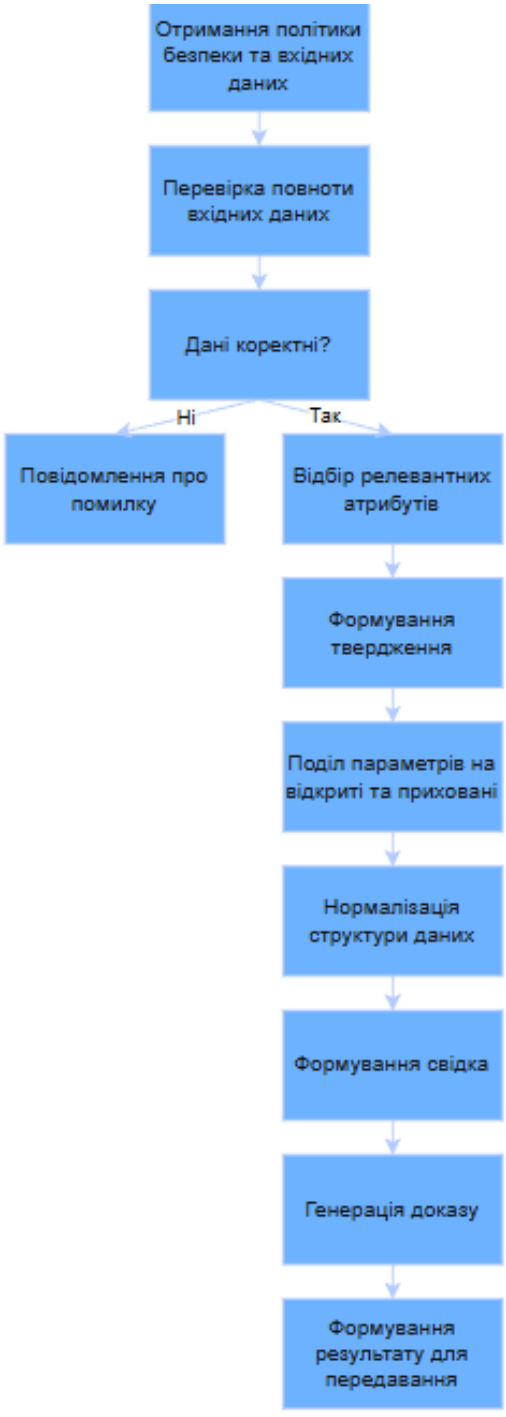


Рисунок 3.1 – Алгоритм формування доказу відповідності політиці безпеки.

З погляду програмної реалізації важливо, щоб алгоритм був не лише коректним, а й достатньо стійким до помилкових або неповних вхідних даних. Саме тому до його структури доцільно включати контрольні перевірки, що фіксують відсутність обов’язкових атрибутів, невідповідність формату, помилки узгодження версій політики або використання застарілих параметрів. Це дозволяє припиняти виконання процедури ще до етапу криптографічних обчислень і не витрачати ресурси на формування доказу, який усе одно не зможе бути коректно перевірений.

Окремої уваги потребує і питання масштабованості. У випадку зростання кількості запитів на генерацію доказів саме цей алгоритм стає одним із найбільш навантажених елементів системи. Через це його структура повинна бути придатною до паралельної обробки незалежних запитів, повторного використання стабільних публічних параметрів та мінімізації дублювання проміжних обчислень. Унаслідок цього підвищується придатність програмного засобу до функціонування в розподіленому середовищі з нерівномірним навантаженням і великою кількістю одночасних перевірок.

У підсумку алгоритм формування доказу відповідності політиці безпеки забезпечує послідовний перехід від політики, атрибутів компонента та контексту запиту до структурованого доказу, придатного до подальшої перевірки. Його основними етапами є підготовка вхідних даних, відбір релевантних атрибутів, побудова твердження, формування свідка, нормалізація параметрів та генерація доказу. Така організація створює алгоритмічну основу для наступного підрозділу, у якому буде розглянуто алгоритм перевірки доказу та прийняття рішення про доступ.

3.2 Алгоритм перевірки доказу та прийняття рішення про доступ

Алгоритм перевірки доказу та прийняття рішення про доступ є наступною ключовою складовою розроблюваного програмного засобу, оскільки саме на цьому етапі результат приватної верифікації перетворюється на практичне управлінське

рішення в межах розподіленої комп’ютерної системи. Якщо алгоритм формування доказу забезпечує підготовку та генерацію криптографічного підтвердження відповідності політиці безпеки, то алгоритм перевірки повинен встановити коректність цього доказу, оцінити його придатність до використання в поточному контексті та передати результат у підсистему контролю доступу. У підсумку саме цей алгоритм визначає, чи буде дозволено виконання операції, чи вона буде відхилена, або ж вимагатиме додаткової перевірки.

Основною особливістю цього алгоритму є те, що він працює не з повним набором внутрішніх атрибутів компонента, а лише з доказом, формалізованим твердженням, публічними параметрами та службовим контекстом запиту. Така побудова дозволяє зберегти конфіденційність прихованих даних, але водночас накладає підвищені вимоги на точність і послідовність самої процедури перевірки. Алгоритм повинен не тільки встановити математичну коректність доказу, а й переконатися, що він сформований саме для тієї політики, того типу операції та того контексту, в межах яких запитується доступ. Отже, перевірка має бути не лише криптографічною, а й логічно прив’язаною до прикладного сценарію використання.

Вхідними даними для алгоритму є отриманий доказ, формалізоване твердження, ідентифікатор або структура політики безпеки, контекст запиту та публічні параметри протоколу. Доказ виступає результатом попереднього етапу і містить криптографічне підтвердження істинності твердження. Формалізоване твердження визначає, яка саме властивість безпеки перевіряється. Політика безпеки задає умови, на підставі яких у системі може бути надано доступ. Контекст запиту включає службові дані, що дозволяють пов’язати перевірку з конкретною операцією, ресурсом або часовим інтервалом. Публічні параметри забезпечують можливість узгодженої перевірки доказу в межах вибраної криптографічної схеми.

Першим етапом алгоритму є приймання й первинна валідація вхідних даних. На цьому кроці перевіряється наявність усіх обов’язкових структур: самого доказу, службових параметрів, ідентифікатора політики, контексту запиту та версії публічних параметрів. Якщо будь-який із цих елементів відсутній або має

некоректний формат, перевірка не повинна продовжуватися. Це пояснюється тим, що навіть математично коректний доказ не може бути використаний у прикладній системі, якщо відсутня можливість однозначно визначити умови його застосування. Отже, уже на початковому етапі алгоритм відсіює пошкоджені, неповні або несумісні запити.

Другим етапом є перевірка узгодженості контексту. На цьому кроці встановлюється, чи відповідає отриманий доказ саме тому запиту, який надійшов до системи. Для цього зіставляються ідентифікатор політики, тип операції, версія схеми перевірки, часові параметри та інші службові ознаки, що зв’язують доказ із конкретною ситуацією доступу. Така перевірка необхідна для того, щоб виключити можливість повторного використання раніше сформованого доказу в іншому контексті. У межах практичної системи саме цей етап забезпечує зв’язок між криптографічним підтвердженням і реальною дією, яку намагається виконати компонент.

Третім етапом є перевірка цілісності та коректності службових структур, пов’язаних із доказом. На цьому кроці перевіряється, чи відповідає доказ очікуваному формату, чи правильно сформовано пов’язані з ним відкриті параметри, чи не порушено порядок серіалізації, а також чи не використовуються застарілі або несумісні публічні параметри. Цей етап має технічний, але дуже важливий характер, оскільки будь-яка невідповідність структури може зробити неможливою подальшу криптографічну верифікацію або призвести до хибного трактування результату.

Четвертим етапом є власне криптографічна перевірка доказу. На цьому етапі алгоритм передає доказ, формалізоване твердження та публічні параметри в модуль верифікації, який повертає логічний результат перевірки. Якщо доказ є коректним, система отримує підтвердження істинності твердження. Якщо перевірка завершується негативно, твердження вважається непідтвердженим, а запит не може бути автоматично допущений до виконання. У межах алгоритму цей етап є центральним, однак він не вичерпує всього процесу прийняття рішення, оскільки

позитивний результат верифікації ще повинен бути правильно інтерпретований у прикладному контексті.

П’ятий етап алгоритму полягає в інтерпретації результату перевірки у межах політики доступу. Якщо результат є позитивним, система повинна визначити, чи достатньо цього для надання доступу до запитуваного ресурсу або виконання операції. У деяких випадках успішна верифікація є достатньою підставою для допуску. В інших випадках вона може виступати лише однією з умов, що доповнюється перевіркою типу ресурсу, режиму роботи системи, рівня критичності операції або наявності інших службових обмежень. Якщо ж результат перевірки є негативним, доступ повинен бути відхилений або переданий на додатковий розгляд відповідно до правил політики безпеки.

Шостим етапом є формування підсумкового рішення. На цьому кроці алгоритм перетворює технічний результат перевірки на конкретну дію системи: дозвіл доступу, відмову або переведення запиту в режим додаткової перевірки. У результаті формується завершений вихід алгоритму, який уже може бути використаний прикладним програмним забезпеченням. Саме на цьому етапі встановлюється остаточний зв’язок між приватною верифікацією та логікою функціонування системи безпеки.

Узагальнено логіку прийняття рішення виглядає так:

$$D = Decide(R, Q, P), \tag{3.2}$$

де D - підсумкове рішення про доступ;

Decide - процедура інтерпретації результату перевірки;

R - результат криптографічної верифікації;

Q - контекст запиту;

P - політика безпеки.

Сьомим етапом є журналювання результатів перевірки (рисунок 3.2). У межах цього етапу до журналу подій доцільно заносити ідентифікатор запиту,

ідентифікатор політики, часову мітку, тип перевірюваної операції та результат прийнятого рішення. При цьому приховані атрибути компонента, внутрішні значення свідка або будь-які секретні дані до журналу потрапляти не повинні. Така організація дозволяє забезпечити придатність системи до аудиту й аналізу подій без створення додаткових ризиків розкриття конфіденційної інформації.

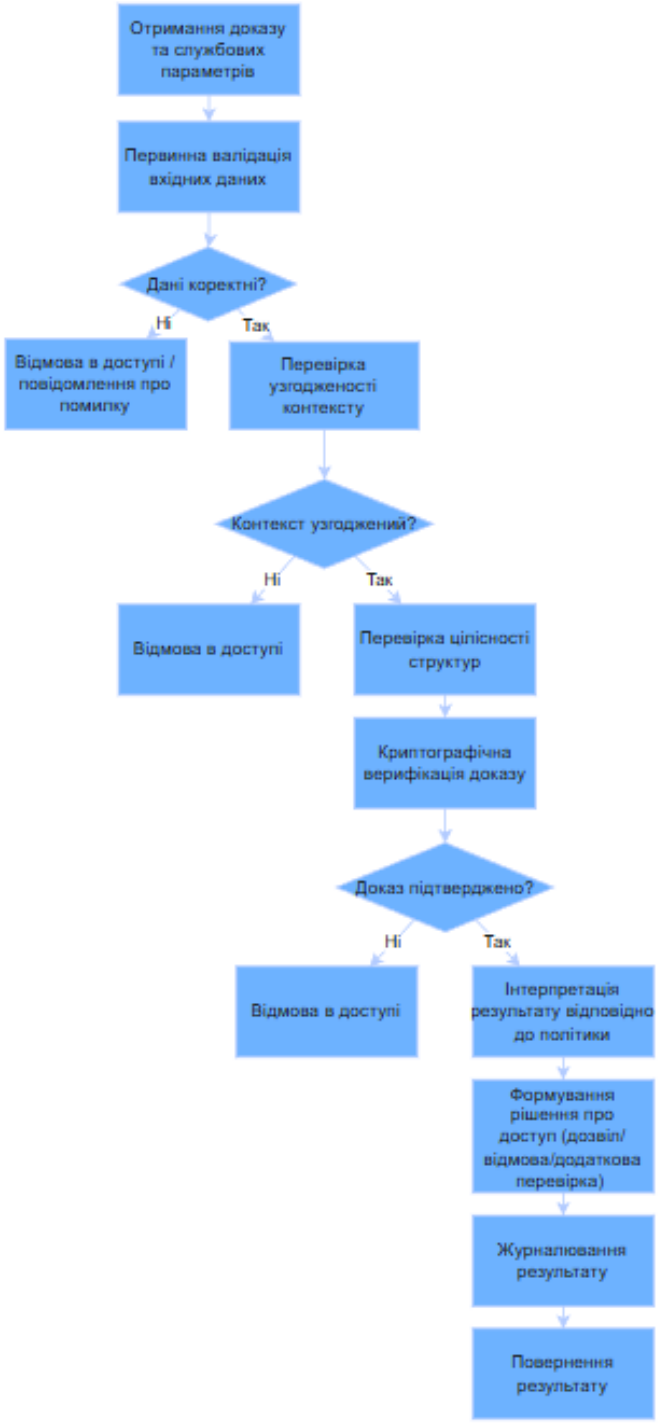


Рисунок 3.2 – Алгоритм перевірки доказу та прийняття рішення про доступ.

З погляду програмної реалізації важливо, щоб алгоритм перевірки працював стабільно навіть у випадку високої інтенсивності запитів. На відміну від етапу генерації доказу, де навантаження переважно зосереджується на стороні, що доводить, перевірка може виконуватися значно частіше й одночасно для багатьох запитів. Через це структура алгоритму повинна підтримувати повторне використання актуальних публічних параметрів, швидке відсіювання некоректних запитів на ранніх етапах і мінімізацію зайвих обчислень до моменту фактичної верифікації. Це підвищує продуктивність системи та робить її більш придатною до практичної експлуатації.

Не менш важливою є стійкість алгоритму до помилкових і навмисно некоректних вхідних даних. У реальному розподіленому середовищі система повинна бути готовою до надходження пошкоджених доказів, застарілих контекстів, невірних службових параметрів або спроб повторного використання раніше сформованих перевірочних структур. Саме тому перевірка не повинна зводитися лише до криптографічного етапу. Наявність попередніх кроків валідації, узгодження контексту й контролю службових даних дозволяє виявляти значну частину таких ситуацій ще до початку основних обчислень.

У підсумку алгоритм перевірки доказу та прийняття рішення про доступ забезпечує послідовний перехід від отриманого криптографічного підтвердження до прикладного рішення в системі безпеки. Його основними етапами є первинна валідація вхідних даних, перевірка узгодженості контексту, контроль цілісності структур, криптографічна верифікація, інтерпретація результату, формування рішення та журналювання події. Така організація дозволяє забезпечити не лише коректність перевірки, а й практичну придатність розроблюваного програмного засобу до використання в умовах розподіленої комп’ютерної системи.

