Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОД ТА ЗАСІБ ЗАХИСТУ МЕДИЧНИХ ДАНИХ НА ОСНОВІ ГОМОМОРФНОГО ШИФРУВАННЯ»

Виконала: студентка 2 курсу групи 1БС-24 м спеціальності 125 Кібербезпека та захист інформації
_____Владислава ЛАНОВА

Керівник: к. т. н., доц., доцент каф. ЗІ
_____Юрій БАРИШЕВ
«16» грудня 2025 р.

Опонент: к. т. н., доцент каф. ПЗ
_____ Оксана РОМАНЮК
«16» грудня 2025 р.

Допущено до захисту
В. о. завідувача кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«16» грудня 2025 р.

Вінниця ВНТУ – 2025 року

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека та захист інформації
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«24» 09 2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Владиславі ЛАНОВІЙ

1. Тема роботи: «Метод та засіб захисту медичних даних на основі гомоморфного шифрування»
   керівник роботи: Юрій БАРИШЕВ, к. т. н., доцент, доцент кафедри ЗІ,
   затверджені наказом ректора ВНТУ від 24 вересня 2025 року № 313.
2. Строк подання студентом роботи 16 грудня 2025 р.
3. Вихідні дані до роботи:
   - Мови програмування: JavaScript, Solidity.
   - Підтримка браузерів: Firefox, Chrome, TOR.
   - Вид шифрування – частково гомоморфне шифрування.
   - Підтримка блокчейну – Ethereum.
4. Зміст текстової частини: Вступ. 1. Аналіз методів та засобів захисту медичних даних. 2. Метод захисту медичних даних. 3. Засіб захисту медичних даних. 4. Експериментальні дослідження та тестування. 5. Економічна частина. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: порівняльний аналіз нормативно-правових актів, математичний опис завдання, метод захисту медичних даних, архітектура засобу, алгоритм зберігання даних, алгоритм отримання даних, алгоритм оновлення даних, результати модульного тестування, результати тестування безпеки, показники економічної ефективності.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийня
1	Ю. БАРИШЕВ, к.т.н., доц., доц. каф. ЗІ	25.09.25	06.10.25
2	Ю. БАРИШЕВ, к.т.н., доц., доц. каф. ЗІ	25.09.25	03.11.25
3	Ю. БАРИШЕВ, к.т.н., доц., доц. каф. ЗІ	25.09.25	17.11.25
4	Ю. БАРИШЕВ, к.т.н., доц., доц. каф. ЗІ	25.09.25	07.12.25
5	О. ЛЕСЬКО, зав. каф. ЕПВМ, к.е.н., доц.	25.09.25	08.12.25

7. Дата видачі завдання. 24.09.25р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	24.09.2025 – 26.09.2025	
2	Аналіз інформаційних джерел за напрямком магістерської кваліфікаційної роботи	27.09.2025 – 07.10.2025	
3	Науково-технічне обгрунтування	11.10.2025 – 22.10.2025	
4	Аналіз методів та засобів захисту медичних даних	23.10.2025 – 26.10.2025	
5	Виконання математичного опису завдання	27.10.2025 – 02.11.2025	
6	Розробка та формування вимог до методу	03.11.2025 – 10.11.2025	
7	Застосування методу захисту медичних даних на основі гомоморфного шифрування	10.11.2025 – 17.11.2025	
8	Розробка розділу економічного обгрунтування доцільності розробки	18.11.2025 – 22.11.2025	
9	Оформлення пояснювальної записки	23.11.2025 – 29.11.2025	
10	Попередній захист та доопрацювання МКР	29.11.2025 – 11.12.2025	
11	Перевірка на наявність текстових запозичень	12.12.2025 – 15.12.2025	
12	Представлення МКР до захисту, рецензування	16.12.2025 – 19.12.2025	
13	Захист МКР	19.12.2025 – 23.12.2025	

Студентка _____ Владислава ЛАНОВА

Керівник роботи _____ Юрій БАРИШЕ

АНОТАЦІЯ

УДК 004.056

Ланова В. Метод та засіб захисту медичних даних на основі гомоморфного шифрування. Магістерська кваліфікаційна робота зі спеціальності 125 – Кібербезпека та захист інформації, освітня програма – Безпека інформаційних і комунікаційних систем. Вінниця: ВНТУ, 2025. 95 с.

Укр. мовою. Бібліогр.: 56 назв; рис. 27; табл.: 12.

Магістерська кваліфікаційна робота присвячена розробці методу та засобу захисту медичних даних на основі гомоморфного шифрування. Здійснено аналіз нормативно-правової бази, яка регулює вимоги щодо кібербезпеки медичних даних. Проаналізовано відомі інформаційні системи, які використовуються в медицині з основним акцентом на методи та засоби захисту даних в цих системах. Здійснено математичний опис процесу захищеної обробки медичних даних, запропоновано метод захисту медичних даних на основі гомоморфного шифрування, який дозволяє виконувати обчислення над зашифрованими даними без потреби у їх розшифруванні. Розроблено алгоритми та програмний засіб, які реалізують запропонований метод. Проведено тестування засобу, яке включало статичне та динамічне тестування безпеки. Наведено результати експериментальних досліджень. Оцінено витрати та визначено економічну доцільність реалізації.

Ілюстративна частина складається з 10 плакатів з демонстрацією результатів проведеного тестування.

Ключові слова: гомоморфне шифрування, кібербезпека, захист даних, медичні дані, блокчейн, смарт-контракти, медицина, критична інфраструктура.