3.3 Розроблення вимог до програмного забезпечення засобу захисту даних

Розроблення вимог до програмного забезпечення засобу захисту даних є необхідним етапом побудови цілісного програмного рішення, оскільки саме на

цьому рівні виконується перехід від загальної моделі, методів і алгоритмів до конкретних характеристик майбутньої системи. Якщо в попередніх підрозділах було визначено логіку формування доказу відповідності політиці безпеки та порядок його перевірки, то в межах цього підрозділу увага зосереджується на тому, якими властивостями повинно володіти програмне забезпечення, щоб забезпечити практичну реалізацію запропонованого підходу. Сукупність таких вимог формує основу для подальшого проєктування структури програмного засобу, виділення його компонентів і вибору технологій реалізації.

Програмне забезпечення засобу захисту даних повинно розглядатися як багатокомпонентна система, що функціонує в умовах розподіленого середовища та забезпечує приватну верифікацію відповідності компонента політиці безпеки без розкриття його внутрішніх атрибутів. Основне призначення такого програмного засобу полягає у підтримці повного циклу перевірки: від отримання вхідних параметрів і політики безпеки до формування твердження, генерації доказу, його перевірки, прийняття рішення про доступ і журналювання результату. Відповідно до цього вимоги до програмного забезпечення повинні охоплювати не лише окремі функціональні можливості, а й питання надійності, масштабованості, безпечності обробки даних, узгодженості між компонентами та придатності до інтеграції в наявну інформаційну інфраструктуру.

Насамперед доцільно визначити функціональні вимоги до програмного забезпечення. Програмний засіб повинен забезпечувати приймання та обробку політик безпеки, відбір релевантних атрибутів компонента, формування формалізованого твердження, підготовку відкритих і прихованих параметрів, генерацію доказу, перевірку доказу та формування результату верифікації. Окремо повинна бути передбачена підтримка механізму прийняття рішення про доступ, який використовує результат перевірки в прикладному контексті. Крім цього, програмне забезпечення повинно забезпечувати журналювання фактів перевірки, фіксацію службових подій, обробку помилок та підтримку службових механізмів адміністрування. У підсумку функціональні вимоги мають охоплювати весь цикл роботи системи, а не лише окрему криптографічну частину.

Однією з базових вимог є модульність структури програмного забезпечення. Це зумовлено тим, що окремі функції системи мають різну природу, різний рівень навантаження та різні вимоги до ізоляції. Наприклад, модуль формування твердження працює переважно з прикладною логікою політик і атрибутів, модуль генерації доказу виконує ресурсоємні криптографічні операції, модуль перевірки доказу орієнтований на швидку обробку запитів, а модуль керування політиками виконує службову координаційну роль. Саме тому програмне забезпечення повинно будуватися так, щоб окремі модулі могли змінюватися, масштабуватися або супроводжуватися незалежно один від одного без порушення цілісності всієї системи.

Важливою функціональною вимогою є підтримка роботи з політиками безпеки у формалізованому вигляді. Програмне забезпечення повинно забезпечувати завантаження, зберігання, ідентифікацію версій, вибір і застосування політик для конкретного запиту або типу операції. Крім цього, повинна підтримуватися можливість перевірки цілісності політики, узгодженості її структури та коректності використання у відповідному сценарії. Це має важливе значення, оскільки результат приватної верифікації безпосередньо залежить від того, яка саме політика безпеки застосовується на момент перевірки. Отже, програмний засіб повинен виключати неоднозначність у виборі правил, на основі яких формується твердження й приймається рішення.

Не менш важливою є вимога до підтримки роботи з атрибутами компонента. Програмне забезпечення повинно вміти отримувати необхідні атрибути з внутрішніх або зовнішніх джерел, перевіряти їх повноту, актуальність і коректність, а також приводити їх до уніфікованої структури, придатної до подальшого використання. У межах цього процесу повинно бути забезпечено чітке розмежування між відкритими службовими параметрами та прихованими значеннями, які входять до свідка. Така вимога має принципове значення для дотримання самої ідеї приватної верифікації, оскільки некоректний поділ параметрів може призвести або до надмірного розкриття інформації, або до неможливості виконати коректну перевірку.

Окрема група вимог стосується криптографічного модуля. Програмне забезпечення повинно забезпечувати генерацію доказу на основі підготовленого твердження та свідка, а також підтримувати його перевірку із використанням узгоджених публічних параметрів. Водночас важливо, щоб криптографічна частина не була жорстко вбудована в усю систему без можливості заміни або оновлення.

Суттєве значення мають нефункціональні вимоги до програмного забезпечення. Насамперед ідеться про продуктивність. Оскільки програмний засіб повинен функціонувати в розподіленому середовищі, де одночасно можуть надходити численні запити на формування та перевірку доказів, він має забезпечувати прийнятний час обробки звернень і не створювати критичних затримок у роботі прикладних сервісів.

Ще однією важливою нефункціональною вимогою є масштабованість. Програмне забезпечення повинно зберігати працездатність у разі зростання кількості компонентів, які використовують механізм приватної верифікації, збільшення частоти запитів або розширення набору підтримуваних політик безпеки. Це означає, що програмний засіб має бути придатним до горизонтального розподілу навантаження, незалежного розгортання окремих модулів та ізольованого масштабування тих компонентів, які стають критичними за продуктивністю.

Не менш суттєвою є вимога надійності. Програмне забезпечення повинно забезпечувати стійку роботу навіть у випадку неповних, некоректних або застарілих вхідних даних. Для цього в його структурі мають бути передбачені механізми валідації запитів, контролю форматів, перевірки версій політик і публічних параметрів, а також керованої обробки помилок. У разі неможливості сформувати або перевірити доказ система повинна завершувати відповідну процедуру контрольованим чином, фіксувати подію в журналі та не допускати переходу до небезпечного або неоднозначного стану. Це забезпечує передбачуваність роботи засобу навіть у випадку збоїв або нестандартних ситуацій.

Окремо слід виділити вимоги до безпечності самого програмного забезпечення. Оскільки система працює з чутливими атрибутами, службовими параметрами та результатами перевірки, програмна реалізація не повинна створювати додаткових каналів витоку інформації. Це означає, що приховані значення повинні оброблятися лише в межах тих компонентів, для яких це дійсно необхідно, а журнали, відповіді інтерфейсів і службові повідомлення не повинні містити секретних даних. Крім цього, програмний засіб повинен обмежувати доступ до модулів керування політиками, ключами й журналами та забезпечувати розмежування прав між різними категоріями користувачів і сервісів.

Для повноцінного функціонування в реальному середовищі програмне забезпечення повинно також відповідати вимозі сумісності з іншими системами. Йдеться про можливість інтеграції із зовнішніми сервісами автентифікації, авторизації, журналювання, моніторингу, керування конфігураціями та прикладними інформаційними системами. У зв’язку з цим необхідно передбачити наявність чітко визначених інтерфейсів взаємодії, узгоджених форматів передавання даних і стабільної логіки обробки відповідей. Це дозволяє використовувати розроблюваний засіб не як ізольований експериментальний модуль, а як повноцінний компонент розподіленої програмної інфраструктури.

Узагальнено вимоги до програмного забезпечення можна подати як сукупність функціональних і нефункціональних характеристик, що визначають його готовність до використання в межах розподіленої комп’ютерної системи:

$$Req = \{F, NF\}, \tag{3.3}$$

де Req - множина вимог до програмного забезпечення;

F - функціональні вимоги;

NF - нефункціональні вимоги.

У більш деталізованому вигляді функціональні вимоги охоплюють обробку політик безпеки, роботу з атрибутами, формування твердження, генерацію та перевірку доказу, прийняття рішення про доступ і журналювання (таблиця 3.1).

Нефункціональні вимоги охоплюють продуктивність, масштабованість, надійність, безпечність реалізації, сумісність та придатність до супроводу. Саме поєднання цих двох груп визначає реальну якість розроблюваного програмного засобу, оскільки навіть функціонально повна система не може вважатися придатною до експлуатації, якщо вона нестійка, надто повільна або складна для інтеграції.

Таблиця 3.1 - Структура вимог до програмного забезпечення засобу захисту даних

<table>
<tr><td colspan="4">Вимоги до програмного забезпечення засобу захисту даних</td></tr>
<tr><td colspan="2">Функціональні вимоги</td><td colspan="2">Нефункціональні вимоги</td></tr>
<tr><td>Обробка політики безпеки</td><td>Робота з атрибутами</td><td>Формування твердження</td><td>Генерація доказу</td></tr>
<tr><td>Перевірка доказу</td><td>Прийняття рішення про доступ</td><td>Журналювання події</td><td>Продуктивність</td></tr>
<tr><td colspan="2" rowspan="3"></td><td>Масштабованість</td><td>Надійність</td></tr>
<tr><td>Безпечність реалізації</td><td>Сумісність</td></tr>
<tr><td colspan="2">Придатність до супроводу</td></tr>
</table>

У підсумку розроблення вимог до програмного забезпечення засобу захисту даних дозволяє конкретизувати, якими властивостями повинна володіти система для реалізації запропонованого підходу в умовах розподіленого середовища. Визначені вимоги створюють основу для наступного етапу - проєктування структури програмного забезпечення та взаємодії його компонентів, де загальні положення буде переведено у вигляд конкретної архітектурної організації розроблюваного засобу.

3.4 Проєктування структури програмного забезпечення та взаємодії його компонентів

Проєктування структури програмного забезпечення засобу захисту даних є етапом, на якому визначені раніше моделі, методи, алгоритми та вимоги переводяться у форму цілісної архітектурної організації. Якщо в попередньому підрозділі було окреслено, якими властивостями повинно володіти програмне забезпечення, то в межах цього підпункту увага зосереджується на тому, яким чином ці властивості реалізуються через взаємодію окремих функціональних компонентів. Основна ідея полягає в побудові такої структури, за якої формування твердження, генерація доказу, його перевірка, прийняття рішення та супровідні службові процеси не зливаються в один монолітний блок, а реалізуються як взаємопов’язані, але логічно відокремлені частини системи.

Побудова такої структури зумовлена особливостями самого завдання. Засіб захисту даних, орієнтований на приватну верифікацію в розподіленому середовищі, повинен одночасно працювати з політиками безпеки, атрибутами компонентів, криптографічними перетвореннями, результатами перевірки, журналами подій і механізмами контролю доступу. Поєднання всіх цих функцій в одному програмному модулі ускладнило б супровід, знизило б гнучкість системи та створило б зайві ризики для безпечності реалізації. Саме тому доцільною є модульна структура, у якій кожен компонент виконує чітко визначену роль і взаємодіє з іншими через узгоджені інтерфейси.

У межах запропонованої структури доцільно виділити кілька основних компонентів. Першим є модуль керування політиками безпеки. Його призначення полягає у зберіганні, виборі, ідентифікації версій і наданні політик для конкретних сценаріїв перевірки. Саме цей модуль забезпечує, щоб усі інші компоненти працювали з актуальними і несуперечливими правилами. Другим є модуль збирання та підготовки атрибутів, який отримує вхідні параметри компонента, виконує їх базову валідацію, відбирає релевантні значення та готує їх до подальшої обробки. Третім є модуль формування твердження, який на основі політики

безпеки та підготовлених атрибутів будує формалізоване висловлювання про стан безпеки компонента.

Четвертим компонентом виступає модуль генерації доказу. Саме він виконує побудову криптографічного підтвердження істинності сформованого твердження на основі прихованих значень і публічних параметрів протоколу. П’ятим компонентом є модуль перевірки доказу, який приймає доказ, твердження та службові параметри, виконує верифікацію і повертає формальний результат перевірки. Шостим компонентом є модуль прийняття рішення, що інтерпретує результат перевірки у прикладному контексті та формує висновок щодо дозволу, відмови або потреби додаткової перевірки.

Така структура дає змогу побудувати засіб захисту даних як послідовний ланцюг обробки запиту. На початковому етапі запит надходить до системи разом із контекстом операції. Далі модуль керування політиками визначає, яка саме політика безпеки повинна бути застосована. Після цього модуль підготовки атрибутів збирає та нормалізує необхідні параметри. Модуль формування твердження будує формалізований опис умови безпеки. На його основі модуль генерації формує доказ. Потім модуль перевірки виконує верифікацію доказу, а модуль прийняття рішення перетворює технічний результат у дію прикладної системи. Усі важливі події при цьому передаються в підсистему журналювання.

З архітектурного погляду важливо, щоб взаємодія між компонентами була організована через чітко визначені інтерфейси. Це означає, що кожен модуль повинен отримувати лише ті вхідні дані, які дійсно потрібні для виконання його функції, і повертати лише той результат, який буде використаний на наступному етапі. Такий підхід зменшує зв’язаність між компонентами, спрощує заміну окремих модулів і знижує ризик несанкціонованого поширення чутливої інформації між частинами системи.

У межах проєктування структури необхідно також врахувати поділ компонентів за рівнем довіри та характером навантаження. Модуль генерації доказу, який працює з прихованими даними, доцільно логічно ізолювати від модулів, що виконують публічну перевірку або прийняття рішення. Це дозволяє

зменшити площину потенційного витоку чутливої інформації. Водночас модуль перевірки доказу, який може обслуговувати велику кількість запитів, повинен бути придатним до масштабування незалежно від інших компонентів.

Наведене подання відображає загальну логіку архітектури, однак у реальній програмній реалізації кожен із цих модулів може складатися з кількох внутрішніх підкомпонентів. Наприклад, модуль підготовки атрибутів може включати засоби збирання даних, валідації та нормалізації. Модуль керування політиками може містити підсистему зберігання правил, контроль версій і механізм вибору політики для конкретного сценарію. Модуль журналювання може бути поєднаний із підсистемою моніторингу, що відстежує навантаження, кількість помилок і стан ключових сервісів.

Окрему увагу під час проєктування структури слід приділити маршрутам проходження даних. Відкриті службові параметри можуть циркулювати між більшістю компонентів системи, тоді як приховані значення повинні оброблятися лише в обмеженому контурі. Це означає, що архітектура повинна чітко розділяти канали передавання відкритих і конфіденційних даних.

Ще однією важливою вимогою до структури є придатність до розширення. Оскільки в подальшому може виникнути потреба у зміні формату політик, додаванні нових типів тверджень, підтримці інших криптографічних реалізацій або ускладненні механізмів прийняття рішень, архітектура не повинна бути жорстко прив’язаною до одного сценарію.

У підсумку проєктування структури програмного забезпечення та взаємодії його компонентів дозволяє сформувати архітектурну основу розроблюваного засобу захисту даних. Запропонована структура забезпечує логічне розмежування функцій, контроль потоків даних, можливість незалежного масштабування критичних модулів і зручність подальшої реалізації. Це створює основу для наступного підрозділу, у якому буде обґрунтовано вибір технологій реалізації програмного засобу.

3.5 Обґрунтування вибору технологій реалізації засобу захисту даних

Вибір технологій реалізації засобу захисту даних має визначатися не лише зручністю розроблення, а насамперед відповідністю функціональним і нефункціональним вимогам, сформованим у попередніх підрозділах. Оскільки розроблюваний програмний засіб повинен забезпечувати формування твердження, генерацію та перевірку доказу, взаємодію між компонентами в розподіленому середовищі, журналювання подій і підтримку подальшого масштабування, технологічна основа рішення має бути достатньо гнучкою, надійною і придатною до модульної організації. У зв’язку з цим вибір технологій доцільно розглядати як окремий етап проєктування, від якого безпосередньо залежить працездатність і подальша розширюваність усього програмного засобу.

Під час вибору технологій для реалізації серверної частини доцільно орієнтуватися на платформу, яка підтримує побудову модульних сервісів, обробку великої кількості запитів і зручну інтеграцію з криптографічними бібліотеками. Для цього найбільш доцільним є використання середовища, яке дозволяє реалізувати окремі модулі формування твердження, генерації доказу, перевірки доказу, роботи з політиками та журналювання як самостійні компоненти з чіткими інтерфейсами взаємодії. Такий підхід узгоджується з раніше визначеною архітектурою і дозволяє уникнути жорсткої прив’язки всієї системи до одного монолітного програмного блоку.

Для реалізації прикладної логіки, обробки запитів і координації взаємодії між компонентами доцільно використовувати мову програмування, яка поєднує достатню швидкість розроблення, розвинену екосистему бібліотек і зручність побудови мережевих сервісів. У межах цієї роботи доцільним виглядає використання Python як базової мови серверної логіки. Такий вибір пояснюється тим, що Python дозволяє швидко реалізувати модулі обробки політик, формування тверджень, валідації параметрів, журналювання та інтеграції між сервісами. Крім цього, середовище Python є зручним для створення прототипів і експериментальної

реалізації програмних рішень, що має важливе значення на етапі розроблення магістерської роботи.

Для побудови вебсервісної взаємодії між компонентами доцільно використовувати FastAPI або інший легкий серверний фреймворк подібного класу. Такий інструмент дозволяє реалізувати чітко структуровані прикладні інтерфейси, через які можуть взаємодіяти модуль генерації доказу, модуль перевірки, підсистема контролю доступу та допоміжні сервіси. Перевагою такого підходу є простота розмежування функцій між компонентами, зручність тестування окремих маршрутів, підтримка серіалізації структурованих даних і можливість подальшого розгортання сервісів у розподіленому середовищі. Це добре відповідає архітектурі, у якій логіка системи поділяється на окремі функціональні модулі.

Оскільки центральною частиною засобу є робота з доказами нульового знання, важливим є вибір криптографічної бібліотеки або інструментального середовища, яке підтримує формування та перевірку відповідних доказів. У межах цієї роботи доцільно орієнтуватися на використання готових бібліотечних засобів або зовнішніх модулів, що забезпечують реалізацію обраної схеми доказу, замість самостійної реалізації криптографічних примітивів з нуля. Такий підхід є більш виправданим, оскільки дозволяє зосередити увагу на прикладній логіці програмного засобу, узгодженні політик безпеки, формуванні тверджень і інтеграції доказу в систему контролю доступу. Крім цього, використання готових перевірених реалізацій зменшує ризик появи помилок у критичних обчислювальних процедурах.

Для збереження політик безпеки, службових конфігурацій, журналів подій і параметрів взаємодії доцільно використовувати систему керування базами даних. З урахуванням структури розроблюваного рішення доцільним виглядає застосування PostgreSQL, оскільки така система забезпечує надійне збереження структурованих даних, підтримує транзакційність, дозволяє працювати з версіями записів і добре підходить для серверних застосунків, орієнтованих на подальше розширення. Використання реляційної бази даних також полегшує підтримку

журналювання, оскільки події верифікації, рішення про доступ і службові записи можуть бути організовані у вигляді логічно пов’язаних таблиць.

Для внутрішнього обміну даними між компонентами доцільно використовувати формат JSON, оскільки він є достатньо простим, поширеним і придатним до інтеграції з вебсервісними інтерфейсами (таблиця 3.2). Використання єдиного формату подання запитів, результатів перевірки, службових параметрів і конфігураційних структур дозволяє зменшити складність інтеграції між модулями. Крім цього, формат JSON є зручним для журналювання, тестування та подальшого аналізу роботи системи. У межах реалізації важливо забезпечити єдині правила формування таких структур, щоб уникнути неоднозначності під час обробки даних різними компонентами програмного засобу.

Окремої уваги потребує вибір засобів контейнеризації та розгортання. Оскільки розроблюваний програмний засіб має модульну структуру і в подальшому може розгортатися як сукупність кількох сервісів, доцільно використовувати Docker для контейнеризації основних компонентів. Це дозволяє забезпечити однакове середовище виконання під час розроблення, тестування та демонстрації працездатності системи.

Крім цього, контейнерний підхід спрощує ізоляцію окремих модулів, керування залежностями й подальше масштабування окремих частин архітектури без змін у загальній логіці роботи програмного засобу.

Обраний набір технологій є доцільним з кількох причин. По-перше, він забезпечує достатню швидкість реалізації і дозволяє побудувати працюючий прототип без надмірного ускладнення архітектури.

По-друге, він підтримує модульний підхід до розроблення, що є важливим для поділу системи на окремі компоненти.

По-третє, він дозволяє поєднати прикладну логіку, мережеву взаємодію, криптографічні операції та збереження службових даних у межах єдиного програмного рішення. По-четверте, така технологічна основа є достатньо гнучкою для подальшого розширення або заміни окремих елементів без повного перегляду всієї системи.

Таблиця 3.2 Технологічна основа реалізації програмного засобу захисту даних

Технологія	Категорія	Призначення
Python	Мова програмування	Реалізація серверної логіки, обробки запитів і внутрішніх сервісів
FastAPI	Серверний фреймворк	Організація API, маршрутів обробки запитів і взаємодії між компонентами
Криптографічний модуль	Засіб побудови доказу	Генерація та перевірка доказу відповідності політиці безпеки
PostgreSQL	Система керування базою даних	Збереження політик, журналів і службових параметрів системи
JSON	Формат даних	Передавання структурованих вхідних і вихідних повідомлень між компонентами
Docker	Засіб контейнеризації	Забезпечення однакового середовища запуску та розгортання програмного засобу
Git	Система контролю версій	Керування змінами у вихідному коді та підтримка процесу розроблення

Для контролю вихідного коду, фіксації змін і підтримки послідовності розроблення доцільно використовувати Git. Це забезпечує керованість процесу реалізації, можливість відстеження змін у структурі проєкту, конфігураціях і програмних модулях.

Такий вибір є важливим не лише з погляду організації розроблення, а й з позиції відтворюваності результатів, оскільки дозволяє фіксувати стан програмного засобу на різних етапах його побудови. Для магістерської роботи це має додаткове значення, оскільки полегшує структурування матеріалу між етапами проєктування, реалізації та перевірки працездатності.

У підсумку вибір технологій реалізації засобу захисту даних повинен ґрунтуватися на вимогах до модульності, надійності, придатності до інтеграції та можливості підтримки криптографічної перевірки в умовах розподіленого середовища.

Використання Python, FastAPI, PostgreSQL, JSON, Docker і Git створює достатню технологічну основу для реалізації розроблюваного програмного засобу та забезпечує узгодженість між архітектурою, алгоритмами і практичною реалізацією. Це створює підґрунтя для переходу до четвертого розділу, у якому розглядатиметься безпосередня програмна реалізація, структура модулів і перевірка працездатності запропонованого рішення.

3.6 Висновки до розділу 3

У третьому розділі було сформовано алгоритмічне та технологічне підґрунтя розроблюваного програмного засобу захисту даних для розподілених комп’ютерних систем.

Послідовно розглянуто алгоритм формування доказу відповідності політиці безпеки, алгоритм перевірки доказу та прийняття рішення про доступ, а також визначено вимоги до програмного забезпечення, його структурну організацію та технологічну основу реалізації. Така побудова дозволила перейти від загальної

моделі приватної верифікації до конкретних механізмів, які можуть бути реалізовані у вигляді цілісного програмного рішення.

У межах розділу встановлено, що програмний засіб повинен забезпечувати узгоджену роботу кількох взаємопов’язаних компонентів: модуля керування політиками безпеки, модуля підготовки атрибутів, модуля формування твердження, модуля генерації доказу, модуля перевірки доказу, модуля прийняття рішення та підсистеми журналювання.

Окрему увагу було приділено технологічному забезпеченню розроблюваного засобу. Обґрунтовано доцільність використання сучасного стеку, що дозволяє поєднати прикладну логіку, сервісну взаємодію, роботу з даними та підтримку криптографічних процедур у межах єдиної архітектури. У підсумку третій розділ сформував практичну основу для переходу до наступного етапу - безпосередньої програмної реалізації засобу захисту даних, опису його модулів та перевірки працездатності в межах обраного сценарію застосування.

4 РЕАЛІЗАЦІЯ ТА ПЕРЕВІРКА ПРАЦЕЗДАТНОСТІ ПРОГРАМНОГО ЗАСОБУ ЗАХИСТУ ДАНИХ

4.1 Програмна реалізація засобу захисту даних і протоколу доказу “нульового дня”

У межах роботи було реалізовано програмний засіб захисту даних, призначений для підтвердження відповідності компонента політиці безпеки без розкриття його внутрішніх атрибутів.

Програмну реалізацію побудовано як модульний серверний застосунок, у якому окремо виділено логіку роботи з політиками безпеки, формування твердження, генерації доказу, перевірки доказу, прийняття рішення та журналювання подій.

Така побудова дозволила зробити систему впорядкованою, зрозумілою з точки зору реалізації та придатною до подальшого розширення.

На відміну від монолітного підходу, де вся обробка поєднується в одному програмному блоці, реалізована структура дала змогу чітко розмежувати функції між окремими частинами програмного засобу та спростити супровід коду.

Під час реалізації проєкту було обрано модульний підхід до організації вихідного коду. Основний застосунок було розміщено в окремому каталозі, де зосереджено маршрути обробки запитів, моделі вхідних і вихідних даних, сервіси роботи з політиками, модулі побудови твердження, генерації та перевірки доказу, а також підсистему журналювання.

Окремо було винесено файли політик, службові журнали та конфігураційні параметри. Унаслідок цього структура проєкту стала логічно впорядкованою, а кожен компонент отримав чітко визначене місце в загальній архітектурі програмного засобу (рисунок 4.1).

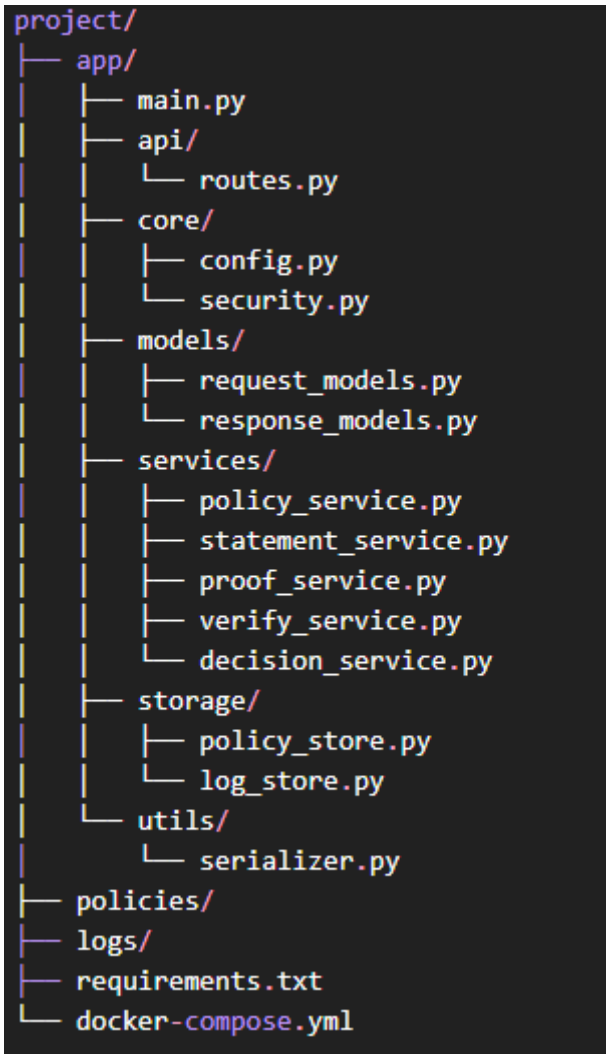


Рисунок 4.1 Загальну структуру проєкту

Наведена структура відображає фактичний поділ програмного засобу на окремі рівні. Основний код було зосереджено в каталозі app, тоді як політики безпеки, журнали подій і параметри запуску винесено окремо. Це дозволило розмежувати прикладну логіку, службові дані та конфігураційне середовище. Файл main.py використано для ініціалізації серверної частини, routes.py - для обробки зовнішніх запитів, а каталог services - для реалізації основного функціоналу системи. Саме в межах сервісного шару було зосереджено виконання ключових операцій: завантаження політики, побудову твердження, генерацію доказу, перевірку та формування підсумкового рішення.

Основну точку входу в систему було реалізовано через серверний модуль, який відповідає за створення застосунку та підключення маршрутів обробки

запитів. Для цього використано окремий файл ініціалізації, у якому налаштовано запуск серверної частини та реєстрацію API-маршрутів. Такий підхід дозволив чітко розділити рівень взаємодії із зовнішнім середовищем і внутрішню логіку системи. Унаслідок цього структура реалізації стала більш прозорою, а маршрути обробки не були перевантажені зайвими службовими деталями.

Для уніфікації роботи з вхідними й вихідними даними в програмному засобі було реалізовано окремі моделі запитів і відповідей. Це дозволило задати єдину форму даних, з якими працює система під час перевірки. Вхідний запит містить ідентифікатор політики, набір атрибутів компонента та контекст виконуваної операції. Вихідна відповідь містить результат перевірки, рішення про доступ і службове повідомлення. Використання таких моделей дозволило забезпечити єдину структуру даних для всіх запитів, що надходять до системи. Це спростило перевірку коректності вхідної інформації ще на початковому етапі обробки та зменшило ризик появи помилок, пов’язаних із неповнотою або неправильним форматом параметрів. Крім цього, сам код став більш впорядкованим, оскільки на рівні моделі було чітко визначено, які саме дані система отримує і який результат повинна повертати після завершення верифікації.

Для взаємодії із зовнішніми компонентами в програмному засобі було реалізовано окремий маршрут перевірки. Через цей маршрут прикладна система або інший сервіс передає параметри, необхідні для приватної верифікації. Логіку маршруту було зведено до приймання запиту й передачі його до відповідного сервісного модуля:

```
from fastapi import APIRouter
from app.models.request_models import VerificationRequest
from app.models.response_models import VerificationResponse
from app.services.decision_service import process_verification

router = APIRouter()

@router.post("/verify", response_model=VerificationResponse)
def verify_component(payload: VerificationRequest):
    return process_verification(payload)
```

У наведеному фрагменті видно, що зовнішній запит надходить на маршрут /verify, після чого передається у внутрішній сервіс обробки. Така організація дозволила залишити API-рівень компактним і не перевантажувати його прикладною логікою. Уся основна обробка була перенесена до сервісного шару, що добре узгоджується з попередньо визначеною архітектурою програмного засобу.

Координацію основного сценарію роботи було реалізовано в окремій сервісній функції, яка послідовно виконує завантаження політики, формування твердження, генерацію доказу, його перевірку, прийняття рішення та збереження результату в журналі. У спрощеному вигляді логіку цієї функції було організовано так:

```
    def process_verification(payload):
        policy = load_policy(payload.policy_id)
        statement = build_statement(policy, payload.attributes,
payload.context)
        proof = generate_proof(statement, payload.attributes)
        verified = verify_proof(statement, proof)
        decision = make_decision(verified, payload.context, policy)
        save_log(payload, verified, decision)
        return {
            "verified": verified,
            "decision": decision,
            "message": "Перевірку завершено"
        }
```

Наведений фрагмент відображає загальну послідовність роботи програмного засобу. У межах одного сценарію система проходить усі основні етапи, які були описані в попередніх розділах: від отримання політики та атрибутів до формування результату перевірки. Саме така реалізація дозволила забезпечити логічну завершеність програмного рішення та показати, що розроблений підхід доведено до рівня працюючої послідовності дій, а не лише теоретичної моделі.

Окремим елементом реалізації став модуль конфігурації, у якому було винесено службові параметри середовища. Це дозволило уникнути жорсткого закріплення шляхів до політик, журналів та параметрів запуску безпосередньо в прикладному коді. Приклад такого модуля має вигляд:

```
import os

POLICY_PATH = os.getenv("POLICY_PATH", "./policies")
LOG_PATH = os.getenv("LOG_PATH", "./logs/app.log")
APP_HOST = os.getenv("APP_HOST", "0.0.0.0")
APP_PORT = int(os.getenv("APP_PORT", 8000))
```

Винесення конфігураційних значень в окремий модуль дозволило зробити програмний засіб більш гнучким у використанні. Унаслідок цього запуск і тестування системи могли виконуватися в різних середовищах без зміни основного коду застосунку. Це також підтвердило, що реалізація розглядалася не як одноразовий демонстраційний приклад, а як повноцінний серверний застосунок із підготовленою конфігурацією та впорядкованою структурою.

Окремо в межах проєкту було організовано зберігання політик безпеки та журналів подій. Політики розміщено в окремому каталозі у структурованому форматі, що дозволило завантажувати їх за ідентифікатором і використовувати в процесі перевірки. Журнали було винесено в окрему директорію, де фіксуються факти перевірок, результати верифікації та підсумкові рішення системи. Така організація дозволила не змішувати службові дані з основним кодом застосунку та забезпечила зрозумілу внутрішню структуру проєкту.

У підсумку в межах цього підрозділу було реалізовано загальну програмну основу засобу захисту даних у вигляді модульного серверного проєкту з чітко визначеною структурою каталогів, моделей, маршрутів, сервісів і конфігурацій. Такий підхід дозволив побудувати впорядковану й придатну до супроводу систему, у якій кожен функціональний блок виконує окрему роль.

4.2 Реалізація модулів формування тверджень, генерації та перевірки доказів

У межах програмної реалізації основну функціональну роль було зосереджено в трьох взаємопов’язаних модулях: модулі формування твердження,

модулі генерації доказу та модулі перевірки доказу. Саме ці компоненти забезпечили практичне втілення раніше розробленої моделі приватної верифікації, оскільки дозволили перейти від набору атрибутів і політики безпеки до перевірюваного результату без розкриття внутрішніх значень компонента. Реалізацію цих модулів було побудовано так, щоб кожен із них виконував окрему функцію, але водночас міг працювати в межах єдиного ланцюга обробки запиту.

Першим етапом обробки було формування твердження про відповідність компонента політиці безпеки. Для цього в системі було реалізовано окремий сервіс, який приймає політику, набір атрибутів та контекст запиту, після чого формує структурований опис умови, істинність якої надалі повинна бути підтверджена. У межах реалізації твердження подається не у вигляді вільного тексту, а як впорядкована структура з ідентифікатором політики, набором умов, службовими параметрами та контекстом перевірки. Такий підхід дозволив забезпечити однозначність подальшої обробки й уникнути неоднозначностей під час серіалізації даних.

У спрощеному вигляді модуль формування твердження було реалізовано так:

```
    def build_statement(policy: dict, attributes: dict, context: dict) -> dict:
        statement = {
            "policy_id": policy["id"],
            "rules": [],
            "context": {
                "request_id": context.get("request_id"),
                "action": context.get("action"),
                "timestamp": context.get("timestamp")
            }
        }

        for rule in policy["rules"]:
            field_name = rule["field"]
            statement["rules"].append({
                "field": field_name,
                "operator": rule["operator"],
                "expected": rule["value"],
                "actual": attributes.get(field_name)
            })

        return statement
```

Наведений фрагмент показує, що під час побудови твердження кожне правило політики переноситься в структуру перевірки разом із відповідним значенням атрибута компонента. Унаслідок цього формується єдиний об’єкт, який надалі може бути використаний для генерації доказу. Важливим є те, що вже на цьому етапі було забезпечено зв’язок між політикою, фактичними значеннями атрибутів і контекстом запиту. Це дозволило зробити твердження не абстрактним, а прив’язаним до конкретної операції в системі.

Після формування твердження виконувалася його нормалізація. Необхідність цього етапу пояснюється тим, що одна й та сама логічна умова може бути подана в різному порядку або з різними варіантами внутрішнього оформлення, що ускладнює побудову відтворюваного доказу. У межах реалізації було використано окрему службову функцію, яка впорядковує поля твердження, очищує службові пропуски та переводить структуру до єдиного формату. Саме це дозволило забезпечити повторюваність результатів генерації та коректну перевірку на стороні верифікатора.

Для підготовки твердження до передавання в криптографічний модуль було реалізовано серіалізацію структури у канонічному вигляді:

```
import json

def serialize_statement(statement: dict) -> str:
    return json.dumps(statement, sort_keys=True, ensure_ascii=False)
```

Такий підхід дав змогу отримати стабільне текстове подання твердження, яке однаково формується як під час генерації доказу, так і під час подальшої перевірки. Це має важливе значення для всієї реалізації, оскільки невідповідність у серіалізації могла б призводити до різного трактування одного й того самого твердження в різних модулях програмного засобу.

Наступним етапом стала реалізація модуля генерації доказу. У межах цієї роботи сам доказ було подано як криптографічно сформований результат для нормалізованого твердження та прихованих атрибутів. Оскільки головною метою

магістерської роботи є створення прикладного програмного засобу, у реалізації було зосереджено увагу на організації виклику криптографічного модуля, підготовці вхідних параметрів і поверненні структурованого результату, а не на низькорівневій реалізації криптографічних примітивів. Це дозволило зберегти прикладний характер роботи і зосередитися на зв’язку між політикою, твердженням, доказом і рішенням системи.

У спрощеному вигляді модуль генерації доказу було побудовано так:

```
    import hashlib

    def generate_proof(statement: dict, attributes: dict) -> dict:
        serialized = serialize_statement(statement)
        witness = "|".join(str(v) for v in attributes.values())
        raw_data = f"{serialized}:{witness}"
        proof_hash      =      hashlib.sha256(raw_data.encode("utf-
8")).hexdigest()

        return {
            "statement_hash": hashlib.sha256(serialized.encode("utf-
8")).hexdigest(),
            "proof": proof_hash
        }
```

Наведений фрагмент показує загальний принцип реалізації: спочатку формується нормалізоване представлення твердження, після чого до нього додаються внутрішні значення, що виступають спрощеним аналогом свідка, а далі обчислюється підсумкове значення доказу. У реальній системі на цьому місці використовується спеціалізована схема доказу нульового знання, однак у межах реалізованого програмного засобу важливим є сам алгоритмічний маршрут побудови доказу, який зберігає логіку переходу від політики та атрибутів до результату, придатного до перевірки.

Окремо було реалізовано модуль перевірки доказу. Його призначення полягає в тому, щоб отримати сформоване твердження і доказ, перевірити їх узгодженість та повернути логічний результат, який у подальшому використовується модулем прийняття рішення. На рівні програмної реалізації це означало побудову функції, яка повторно виконує нормалізацію твердження,

відтворює службові контрольні значення та порівнює їх із тими, що містяться в отриманому доказі.

У спрощеному вигляді перевірку було реалізовано так:

```
    def verify_proof(statement: dict, proof_data: dict) -> bool:
        serialized = serialize_statement(statement)
        recalculated_statement_hash = hashlib.sha256(
            serialized.encode("utf-8")
        ).hexdigest()

        return recalculated_statement_hash ==
proof_data["statement_hash"]
```

У цьому фрагменті показано загальний підхід до перевірки: система знову формує канонічне представлення твердження і на його основі обчислює контрольне значення. Далі виконується порівняння з тим значенням, яке надійшло разом із доказом. У повнішій реалізації до цього етапу додається криптографічна верифікація самого доказу, однак уже цей фрагмент демонструє важливий принцип: перевірка здійснюється на основі формалізованого твердження і службових параметрів, а не через пряме розкриття внутрішніх атрибутів компонента.

Для того щоб забезпечити зв’язок між модулями, у системі було реалізовано послідовний маршрут передавання даних. Спочатку модуль формування твердження повертає структурований об’єкт, який далі передається в модуль генерації доказу. Після отримання результату доказ разом із твердженням передається в модуль перевірки. Унаслідок цього кожен етап обробки має власний результат і не дублює функцій іншого компонента. Така організація позитивно вплинула на підтримуваність коду, оскільки кожен модуль можна тестувати окремо й перевіряти незалежно від решти частин системи.

Окрему увагу в реалізації було приділено обробці помилок. У випадку відсутності політики, порожнього набору атрибутів, пошкодженої структури твердження або некоректного формату доказу система не переходить до наступного етапу, а завершує обробку контрольованим чином. Це дозволило

уникнути ситуацій, коли помилка в одному модулі спричиняє некоректну роботу всієї системи. Така поведінка є особливо важливою для програмного засобу захисту даних, оскільки будь-яка неоднозначність у результаті перевірки може призводити до небезпечного стану системи.

Для фіксації стану обробки також було додано прості службові перевірки:

```
def validate_input(payload):
    if not payload.policy_id:
        raise ValueError("Не вказано ідентифікатор політики")
    if not payload.attributes:
        raise ValueError("Набір атрибутів порожній")
    if not payload.context:
        raise ValueError("Відсутній контекст запиту")
```

Наведена функція не є складною, але вона демонструє підхід, використаний у програмному засобі: ще до початку побудови твердження та доказу система перевіряє мінімальну коректність вхідних даних. Це зменшує кількість помилкових викликів внутрішніх модулів і підвищує загальну надійність реалізації.

У підсумку в межах цього підрозділу було реалізовано три ключові модулі програмного засобу: модуль формування твердження, модуль генерації доказу та модуль перевірки доказу. Їх взаємодію побудовано як послідовний ланцюг обробки, у якому кожен компонент виконує чітко визначену функцію. Така реалізація дозволила втілити основну логіку розробленого підходу на практичному рівні й створила основу для реалізації допоміжних підсистем, пов’язаних із керуванням політиками, конфігурацією, ключами та журналюванням подій, що розглядатиметься в наступному підрозділі.

4.3 Реалізація підсистеми керування політиками, ключами та журналювання подій

У межах програмної реалізації окрему увагу було приділено підсистемі керування політиками безпеки, ключами та журналюванням подій, оскільки саме ці складові забезпечують зв’язок між прикладною логікою перевірки, службовими

параметрами середовища та фіксацією результатів роботи системи. Якщо модулі формування твердження, генерації та перевірки доказу реалізують основний маршрут приватної верифікації, то допоміжна підсистема забезпечує керованість цього процесу, повторюваність перевірок і можливість відстеження виконаних операцій. У результаті програмний засіб було побудовано не лише як набір окремих функцій, а як цілісну систему, у якій правила перевірки, службові параметри та результати обробки зберігаються в упорядкованому вигляді.

Першою складовою цієї підсистеми стало керування політиками безпеки. У реалізованому програмному засобі політики було винесено в окремий каталог, де вони зберігаються у структурованому форматі. Такий підхід дозволив відокремити сам програмний код від правил перевірки та зробив можливим оновлення політик без зміни внутрішньої логіки сервісів. Кожна політика містить власний ідентифікатор, назву, перелік правил та службові параметри, що використовуються під час формування твердження. Унаслідок цього в системі було забезпечено чіткий зв’язок між конкретним запитом і тією політикою, на основі якої виконується перевірка.

Для завантаження політики за її ідентифікатором було реалізовано окремий сервісний модуль. Його призначення полягає у відкритті потрібного файла, зчитуванні вмісту та поверненні структури політики для подальшої обробки іншими компонентами системи. У спрощеному вигляді така функція має вигляд:

```
import json
from pathlib import Path
from app.core.config import POLICY_PATH

def load_policy(policy_id: str) -> dict:
    file_path = Path(POLICY_PATH) / f"{policy_id}.json"
    if not file_path.exists():
        raise FileNotFoundError(f"Політику {policy_id} не
знайдено")

    with open(file_path, "r", encoding="utf-8") as file:
        return json.load(file)
```

Наведений фрагмент показує, що політика завантажується не з жорстко вбудованої структури, а з окремого файла, який визначається за ідентифікатором. Це зробило реалізацію більш гнучкою та спростило організацію набору правил перевірки. Крім цього, такий підхід полегшив тестування, оскільки для різних сценаріїв можна було підключати різні файли політик без зміни решти коду.

Для зручності подальшої обробки самі політики було організовано в єдиному форматі. У межах реалізованого проєкту приклад файла політики мав такий вигляд:

```
{
  "id": "policy_secure_node",
  "name": "Перевірка безпечного стану компонента",
  "rules": [
    {
      "field": "status",
      "operator": "eq",
      "value": "trusted"
    },
    {
      "field": "version",
      "operator": "gte",
      "value": 2
    }
  ]
}
```

Така структура дала змогу подати політику як набір формалізованих правил, придатних до машинної обробки. Унаслідок цього модуль формування твердження працював уже не з довільним описом перевірки, а з чітко визначеним набором умов. Це позитивно вплинуло на повторюваність обробки й дозволило зберегти однаковий формат для різних сценаріїв перевірки.

Наступною складовою підсистеми стало керування ключами та службовими параметрами, які використовуються під час генерації і перевірки доказу. У межах цієї роботи було реалізовано спрощений механізм зчитування службових ключів і параметрів середовища з конфігураційного модуля або змінних оточення. Такий підхід дозволив не розміщувати критичні значення безпосередньо в основному коді та забезпечив більш впорядковану організацію конфігурації. Хоча в межах демонстраційної реалізації не вводилася повноцінна зовнішня система керування

секретами, сам принцип розділення прикладної логіки та службових ключових параметрів було збережено.

Для зчитування службових параметрів було використано окрему функцію:

```
import os

def load_security_config() -> dict:
    return {
        "secret_key":  os.getenv("APP_SECRET_KEY",  "local-dev-
secret"),
        "proof_mode": os.getenv("PROOF_MODE", "demo"),
        "policy_version": os.getenv("POLICY_VERSION", "1.0")
    }
```

У цьому фрагменті видно, що система завантажує ключ і службові параметри із середовища виконання. Така організація дала змогу відокремити налаштування безпеки від бізнес-логіки застосунку та зробила систему зручнішою для запуску в різних конфігураціях. Під час локального тестування використовувалися стандартні значення, а в разі зміни умов запуску параметри могли бути перевизначені без редагування основного коду.

Окрему роль у реалізації підсистеми відіграло журналювання подій. Його призначення полягає у фіксації фактів перевірки, результатів верифікації, застосованих політик і підсумкових рішень системи. У межах розробленого засобу журналювання було реалізовано як окремий сервісний модуль, який приймає службову інформацію про завершену операцію та зберігає її у структурованому вигляді. При цьому в журнал не вносяться приховані атрибути, службові свідки або інші конфіденційні значення. Такий підхід дозволив забезпечити контрольованість роботи програмного засобу без порушення принципу мінімізації розкриття даних.

Функцію збереження журналу було реалізовано так:

```
import json
from datetime import datetime
from pathlib import Path
from app.core.config import LOG_PATH
```

```
    def save_log(payload, verified: bool, decision: str) -> None:
        log_entry = {
            "timestamp": datetime.utcnow().isoformat(),
            "policy_id": payload.policy_id,
            "request_id": payload.context.get("request_id"),
            "verified": verified,
            "decision": decision
        }

        log_file = Path(LOG_PATH)
        log_file.parent.mkdir(parents=True, exist_ok=True)

        with open(log_file, "a", encoding="utf-8") as file:
            file.write(json.dumps(log_entry,  ensure_ascii=False)  +
"\n")
```

Наведений фрагмент показує, що для кожної завершеної перевірки формується окремий запис журналу із часовою міткою, ідентифікатором політики, ідентифікатором запиту, результатом верифікації та підсумковим рішенням. Завдяки цьому вдалося забезпечити простий, але надійний механізм фіксації подій. У разі потреби такі записи можна використовувати для аналізу роботи системи, перевірки послідовності виконання операцій або демонстрації працездатності реалізованого засобу.

Для підвищення надійності журналювання в системі було також передбачено формування окремого повідомлення про помилки. Якщо в процесі завантаження політики, підготовки атрибутів, генерації доказу або перевірки виникала помилка, відповідний стан також міг бути зафіксований у журналі. Це дозволило відстежувати не лише успішні перевірки, а й ситуації, у яких обробка запиту була перервана. Такий підхід виявився корисним під час налагодження програмного засобу, оскільки давав можливість швидше локалізувати проблемні місця в маршруті обробки.

У межах реалізації було також забезпечено взаємодію між модулем політик, службовою конфігурацією та журналюванням у межах єдиного сценарію обробки. Після отримання запиту система завантажувала потрібну політику, зчитувала актуальні службові параметри, виконувала формування твердження і перевірку, а далі зберігала підсумковий результат у журналі. Унаслідок цього допоміжна

підсистема стала повноцінною частиною загальної архітектури програмного засобу, а не окремим службовим доповненням.

Для прикладу, зв’язок між цими операціями було реалізовано так:

```
    def process_verification(payload):
        policy = load_policy(payload.policy_id)
        security_config = load_security_config()

        statement = build_statement(policy, payload.attributes,
payload.context)
        proof = generate_proof(statement, payload.attributes)
        verified = verify_proof(statement, proof)

        decision = make_decision(verified, payload.context, policy)
        save_log(payload, verified, decision)

        return {
            "verified": verified,
            "decision": decision,
            "policy_version": security_config["policy_version"]
        }
```

Цей фрагмент добре показує, що підсистема керування політиками, ключами та журналюванням була інтегрована безпосередньо в основний маршрут обробки запиту. Усі допоміжні компоненти працюють узгоджено: політика визначає правила перевірки, конфігурація задає службові параметри середовища, а журналювання фіксує підсумковий результат роботи системи.

У підсумку в межах цього підрозділу було реалізовано допоміжну підсистему, яка забезпечує завантаження й використання політик безпеки, роботу зі службовими параметрами та ключами, а також фіксацію подій у журналі. Така реалізація підвищила керованість програмного засобу, зробила його структуру більш завершеною та забезпечила підтримку основного маршруту приватної верифікації. Створена підсистема стала важливою частиною всього програмного рішення і підготувала основу для подальшого опису налаштування середовища та організації взаємодії компонентів, що розглядатиметься в наступному підрозділі.

4.4 Організація взаємодії програмних компонентів у розподіленому середовищі

Після реалізації основних функціональних модулів програмного засобу було виконано налаштування програмного середовища, необхідного для запуску, тестування та узгодженої взаємодії всіх компонентів системи. У межах цієї роботи програмне середовище було організовано так, щоб забезпечити стабільну роботу серверної частини, доступ до політик безпеки, збереження журналів подій, обробку вхідних запитів і внутрішню взаємодію між модулями формування твердження, генерації доказу, перевірки та прийняття рішення. Такий підхід дозволив розглядати розроблений засіб не як набір ізольованих програмних фрагментів, а як цілісну систему, придатну до практичного запуску.

Основою середовища виконання було обрано віртуалізоване програмне оточення, у якому встановлювалися всі необхідні бібліотеки та залежності проєкту. Це дозволило ізолювати реалізацію від сторонніх пакетів, уникнути конфліктів версій і забезпечити повторюваність запуску. Для проєкту було сформовано окремий файл залежностей, у якому зафіксовано основні компоненти серверної частини, модулі роботи з моделями даних, засоби запуску застосунку та бібліотеки, використані для допоміжної обробки. Така мінімалістична конфігурація також позитивно вплинула на зрозумілість реалізації, оскільки програмний засіб не був перевантажений зайвими залежностями.

Для зручності запуску системи службові параметри було винесено в окремий конфігураційний файл середовища. У ньому задавалися шляхи до політик безпеки, місце збереження журналів, адреса сервера та службові значення, пов’язані з режимом роботи програмного засобу. Використання окремого конфігураційного файла дало змогу змінювати налаштування без редагування основного коду та спростило перенесення проєкту між різними середовищами виконання.

Приклад такого файла мав вигляд:

```
APP_HOST=0.0.0.0
```

```
APP_PORT=8000
POLICY_PATH=./policies
LOG_PATH=./logs/app.log
APP_SECRET_KEY=local-dev-secret
PROOF_MODE=demo
POLICY_VERSION=1.0
```

Зчитування цих параметрів було організовано в окремому модулі конфігурації. Унаслідок цього під час запуску сервіс автоматично отримував необхідні шляхи й службові значення з середовища, що зробило реалізацію більш впорядкованою. Це також спростило перевірку працездатності, оскільки для зміни умов запуску достатньо було змінити конфігураційні значення без втручання у програмну логіку.

Для запуску серверної частини було використано стандартну команду середовища виконання застосунку:

```
uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload
```

Застосування такого способу запуску дало змогу швидко перевіряти зміни під час розроблення та тестування. Параметр автоматичного перезавантаження дозволив повторно запускати сервер після змін у коді без ручного перезапуску. Це виявилося зручним під час налагодження взаємодії між модулями та перевірки окремих маршрутів обробки запитів.

Окремо було налаштовано структуру каталогів для збереження робочих даних. Каталог політик використовувався для розміщення JSON-файлів з описом правил безпеки, тоді як каталог журналів забезпечував фіксацію результатів перевірки. Така організація дозволила чітко розмежувати вихідний код, конфігураційні параметри та службові дані, що позитивно вплинуло на впорядкованість проєкту. Крім цього, під час першого запуску було реалізовано автоматичне створення каталогу журналів у разі його відсутності, що спростило підготовку середовища до роботи.

Важливим етапом стало налагодження внутрішньої взаємодії між компонентами системи. У межах реалізованого засобу маршрут обробки запиту

було побудовано так, щоб дані послідовно проходили через кілька модулів без дублювання функцій. Після надходження запиту зовнішній API-рівень передає його в сервіс обробки. Далі виконується завантаження політики безпеки, формування твердження, генерація доказу, перевірка доказу, формування рішення та збереження запису в журналі. Така послідовність дозволила зробити логіку роботи системи прозорою і керованою.

У спрощеному вигляді організацію взаємодії компонентів було реалізовано так:

```
    def process_verification(payload):
        policy = load_policy(payload.policy_id)
        config = load_security_config()

        statement = build_statement(policy, payload.attributes,
payload.context)
        proof = generate_proof(statement, payload.attributes)
        verified = verify_proof(statement, proof)

        decision = make_decision(verified, payload.context, policy)
        save_log(payload, verified, decision)

        return {
            "verified": verified,
            "decision": decision,
            "policy_version": config["policy_version"]
        }
```

У цьому фрагменті відображено повний маршрут проходження запиту в межах реалізованого програмного засобу. Кожен виклик відповідає окремому модулю, що дозволило зберегти модульність системи й уникнути змішування кількох рівнів логіки в одному фрагменті коду. Саме така організація зробила програмний засіб придатним до подальшого розширення, оскільки заміна або модифікація окремого модуля не вимагає зміни всієї послідовності обробки.

Для забезпечення обміну даними між компонентами було використано єдиний формат структурованих словників і моделей запитів. Це дозволило передавати дані між модулями без додаткових перетворень на кожному етапі. Наприклад, модуль формування твердження повертає словник, який безпосередньо

передається в модуль генерації доказу, а результат перевірки у вигляді логічного значення передається в модуль прийняття рішення. Такий уніфікований підхід спростив внутрішню взаємодію між сервісами та зробив код більш передбачуваним.

Окрему увагу було приділено початковому налаштуванню тестових політик і вхідних запитів. Для перевірки роботи системи було створено набір політик у каталозі policies, після чого через API передавався запит із набором атрибутів і контекстом перевірки. Це дозволило імітувати реальний сценарій роботи програмного засобу: від завантаження політики до отримання рішення про доступ. Такий спосіб перевірки дав змогу одночасно протестувати маршрут API, внутрішню логіку сервісів і роботу журналювання.

Приклад тестового запиту мав такий вигляд:

```
{
  "policy_id": "policy_secure_node",
  "attributes": {
    "status": "trusted",
    "version": 3
  },
  "context": {
    "request_id": "REQ-001",
    "action": "connect",
    "timestamp": "2026-03-25T10:00:00"
  }
}
```

Наведена структура запиту показує, що для роботи системи достатньо передати ідентифікатор політики, атрибути компонента та мінімально необхідний контекст. Унаслідок цього API-запит залишається компактним, а вся складна логіка обробки переноситься до внутрішніх модулів. Це добре узгоджується із загальною архітектурою реалізованого програмного засобу.

Для спрощення перенесення проєкту між різними середовищами виконання було також підготовлено контейнеризовану конфігурацію запуску. У межах роботи це дозволило зафіксувати єдині умови розгортання та відокремити залежності

проєкту від локального оточення розробника. Базова конфігурація контейнерного запуску була оформлена так:

```
version: "3.9"

services:
  app:
    build: .
    ports:
      - "8000:8000"
    env_file:
      - .env
    volumes:
      - ./policies:/app/policies
      - ./logs:/app/logs
```

Такий фрагмент конфігурації показує, що контейнерний запуск забезпечує доступ до політик, журналів і змінних середовища без необхідності вручну налаштовувати кожен елемент окремо. У межах магістерської роботи це є доречним, оскільки підкреслює завершеність програмної реалізації та демонструє, що засіб може бути не лише написаний, а й коректно розгорнутий у контрольованому середовищі.

У підсумку в межах цього підрозділу було виконано налаштування програмного середовища, підготовлено структуру залежностей, організовано конфігурацію службових параметрів і налагоджено взаємодію між основними компонентами програмного засобу. Унаслідок цього система була приведена до стану, придатного для запуску, тестування та подальшої перевірки працездатності. Саме така організація середовища й взаємодії компонентів створила основу для завершального етапу - оцінювання працездатності розробленого програмного засобу, що розглядатиметься в наступному підрозділі.

4.5 Перевірка працездатності та оцінювання характеристик розробленого рішення

Після реалізації основних модулів програмного засобу було виконано перевірку його працездатності в умовах тестового сценарію, наближеного до реальної взаємодії компонентів розподіленої комп’ютерної системи. Основною метою цього етапу стало підтвердження того, що розроблений засіб коректно виконує повний маршрут обробки запиту: приймає вхідні параметри, завантажує політику безпеки, формує твердження, генерує доказ, перевіряє його, приймає рішення про доступ і зберігає результат у журналі. У межах цього підрозділу увагу було зосереджено не на формальному тестуванні окремих функцій, а на перевірці цілісної роботи всієї системи як завершеного програмного рішення.

Для перевірки працездатності було використано тестову політику безпеки, що задавала мінімальний набір умов для підтвердження безпечного стану компонента. Перевірка виконувалася шляхом надсилання HTTP-запиту до серверного маршруту /verify, який виступав основною точкою входу в систему. У запиті передавалися ідентифікатор політики, набір атрибутів компонента та контекст операції. Такий підхід дозволив перевірити одночасно кілька важливих аспектів: коректність API-рівня, узгодженість внутрішнього маршруту обробки, працездатність сервісних модулів і завершеність загальної логіки програмного засобу.

Надсилання тестового запиту виконувалося за допомогою стандартного HTTP-клієнта. Один із прикладів перевірки мав такий вигляд:

```
curl -X POST "http://127.0.0.1:8000/verify" \
-H "Content-Type: application/json" \
-d '{
  "policy_id": "policy_secure_node",
  "attributes": {
    "status": "trusted",
    "version": 3
  },
  "context": {
    "request_id": "REQ-001",
    "action": "connect",
    "timestamp": "2026-03-25T10:00:00"
  }
}'
```

Після надходження такого запиту програмний засіб послідовно виконував усі передбачені етапи обробки. Спочатку завантажувалася політика безпеки, далі формувалося твердження про відповідність компонента встановленим умовам, після цього створювався доказ, виконувався його контроль і на основі отриманого результату формувалося рішення про доступ. Якщо всі етапи проходили без помилок, система повертала структуровану відповідь про успішне завершення перевірки.

Приклад отриманої відповіді мав такий вигляд:

```
{
  "verified": true,
  "decision": "allow",
  "message": "Перевірку завершено"
}
```

Наведений результат свідчить про те, що система коректно обробила вхідний запит, ідентифікувала потрібну політику, виконала внутрішній цикл приватної верифікації та сформувала позитивне рішення про доступ. Для перевірки відмовостійкості було також проведено сценарій із навмисним порушенням умов політики. У цьому випадку частина атрибутів не відповідала встановленим правилам, що мало призвести до негативного результату перевірки.

Приклад такого запиту мав вигляд:

```
{
  "policy_id": "policy_secure_node",
  "attributes": {
    "status": "untrusted",
    "version": 1
  },
  "context": {
    "request_id": "REQ-002",
    "action": "connect",
    "timestamp": "2026-03-25T10:05:00"
  }
}
```

Для цього сценарію програмний засіб повертав негативний результат, оскільки атрибути компонента не відповідали умовам політики безпеки. Відповідь системи в такому випадку мала вигляд:

```
{
  "verified": false,
  "decision": "deny",
  "message": "Перевірку завершено"
}
```

Окремо було перевірено поведінку системи у випадку некоректних вхідних даних. Для цього надсилалися запити без ідентифікатора політики, без контексту або з порожнім набором атрибутів. У таких випадках система не переходила до етапу побудови твердження чи генерації доказу, а завершувала обробку на етапі первинної валідації. Це дозволило підтвердити, що в програмному засобі реалізовано базовий контроль цілісності запиту і що помилкові звернення не призводять до некерованого виконання внутрішніх модулів.

Приклад перевірки валідації можна подати так:

```
def validate_input(payload):
    if not payload.policy_id:
        raise ValueError("Не вказано ідентифікатор політики")
    if not payload.attributes:
        raise ValueError("Набір атрибутів порожній")
    if not payload.context:
        raise ValueError("Відсутній контекст запиту")
```

Під час перевірки працездатності було також підтверджено коректність журналювання подій. Для кожного завершеного запиту система формувала окремий запис із часовою міткою, ідентифікатором політики, ідентифікатором запиту, результатом перевірки та підсумковим рішенням. Це дозволило простежити повну послідовність виконаних операцій і переконатися, що програмний засіб не лише повертає відповідь клієнту, а й фіксує службову інформацію для подальшого аналізу.

Приклад запису в журналі мав такий вигляд:

```
{
  "timestamp": "2026-03-25T10:00:01.245000",
  "policy_id": "policy_secure_node",
  "request_id": "REQ-001",
  "verified": true,
  "decision": "allow"
}
```

Одержані результати перевірки показали, що реалізований програмний засіб коректно функціонує в межах визначеного сценарію. Було підтверджено працездатність маршруту API, узгодженість між модулями завантаження політики, формування твердження, генерації та перевірки доказу, а також коректність формування підсумкового рішення й збереження журналу. Водночас перевірка показала, що система коректно реагує на невідповідність атрибутів вимогам політики та на помилки вхідних даних, не переходячи до небезпечного або неоднозначного стану.

У підсумку перевірка працездатності підтвердила, що розроблений програмний засіб виконує основні функції, закладені в модель і алгоритми попередніх розділів. Система забезпечує приймання запиту, приватну верифікацію відповідності політиці безпеки, формування рішення про доступ і журналювання результатів.

4.6 Висновки до розділу 4

У четвертому розділі було подано програмну реалізацію розробленого засобу захисту даних для розподілених комп’ютерних систем та показано, яким чином запропоновані в попередніх розділах модель, методи й алгоритми були доведені до рівня працездатного програмного рішення. У межах реалізації програмний засіб було побудовано як модульний серверний застосунок із чітким поділом на компоненти обробки запитів, роботи з політиками безпеки, формування твердження, генерації доказу, перевірки доказу, прийняття рішення та

журналювання подій. Така структура дозволила забезпечити впорядкованість проєкту, зрозумілу логіку внутрішньої взаємодії між модулями та придатність системи до подальшого супроводу й розширення.

У межах розділу було розглянуто особливості реалізації ключових програмних модулів. Показано, що формування твердження про стан безпеки компонента, генерація доказу та його перевірка були реалізовані як послідовний маршрут обробки даних, у якому кожен етап має власне функціональне призначення і не дублює роль інших компонентів. Окремо було реалізовано підсистему керування політиками, службовими параметрами та журналюванням, що дозволило пов’язати логіку перевірки з конкретними правилами безпеки та забезпечити фіксацію результатів роботи програмного засобу. Крім цього, було виконано налаштування програмного середовища, підготовлено конфігурацію запуску та організовано взаємодію між компонентами системи в межах єдиного серверного застосунку.

Проведена перевірка працездатності підтвердила, що розроблений програмний засіб коректно виконує повний цикл обробки запиту: приймає вхідні параметри, завантажує політику безпеки, формує твердження, генерує доказ, виконує його перевірку, приймає рішення про доступ і зберігає результат у журналі. Також було встановлено, що система правильно реагує як на коректні запити, так і на випадки невідповідності атрибутів політиці безпеки або некоректності вхідних даних. У підсумку четвертий розділ підтвердив, що в межах магістерської роботи було не лише сформовано теоретичну основу та алгоритмічне забезпечення, а й реалізовано працездатний програмний засіб, який відображає основну ідею приватної верифікації відповідності політиці безпеки без надмірного розкриття внутрішніх атрибутів компонента.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних і практичних напрацювань розроблено засіб захисту даних та протокол доказу “нульового дня” для розподілених комп’ютерних систем, орієнтований на забезпечення приватної верифікації відповідності компонента політиці безпеки без надмірного розкриття його внутрішніх атрибутів. Подальшого розвитку набула модель захисту даних у розподіленому середовищі, у межах якої поєднано формалізоване подання політики безпеки, формування твердження про стан компонента, генерацію доказу, його перевірку та використання результату в механізмі контролю доступу.

Впровадження результатів роботи дозволило сформувати цілісний підхід до перевірки відповідності компонента політиці безпеки без прямого передавання його службових і чутливих параметрів. Розроблений програмний засіб забезпечив послідовну обробку запиту, формування твердження, генерацію доказу, перевірку доказу, прийняття рішення про доступ і журналювання результатів перевірки, що підтвердило практичну придатність запропонованого підходу для використання в розподілених комп’ютерних системах.

Поставлену мету було досягнуто шляхом розв’язання таких основних завдань:

– проаналізовано особливості захисту даних у розподілених комп’ютерних системах, виявлено основні загрози безпеці даних, обмеження традиційних підходів до перевірки доступу та обґрунтовано доцільність використання механізмів приватної верифікації;

– розглянуто сучасні підходи до застосування доказів нульового знання в задачах підтвердження відповідності політиці безпеки та визначено основні вимоги до засобу захисту даних у розподіленому середовищі;

– розроблено концепцію побудови засобу захисту даних, модель приватної верифікації відповідності компонента політиці безпеки, а також метод формування твердження про стан безпеки компонента;

– розроблено метод генерації та перевірки доказу “нульового дня”, який забезпечує підтвердження істинності твердження без безпосереднього розкриття внутрішніх атрибутів компонента;

– розроблено метод інтеграції результатів перевірки в механізми контролю доступу, що дозволяє використовувати результат приватної верифікації в прикладному контурі авторизації;

– побудовано алгоритм формування доказу відповідності політиці безпеки та алгоритм перевірки доказу з подальшим прийняттям рішення про доступ;

– визначено вимоги до програмного забезпечення, спроєктовано структуру програмного засобу та обґрунтовано вибір технологій його реалізації;

– реалізовано програмний засіб захисту даних у вигляді модульного серверного застосунку з виділенням модулів роботи з політиками безпеки, формування твердження, генерації доказу, перевірки доказу, прийняття рішення та журналювання подій;

– виконано перевірку працездатності розробленого програмного засобу в межах тестових сценаріїв, у результаті чого підтверджено коректність обробки запитів, перевірки відповідності політиці безпеки та формування рішень про доступ.

За тематикою кваліфікаційної роботи підготовлено матеріали, що відображають основні результати виконаних напрацювань і можуть бути використані для подальшого розвитку запропонованого підходу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. National Institute of Standards and Technology. The NIST Cybersecurity Framework (CSF) 2.0. NIST, 2024. URL: https://www.nist.gov/cyberframework (дата звернення: 26.02.2026).

2. National Institute of Standards and Technology. NIST CSWP 29: The NIST Cybersecurity Framework (CSF) 2.0. NIST, 2024. URL: https://csrc.nist.gov/pubs/cswp/29/final (дата звернення: 26.02.2026).

3. Cybersecurity and Infrastructure Security Agency. Known Exploited Vulnerabilities (KEV) Catalog. CISA, 2024–2026. URL: https://www.cisa.gov/known-exploited-vulnerabilities-catalog (дата звернення: 26.02.2026).

4. Cybersecurity and Infrastructure Security Agency. Vulnerability Disclosure Policy (VDP) Platform. CISA, 2024–2026. URL: https://www.cisa.gov/vulnerability-disclosure-policy (дата звернення: 26.02.2026).

5. Regulation (EU) 2024/2847 of the European Parliament and of the Council (Cyber Resilience Act). *Official Journal of the European Union*, 2024. URL: https://eur-lex.europa.eu/eli/reg/2024/2847/oj (дата звернення: 26.02.2026).

6. ENISA Threat Landscape 2024. ENISA, 2024. URL: https://www.enisa.europa.eu/publications/enisa-threat-landscape-2024 (дата звернення: 26.02.2026).

7. ENISA Publications (State of Cybersecurity in the Union / NIS2-related materials). ENISA, 2024–2026. URL: https://www.enisa.europa.eu/publications (дата звернення: 26.02.2026).

8. OWASP Top 10 (2025). OWASP Foundation, 2025. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 26.02.2026).

9. OWASP Top 10 for Large Language Model Applications. OWASP Foundation, 2025. URL: https://owasp.org/www-project-top-10-for-large-language-model-applications/ (дата звернення: 26.02.2026).

10. CVSS v4.0: User Guide. FIRST, 2024–2026. URL: https://www.first.org/cvss/ (дата звернення: 26.02.2026).

11. CVE Program (офіційний сайт). CVE, 2024–2026. URL: https://www.cve.org/ (дата звернення: 26.02.2026).

12. RFC 9691: The Secure Shell (SSH) Public Key for X.509 Certificates. IETF, 2024. URL: https://www.rfc-editor.org/rfc/rfc9691 (дата звернення: 26.02.2026).

13. RFC 9632: X.509 Certificates for Secure Shell. IETF, 2024. URL: https://www.rfc-editor.org/rfc/rfc9632 (дата звернення: 26.02.2026).

14. RFC 9582: Post-Quantum Hybrid Key Agreement in TLS 1.3. IETF, 2024. URL: https://www.rfc-editor.org/rfc/rfc9582 (дата звернення: 26.02.2026).

15. CERT-UA (офіційний сайт, публікації та рекомендації). CERT-UA, 2024–2026. URL: https://cert.gov.ua/ (дата звернення: 26.02.2026).

16. Міністерство цифрової трансформації України (матеріали з кібербезпеки/цифрової стійкості). 2024–2026. URL: https://thedigital.gov.ua/ (дата звернення: 26.02.2026).

17. Zhu Y., et al. Teams of LLM Agents can Exploit Zero-Day Vulnerabilities. *arXiv*, 2024. URL: https://arxiv.org/abs/2406.01637 (дата звернення: 26.02.2026).

18. Sayduzzaman M., et al. Interoperability and Explicable AI-based Zero-Day Attacks. *arXiv*, 2024. URL: https://arxiv.org/abs/2408.02921 (дата звернення: 26.02.2026).

19. Sun Y., et al. LLM4Vuln: A Unified Evaluation Framework for Vulnerability Detection. *arXiv*, 2024. URL: https://arxiv.org/abs/2401.16185 (дата звернення: 26.02.2026).

20. Islam N. T., et al. LLM-Powered Code Vulnerability Repair. *arXiv*, 2024. URL: https://arxiv.org/abs/2401.03374 (дата звернення: 26.02.2026).

21. Hu W., et al. Automated CPE Identification and CVE Summaries. *arXiv*, 2024. URL: https://arxiv.org/abs/2405.13568 (дата звернення: 26.02.2026).

22. Zhao M., et al. A Systematic Study on Generating Web Vulnerability Proof. *arXiv*, 2025. URL: https://arxiv.org/abs/2510.10148 (дата звернення: 26.02.2026).

23. Shet A., Alsmadi I. An Empirical Analysis of Zero-Day Vulnerabilities Disclosed by the Zero Day Initiative. *arXiv*, 2025. URL: https://arxiv.org/abs/2512.15803 (дата звернення: 26.02.2026).

24. Nong Y., et al. APPATCH: Automated Adaptive Prompting (оцінювання на наборах zero-day). *arXiv*, 2024. URL: https://arxiv.org/abs/2408.13597 (дата звернення: 26.02.2026).

25. Jiazhen Z., et al. Yama: Precise Opcode-based Data Flow Analysis. *arXiv*, 2024. URL: https://arxiv.org/abs/2410.12351 (дата звернення: 26.02.2026).

26. Wang Z., et al. An Empirical Security Analysis of Two-factor Authentication. *arXiv*, 2024. URL: https://arxiv.org/abs/2411.11551 (дата звернення: 26.02.2026).

27. Pan Y., et al. Shedding LIGHT on Real-World Attacks on Cloudless IoT. *arXiv*, 2025. URL: https://arxiv.org/abs/2501.16784 (дата звернення: 26.02.2026).

28. Lavin R., et al. A Survey on the Applications of Zero-Knowledge Proofs. *arXiv*, 2024. URL: https://arxiv.org/abs/2408.00243 (дата звернення: 26.02.2026).

29. Sheybani N. Zero-Knowledge Proof Frameworks: A Systematic Survey. *arXiv*, 2025. URL: https://arxiv.org/abs/2502.07063 (дата звернення: 26.02.2026).

30. Liang J., et al. SoK: Understanding zk-SNARKs: The Gap. *arXiv*, 2025. URL: https://arxiv.org/abs/2502.02387 (дата звернення: 26.02.2026).

31. Hassanzadeh-Nazarabadi Y., Taheri-Boshrooyeh S. Constraint-Level Design of zkEVMs: Architectures, Trade-offs, and Evolution. *arXiv*, 2025. URL: https://arxiv.org/abs/2510.05376 (дата звернення: 26.02.2026).

32. Ahmadvand M., et al. push0: Scalable and Fault-Tolerant Orchestration for Zero-Knowledge Proof Generation. *arXiv*, 2026. URL: https://arxiv.org/abs/2602.16338 (дата звернення: 26.02.2026).

33. Ahmadvand M., Souto P. Optimizing Optimism: Up to 3.5x Faster zkVM Validity Proofs via Sparse Derivation. *arXiv*, 2025. URL: https://arxiv.org/abs/2510.23172 (дата звернення: 26.02.2026).

34. Zou G., et al. ZeroOS: A Universal Modular Library OS for zkVMs. *arXiv*, 2025. URL: https://arxiv.org/abs/2512.09300 (дата звернення: 26.02.2026).

35. Gassmann T., et al. Evaluating Compiler Optimization Impacts on zkVM Performance. *arXiv*, 2025. URL: https://arxiv.org/abs/2508.17518 (дата звернення: 26.02.2026).

36. Hochrainer C., Wüstholz V., Christakis M. Arguzz: Testing zkVMs for Soundness and Completeness Bugs. *arXiv*, 2025. URL: https://arxiv.org/abs/2509.10819 (дата звернення: 26.02.2026).

37. Hochrainer C., Wüstholz V., Christakis M. Fuzzing Processing Pipelines for Zero-Knowledge Circuits. *arXiv*, 2024. URL: https://arxiv.org/abs/2411.02077 (дата звернення: 26.02.2026).

38. Yang Q., et al. AC4: Algebraic Computation Checker for Circuit Constraints in ZK. *arXiv*, 2024. URL: https://arxiv.org/abs/2403.15676 (дата звернення: 26.02.2026).

39. Cruz A. Towards a zk-SNARK Compiler for Wolfram Language. *arXiv*, 2024. URL: https://arxiv.org/abs/2401.02935 (дата звернення: 26.02.2026).

40. Daftardar A., et al. SZKP: A Scalable Accelerator Architecture for Zero-Knowledge Proofs. *arXiv*, 2024. URL: https://arxiv.org/abs/2408.05890 (дата звернення: 26.02.2026).

41. Butt S. A., et al. Intel FPGA-Based Acceleration of Zero Knowledge Proofs. *arXiv*, 2024. URL: https://arxiv.org/abs/2412.12481 (дата звернення: 26.02.2026).

42. Ahmed A., et al. AMAZE: Accelerated MiMC Hardware Architecture for Zero Knowledge Proofs. *arXiv*, 2024. URL: https://arxiv.org/abs/2411.06350 (дата звернення: 26.02.2026).

43. Guo H., et al. Benchmarking ZK-Friendly Hash Functions and SNARK Toolchains. *arXiv*, 2024. URL: https://arxiv.org/abs/2409.01976 (дата звернення: 26.02.2026).

44. Kuznetsov O., et al. Efficient Zero-Knowledge Proofs for Set Membership in Blockchain-Based Sensor Networks. *arXiv*, 2024. URL: https://arxiv.org/abs/2410.09169 (дата звернення: 26.02.2026).

45. Ramezan G., Meamari E. zk-IoT: Securing the Internet of Things with Zero-Knowledge Proofs on Blockchain Platforms. *arXiv*, 2024. URL: https://arxiv.org/abs/2402.08322 (дата звернення: 26.02.2026).

46. Wu W., et al. Blockchain Based Zero-Knowledge Proof of Location in IoT. *arXiv*, 2024. URL: https://arxiv.org/abs/2406.18389 (дата звернення: 26.02.2026).

47. Tao Y., et al. Zero-Knowledge Proof of Distinct Identity. *arXiv*, 2024. URL: https://arxiv.org/abs/2403.14020 (дата звернення: 26.02.2026).

48. Mastel K., et al. Two prover perfect zero knowledge for MIP*. *arXiv*, 2024. URL: https://arxiv.org/abs/2404.00926 (дата звернення: 26.02.2026).

49. Shi K., et al. On the Relativistic Zero Knowledge Quantum Proofs of Knowledge. *arXiv*, 2024. URL: https://arxiv.org/abs/2409.03635 (дата звернення: 26.02.2026).

50. Weng C.-X., et al. Experimental Relativistic Zero-Knowledge Proofs with Unconditional Security. *arXiv*, 2025. URL: https://arxiv.org/abs/2501.18176 (дата звернення: 26.02.2026).

51. Karthikeyan A., et al. Towards Practical Zero-Knowledge Proof for PSPACE. *arXiv*, 2025. URL: https://arxiv.org/abs/2511.15071 (дата звернення: 26.02.2026).

52. Onur C. B., et al. A Zero-Knowledge Proof of … (ідентифікаційна схема). *arXiv*, 2024. URL: https://arxiv.org/abs/2408.00395 (дата звернення: 26.02.2026).

53. South T., et al. Verifiable Evaluations of Machine Learning Models using zkSNARKs. *arXiv*, 2024. URL: https://arxiv.org/abs/2402.02675 (дата звернення: 26.02.2026).

54. Lee C., et al. End-to-End Verifiable Decentralized Federated Learning (із ZKP). *arXiv*, 2024. URL: https://arxiv.org/abs/2404.12623 (дата звернення: 26.02.2026).

55. Zhang Y., et al. zkVC: Fast Zero-Knowledge Proof for Private and Verifiable Computing. *arXiv*, 2025. URL: https://arxiv.org/abs/2504.12217 (дата звернення: 26.02.2026).

56. Maheri M. M., et al. TeleSparse: Practical Privacy-Preserving Verification of Deep Learning Inference with ZK. *arXiv*, 2025. URL: https://arxiv.org/abs/2504.19274 (дата звернення: 26.02.2026).

57. Wang Y., et al. Zero-Knowledge Proof Based Verifiable Inference of Models. *arXiv*, 2025. URL: https://arxiv.org/abs/2511.19902 (дата звернення: 26.02.2026).

58. Peng Z., et al. A Survey of Zero-Knowledge Proof Based Verifiable Machine Learning. *arXiv*, 2025. URL: https://arxiv.org/abs/2502.18535 (дата звернення: 26.02.2026).

59. Ray D., et al. Computational Attestations of Polynomial Integrity … *arXiv*, 2025. URL: https://arxiv.org/abs/2506.11458 (дата звернення: 26.02.2026).

60. Fu T., et al. Zero-Knowledge Proof-Based Consensus for Blockchain … *arXiv*, 2025. URL: https://arxiv.org/abs/2503.13255 (дата звернення: 26.02.2026).

61. Namazi M., et al. ZKPROV: A Zero-Knowledge Approach to Dataset Provenance for Large Language Models. *arXiv*, 2025. URL: https://arxiv.org/abs/2506.20915 (дата звернення: 26.02.2026).

62. Watanabe H., Uchikoshi M. Generating Privacy-Preserving Personalized Advice with Zero-Knowledge Proofs and LLMs. *arXiv*, 2025. URL: https://arxiv.org/abs/2502.06425 (дата звернення: 26.02.2026).

63. Xue Z., et al. Large Language Models for Zero-Knowledge Proof Code Generation. *arXiv*, 2025. URL: https://arxiv.org/abs/2509.11708 (дата звернення: 26.02.2026).

64. Verma T., et al. Understanding Performance of Zero-Knowledge Proofs on … *arXiv*, 2025. URL: https://arxiv.org/abs/2509.22684 (дата звернення: 26.02.2026).

65. Mo J., et al. MTU: The Multifunction Tree Unit for Accelerating Zero-Knowledge Proofs. *arXiv*, 2025. URL: https://arxiv.org/abs/2507.16793 (дата звернення: 26.02.2026).

66. Soler D., et al. A Privacy-preserving Key Transmission Protocol … using zk-SNARKs. *arXiv*, 2024. URL: https://arxiv.org/abs/2401.16170 (дата звернення: 26.02.2026).

67. Kuznetsov O., et al. Scalable Zero-Knowledge Proofs for Verifying SHA-256. *arXiv*, 2024. URL: https://arxiv.org/abs/2407.03511 (дата звернення: 26.02.2026).

68. Suwannik W., et al. Zero Knowledge Proof for Multiple Sequence Alignment. *arXiv*, 2024. URL: https://arxiv.org/abs/2404.19064 (дата звернення: 26.02.2026).

69. Burgos A., Alchieri E. Privacy-Preserving Smart Contracts … zk-SNARK-Based Recipe. *arXiv*, 2025. URL: https://arxiv.org/abs/2501.03391 (дата звернення: 26.02.2026).

70. Lee S., et al. Spatial Discretization for Fine-Grain Zone Checks with STARKs. *arXiv*, 2025. URL: https://arxiv.org/abs/2512.24238 (дата звернення: 26.02.2026).

71. Oleksak S., et al. Zk-SNARK Marketplace with Proof of Useful Work. *arXiv*, 2025. URL: https://arxiv.org/abs/2510.09729 (дата звернення: 26.02.2026).

72. Bellachia A. A., et al. VerifBFL: Leveraging zk-SNARKs for a Verifiable Federated Learning Framework. *arXiv*, 2025. URL: https://arxiv.org/abs/2501.04319 (дата звернення: 26.02.2026).

73. Commey D., et al. ZKP-FedEval: Verifiable and Privacy-Preserving Federated Learning Evaluation. *arXiv*, 2025. URL: https://arxiv.org/abs/2507.11649 (дата звернення: 26.02.2026).

74. Castillo F., et al. A Framework for Chained Verifiable Computations. *arXiv*, 2025. URL: https://arxiv.org/abs/2504.15717 (дата звернення: 26.02.2026).

75. Lin Z., et al. Binding Agent ID … zkVM-based Code-Level Authentication. *arXiv*, 2025. URL: https://arxiv.org/abs/2512.17538 (дата звернення: 26.02.2026).

76. Li T., et al. Zk-SNARK for String Match. *arXiv*, 2025. URL: https://arxiv.org/abs/2505.13964 (дата звернення: 26.02.2026).

77. Nye L. Zero-Knowledge Proofs in Sublinear Space. *arXiv*, 2025. URL: https://arxiv.org/abs/2509.05326 (дата звернення: 26.02.2026).

78. Nainwal A., Kamble A., Awathare N. A Comparative Analysis of zk-SNARKs and zk-STARKs: Theory and Practice. *arXiv*, 2025. URL: https://arxiv.org/abs/2512.10020 (дата звернення: 26.02.2026).

79. W3C. Verifiable Credentials Data Model v2.0. *W3C Recommendation*, 2025. URL: https://www.w3.org/TR/vc-data-model-2.0/ (дата звернення: 26.02.2026).

80. European Digital Identity Wallet. Discussion Topic G – Zero Knowledge Proof (ARF). 2025. URL: https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/2.8.0/discussion-topics/g-zero-knowledge-proof/ (дата звернення: 26.02.2026).

81. Комп’ютерні інтелектуальні системи та мережі (KICM-2026): XIX Всеукраїнська науково-практична WEB конференція аспірантів, студентів та молодих вчених. URL: https://sites.google.com/view/kicm/ (дата звернення: 26.02.2026).

ДОДАТОК А

(обов'язковий)

Опублікована теза у XIX Всеукраїнській науково-практичній WEB конференції аспірантів, студентів та молодих вчених КОМП'ЮТЕРНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ ТА МЕРЕЖІ

Мельничук В.О., магістрант
Хмельницький національний університет
Клейн О.М., асистент
Хмельницький національний університет

ЗАСІБ ЗАХИСТУ ДАНИХ ТА ПРОТОКОЛ ДОКАЗУ "НУЛЬОВОГО ДНЯ" У РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

У роботі розглянуто проблему підтвердження стану безпеки компонентів у розподілених комп'ютерних системах в умовах загроз "нульового дня". Запропоновано підхід до приватної верифікації відповідності політикам безпеки

на основі протоколів доказу нульового знання, який дозволяє перевіряти критичні властивості вузлів без розкриття чутливих атрибутів конфігурації та деталей середовища.

Розподілені комп'ютерні системи стають домінуючою архітектурою для сервісів, що обробляють дані між незалежними учасниками - мікросервісами, організаціями, хмарними середовищами та крайовими пристроями. У такій моделі дані та керуючі сигнали проходять крізь множину доменів довіри, що ускладнює контроль доступу, забезпечення конфіденційності та підтвердження цілісності. В умовах загроз "нульового дня" додаткового значення набуває можливість швидко й надійно перевіряти, чи відповідає конкретний вузол/компонент політикам безпеки, не покладаючись на неявну довіру до внутрішньої мережі.

Традиційна верифікація відповідності зазвичай вимагає розкриття великого обсягу відомостей: версій пакетів, патч-рівнів, конфігураційних параметрів, переліку залежностей, даних SBOM, політик hardening тощо. Однак таке розкриття формує додаткові ризики: зловмисник або компрометований посередник може використати ці метадані для таргетованих атак, кореляції інфраструктури або підбору експлойтів. Таким чином виникає суперечність між необхідністю прозорої перевірки безпеки та вимогою мінімізувати витік технічних деталей.

Метою роботи є обґрунтування підходу до побудови засобу захисту даних і протоколу доказу "нульового дня" для розподілених систем, у якому перевірка відповідності стану вузла/компонента виконується без надмірного розкриття даних. Як базовий механізм приватної верифікації пропонується використання доказів нульового знання (ZKP) як інструмента підтвердження тверджень про стан безпеки.

У контексті цієї роботи "доказ нульового дня" розглядається не як розкриття деталей уразливості, а як потреба оперативно підтвердити, що компонент перебуває у стані, який знижує ризики експлуатації невідомих або щойно виявлених вразливостей. Практично це зводиться до приватної перевірки тверджень типу:

-вузол використовує дозволені версії бібліотек/контейнерних образів;
критичні залежності відповідають політиці оновлень (патч-рівні не нижче порогу);
-збірка/артефакт підписаний довіреним ключем і відповідає заданому хешу/маніфесту;
-конфігурація містить обов'язкові параметри безпечного режиму (hardening), але конкретні значення не розкриваються.

Ключовий виклик полягає в тому, що перевіряються атрибути самі є чутливою інформацією. Наприклад, повний перелік версій компонентів може бути "картою" інфраструктури, а деталі конфігурації - прямою підказкою до експлуатації. Тому засіб захисту має забезпечити можливість отримати відповідь "так/ні" (або формально верифікований результат) щодо відповідності, не передаючи первинні значення атрибутів.

Протоколи ZKP дозволяють довести істинність твердження без розкриття секретних значень, використовуючи розділення на публічну частину (statement) та конфіденційну частину (witness). Для задачі приватної перевірки відповідності це означає таку модель:

-statement: формалізоване твердження "стан вузла відповідає політиці P" + публічні параметри (ідентифікатор політики, версія правил, публічні ключі підпису, контрольні хеші/коміти тощо);

-witness: конкретні локальні значення (версії, конфіги, маніфести), які не передаються назовні;

-proof: короткий артефакт, який може бути перевірений багатьма вузлами без доступу до witness.

Архітектурно рішення може бути реалізоване як: (1) модуль формування доказів на стороні вузла, що підтверджує відповідність; (2) модуль перевірки доказів у сервісі-споживачі або в "контролері довіри"; (3) підсистема керування політиками та ключами. Політика безпеки задається у вигляді правил, які перетворюються на формальну схему перевірки (circuit/constraints). Важливо, щоб формалізація твердження була однозначною: одна і та сама політика повинна давати однакові результати незалежно від реалізації вузла, інакше виникають логічні "сірі зони" у верифікації.

Для промислового застосування критичними є такі вимоги:

-Ефективність: генерація доказів є дорожчою за верифікацію, тому вузли-продуценти proof потребують планування ресурсів або виділених компонентів.

-Керування ключами і політиками: політики повинні бути версійовані; ключі підпису та параметри мають мати життєвий цикл (ротація, відкликання, аудит доступу).

-Безпечний аудит без витоку: журнали мають фіксувати факт перевірки, ідентифікатор політики/версії, результат валідації та часові мітки, але не містити конфіденційних значень witness.

-Інтеграція з доступом: результати ZKP-перевірки мають бути вбудовані у рішення про авторизацію (policy enforcement), щоб доступ надавався за підтвердженою відповідністю, а не за довірою до "внутрішнього" сегмента.

До обмежень належать складність побудови коректних statement для реальних сценаріїв, потреба в канонічному поданні даних (щоб уникати неоднозначностей), а також компроміси між універсальністю протоколу й вартістю обчислень. Тому практичне рішення повинно починатися з чітко обмеженого набору перевірюваних властивостей (policy set) і поступово розширюватися з урахуванням вимірювань продуктивності.

Запропонований підхід до "доказу нульового дня" у розподілених комп'ютерних системах полягає у приватній верифікації відповідності компонентів політикам безпеки без розкриття чутливих деталей середовища. Використання доказів нульового знання дозволяє зменшити витік метаданих, знизити залежність від довіри до проміжних вузлів та надати формально перевірюваний механізм підтвердження стану безпеки. Разом із тим практична придатність визначається інженерними аспектами: вартістю генерації доказів, коректною формалізацією тверджень, керуванням ключами та організацією

аудиту без витоку конфіденційних значень. Подальший розвиток рішення доцільно спрямувати на оптимізацію продуктивності, розширення набору підтримуваних політик та інтеграцію з механізмами керування ризиками й безпечної розробки.

Список використаних джерел:

1. Souppaya M., Scarfone K., Dodson D. Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities (NIST SP 800-218) [Електронний ресурс]. - Gaithersburg, MD : National Institute of Standards and Technology, 2022. - Режим доступу: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-218.pdf (дата звернення: 02.03.2026).
2. ISO. ISO/IEC 27001:2022 - Information security management systems - Requirements [Електронний ресурс]. - Geneva : International Organization for Standardization, 2022. - Режим доступу: https://www.iso.org/standard/27001 (дата звернення: 02.03.2026).
3. Lavin R., Liu X., Mohanty H., Norman L., Zaarour G., Krishnamachari B. A Survey on the Applications of Zero-Knowledge Proofs [Електронний ресурс]. - arXiv, 2024. - Режим доступу: https://arxiv.org/abs/2408.00243 (дата звернення: 02.03.2026).

ДОДАТОК Б

(обов’язковий)
Сертифікат

ДОДАТОК В

(обов’язковий)
Презентація

Актуальність, мета, об’єкт, предмет

Актуальність

У розподілених системах перевірка доступу часто вимагає передачі надмірної службової інформації, що підвищує ризики витоку даних.

Мета

Підвищення ефективності захисту даних шляхом розроблення засобу приватної верифікації та протоколу доказу “нульового дня”.

Об’єкт

Процес захисту даних у розподілених комп’ютерних системах.

Предмет

Методи, моделі, алгоритми та програмні засоби приватної верифікації відповідності компонентів політиці безпеки.

Засіб захисту даних та протокол доказу “нульового дня” у розподілених комп’ютерних системах

Мельничук Володимир Олегович, КІм-24-2

Наукова новизна результатів

Комплексний підхід до приватної верифікації

Розроблено метод приватної верифікації відповідності компонента політиці безпеки в розподіленому середовищі.

Інтеграція компонентів

Поєднано формалізоване подання політики безпеки, метод формування твердження про стан компонента, метод генерації та перевірки доказу "нульового дня", та механізм інтеграції в контур контролю доступу.

Відмінність від відомих підходів

Запропоноване рішення дозволяє приймати обґрунтоване рішення щодо доступу без безпосереднього розкриття внутрішніх атрибутів компонента.

Основні результати

Метод формування твердження

Правила відбору та нормалізації атрибутів для створення компактного, непрямого твердження про стан компонента.

Генерація та перевірка доказу

Протокол доказу “нульового дня” — підтвердження істинності твердження без розкриття внутрішніх атрибутів.

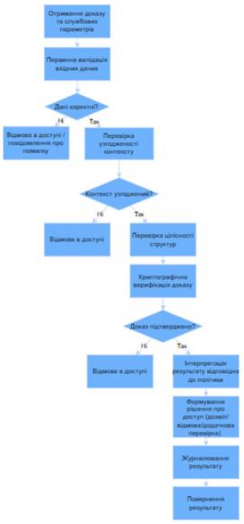
Інтеграція в контроль доступу

Механізм інтеграції результату в політику доступу для прийняття автоматичного рішення.

Алгоритми, розроблені в роботі

Алгоритм формування доказу

Алгоритм перевірки доказу і рішення



Архітектура програмного засобу

Модуль керування політиками

Зберігання та інтерпретація політик доступу, версійність та оновлення.

Модуль підготовки атрибутів

Агрегація, нормалізація і фільтрація атрибутів компонента.

Модулі формування твердження та генерації доказу

Логіка створення компактних тверджень і криптографічних доказів.

Модуль перевірки доказу та прийняття рішення

Криптографічна верифікація та інтеграція з механізмами контролю доступу.

Журналювання

Незмінні записи результатів перевірок, подій та аудиту.

Структура проєкту

```
project/
├── app/
│   ├── main.py
│   ├── api/
│   │   └── routes.py
│   ├── core/
│   │   ├── config.py
│   │   └── security.py
│   ├── models/
│   │   ├── request_models.py
│   │   └── response_models.py
│   ├── services/
│   │   ├── policy_service.py
│   │   ├── statement_service.py
│   │   ├── proof_service.py
│   │   ├── verify_service.py
│   │   └── decision_service.py
│   ├── storage/
│   │   ├── policy_store.py
│   │   └── log_store.py
│   └── utils/
│       └── serializer.py
├── policies/
├── logs/
├── requirements.txt
└── docker-compose.yml
```

Реалізація та технології

Модульний серверний застосунок реалізовано на Python з FastAPI; використано бібліотеки для криптографії та логування.

Фрагмент маршруту /verify

```python
from fastapi import APIRouter
from app.models.request_models import VerificationRequest
from app.models.response_models import VerificationResponse
from app.services.decision_service import process_verification

router = APIRouter()

@router.post("/verify", response_model=VerificationResponse)
def verify_component(payload: VerificationRequest):
    return process_verification(payload)
```

Висновки

Побудовано модель приватної верифікації

Формалізовано ролі, атрибути та обмеження розкриття інформації.

Розроблено метод формування твердження

Правила агрегації та нормалізації атрибутів для мінімального розкриття.

Розроблено методи генерації та перевірки доказу

Протокол “нульового дня” дозволяє верифікувати без розкриття внутрішніх даних.

Реалізовано програмний засіб та підтверджено працездатність

Модульний серверний застосунок пройшов тестування у реалістичних сценаріях.

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Володимир МЕЛЬНИЧУК

Співавтор:

Назва: Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

Експерт: Світлана САЧЕНКО

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.74%

Коефіцієнт подібності 2:0.29%

Мікропробіли: 4

Заміна букв: 0

Інтервали: 0

Білі знаки: 6

Дата створення звіту: 2026-04-25 08:45:27.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2026-04-25 Доцент Андрій Нічепорук

Дата експерт

Anti-Plagiarism (http://ap.km.ua) v-15.701

Максимальне співпадіння з одним документом 21.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 270687 Назва: МКР Засіб захисту даних та протокол доказу “нульового дня” у розподілених комп’ютерних системах Додано в БД: 2026-04-25 Автора: Володимир МЕЛЬНИЧУК Керівники: Світлана САЧЕНКО Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	169545	1149	36880 (22%)	293 (26%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
269754	Назва: Звіт з ПДП Засіб захисту даних та протокол доказу “нульового дня” у розподілених комп’ютерних системах Додано в БД: 2026-03-11 Автора: В.О.Мельничука Керівники: Нічепорука А.О Консультанти: Опоненти:	35602 (21.0%)	286 (25.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Здобувач: Володимир МЕЛЬНИЧУК

Тема: Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи магістра:

Кількість листів креслень – ; кількість сторінок записки 90

1. Короткий зміст роботи та прийнятих рішень У роботі запропоновано засіб захисту даних у розподілених комп'ютерних системах із використанням протоколу доказу "нульового дня".

2. Висновок про відповідність роботи дипломному завданню Кваліфікаційна робота магістра відповідає виданому завданню

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі проведено аналіз предметної області захисту даних у розподілених комп'ютерних системах, розглянуто особливості їх функціонування, основні загрози безпеці даних та проблему надмірного розкриття внутрішніх атрибутів під час перевірки доступу і відповідності політикам безпеки. У другому розділі сформовано концепцію побудови засобу захисту даних, побудовано модель приватної верифікації відповідності компонента політиці безпеки, а також розроблено методи формування твердження, генерації та перевірки доказу "нульового дня" й інтеграції результатів перевірки в механізми контролю доступу. У третьому розділі розроблено алгоритми формування доказу відповідності політиці безпеки та перевірки доказу з подальшим прийняттям рішення про доступ. У четвертому розділі реалізовано програмну частину засобу захисту даних, виконано побудову модулів формування твердження, генерації та перевірки доказу, налаштовано підсистеми керування політиками, конфігурацією та журналюванням подій

4. Позитивні сторони роботи: Запропонований засіб захисту даних та протокол доказу "нульового дня" для розподілених комп'ютерних систем дозволяють забезпечити приватну верифікацію відповідності компонента політиці безпеки без надмірного розкриття його внутрішніх атрибутів, підвищити рівень конфіденційності під час перевірки доступу, формалізувати процес прийняття рішень у контурі контролю доступу, а також забезпечити узгоджену взаємодію між модулями формування твердження, генерації доказу, перевірки результату та журналювання подій.

5. Негативні сторони роботи: У роботі трапляються окремі неточності в описі алгоритмічної та програмної реалізації засобу захисту даних, які потребують додаткового уточнення для більшої логічної цілісності викладеного матеріалу.

6. Оцінка графічного оформлення та пояснювальної записки роботи: –

7. Відгук про роботу в цілому: В загальному робота виконана на невисокому рівні.

8. Інші зауваження: –

9. Оцінка кваліфікаційної роботи магістра:

Розглянувши позитивні та негативні сторони представленої кваліфікаційної роботи магістра вважаю, що робота заслуговує оцінки «задовільно» 60.00 (E)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Мартинюк Валерій Володимирович, к.т.н., проф., професор кафедри АКІТпР

"1 травня" 2026р.

Зав. кафедри КІІС
д-р. філософії Ользі ПАВЛОВІЙ

Володимир МЕЛЬНИЧУК

ПІБ здобувача вищої освіти

ФІТ, 2 курсу, групи КІ2м-24-2

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів академічної відповідальності, ознайомлений (а). Про використання спеціалізованих програмних засобів (СПЗ) StrikePlagiarism та Anti-Plagiarism для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений (а). Надаю університету право на передачу моєї роботи для обробки та збереження в базах даних СПЗ і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються СПЗ.

Також надаю свою згоду на обробку й збереження університетом моєї роботи в Інституційному репозитарії Хмельницького національного університету.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

1 травня 2026 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Назва кваліфікаційної роботи Засіб захисту даних та протокол доказу "нульового дня" у розподілених комп'ютерних системах
Автор Володимир МЕЛЬНИЧУК
Освітня програма Інформаційні системи та технології
Рівень вищої освіти другий (магістерський)
Спеціальність 123 Комп'ютерна інженерія
Науковий керівник: к.е.н., доцент Світлана САЧЕНКО

На основі аналізу кваліфікаційної роботи на дотримання вимог академічної доброчесності (у т.ч. відсутності ознак академічного плагіату) з урахуванням результатів перевірки роботи спеціалізованим програмним засобом(ами) комісія зробила такий висновок:

№	Висновок	Позначка про відповідність
1	Ознаки академічного плагіату	
1.1	Запозичення, виявлені в роботі, є законними і не є академічним плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних, якщо потрібно). Робота приймається до захисту.	відповідає
1.2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована.	
1.3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота може бути допущена до захисту після того як буде відкоригована та доопрацьована і успішно пройде повторну перевірку на академічний плагіат.	
1.4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
2	Інші види порушень академічної доброчесності	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) усі запозичення фрагментарні, або мають належним чином оформленні посилання;

2) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;

3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

4) значна частина знайденого плагіату відноситься до списку використаних джерел

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1.74% і адресується; та системою Anti-Plagiarism складає 21%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

25.04.2026

Завідувач кафедри _____ Підпис — Ольга ПАВЛОВА Ім'я, ПРІЗВИЩЕ

Гарант освітньої програми _____ Підпис — Олег САВЕНКО Ім'я, ПРІЗВИЩЕ

Керівник кваліфікаційної роботи _____ Підпис — Світлана САЧЕНКО Ім'я, ПРІЗВИЩЕ