ABSTRACT

UDC 004.056

Lanova V. Method and software tool for protecting medical data based on homomorphic encryption. Master's qualification work in specialty 125 – Cybersecurity and Information Protection, educational program Information and Communication Systems Security. Vinnytsia: VNTU, 2025. 95 p.

In Ukrainian language. Bibliographer: 56 titles; 27 figures; 12 tables.

The master's qualification work is devoted to the development of a method and a software tool for protecting medical data based on homomorphic encryption. An analysis of the regulatory and legal framework governing cybersecurity requirements for medical data was conducted. Existing information systems used in healthcare were analyzed, with a primary focus on the methods and tools for data protection employed in these systems. A mathematical description of the process of secure medical data processing was developed, and a medical data protection method based on homomorphic encryption was proposed, enabling computations to be performed on encrypted data without the need for decryption. Algorithms and a software tool implementing the proposed method were developed. The tool was tested using both static and dynamic security testing. The results of experimental studies are presented. Costs were evaluated, and the economic feasibility of implementation was determined.

The illustrative part consists of 10 posters demonstrating the results of the testing.

Keywords: homomorphic encryption, cybersecurity, data protection, medical data, blockchain, smart contracts, healthcare, critical infrastructure.

ЗМІСТ

ВСТУП

Потреба в захисті медичних даних, зокрема персональних даних пацієнтів, є актуальною у всьому світі. Так в Україні діє Закон України «Про захист персональних даних» [1]. Подібні вимоги висувають міжнародні нормативні акти, такі як GDPR [2] у ЄС та HIPAA [3] у США. Крім того, GDPR висуває фіскальні стягнення за недотримання їх вимог.

Під час війни, коли лікарні та медичні установи зруйновані, а багато пацієнтів залишаються без доступу до історії своїх хвороб, питання збереження персональних медичних даних набуває надзвичайної важливості. Відсутність історії хвороб та медичних даних може призвести до помилок у діагностиці та лікуванні, що може мати фатальні наслідки для пацієнтів. Крім того, під час війни, коли доступ до медичних установ часто обмежений, а пацієнти переміщуються в інші регіони, питання захисту медичних даних набуває ще більшої актуальності. Впровадження систем шифрування, які дозволяють зберігати та обробляти дані без їх розкриття, дає змогу створити механізм, при якому лікарі можуть отримувати доступ до медичних даних пацієнтів, не порушуючи їх конфіденційність. Це допомагає зберегти історію хвороб і здійснити належне лікування, навіть у таких складних умовах.

Для вирішення цих проблем необхідно впроваджувати захист даних, зокрема через шифрування персональної інформації пацієнтів. У звичайного шифрування є суттєвий недолік — дані стають доступними людині, яка їх оновлює, крім того, після розшифрування вони стають доступними шпигунським програмам, зокрема, вразливими до атак. На противагу цього, гомоморфне шифрування має перевагу — взаємодія та виконання операцій над зашифрованими даними, без потреби в їх розшифруванні.

Відомі методи зберігання даних, такі як бази даних, хмарні сховища та блокчейн, мають свої недоліки: бази даних не гарантують повної захищеності, хмари вимагають безпечного з’єднання та довіри до провайдера, а блокчейн не

підходить для зберігання великих масивів даних і потребує додаткових заходів для захисту конфіденційності.

Тому постає **актуальна задача** розробки методу захищеного зберігання конфіденційних даних.

Об'єкт дослідження – процес зберігання конфіденційних даних.

Предмет – методи та засоби захисту медичних даних онкохворих пацієнтів.

Метою є покращення захисту медичних даних, шляхом використання гомоморфного шифрування та технологій розподіленого зберігання даних.

Для досягнення мети необхідно виконати такі **завдання**:

- проаналізувати методи та засоби захисту медичних даних;
- виконати математичний опис завдання;
- узагальнено представити метод захисту медичних даних;
- адаптувати метод до предметної області;
- розробити архітектуру засобу;
- розробити алгоритми, що реалізують метод;
- реалізувати засіб захисту медичних даних на основі методу;
- протестувати коректність та безпеку програмного засобу;
- визначити показники економічної ефективності розробки.

Наукова новизна роботи полягає в тому, що удосконалено метод захисту медичних даних, який на відміну від відомих передбачає одночасне застосування частково гомоморфного шифрування та децентралізованих сховищ даних, що дозволяє покращити рівень захисту конфіденційності та доступності даних.

Практична цінність цієї роботи:

- засіб захисту медичних даних на основі гомоморфного шифрування,
- експериментальне дослідження схем гомоморфного шифрування.

Публікації результатів магістерської кваліфікаційної роботи.

За результатами магістерської кваліфікаційної роботи отримано публікацію, що індексується у наукометричній базі Scopus в Proceedings of the 7th International Conference on Informatics & Data-Driven Medicine [4].

Опубліковано 2 статті у фахових виданнях категорії Б:

- Інформаційні технології та комп`ютерна інженерія, 2025 [5];
- Наукові праці ВНТУ, 2023 [6].

Отримано патент на корисну модель на спосіб захисту інформації в мережах передавання даних [7].

Результати магістерської кваліфікаційної роботи доповідалися на 10 конференціях з публікацією матеріалів та тез доповідей:

- Міжнародна конференція SMICS-2025 «Безпека сучасних інформаційно-комунікаційних систем», 2025 [8];
- Міжнародна науково–практична конференція «Інформаційні технології та комп’ютерне моделювання», 2025 [9];
- Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп`ютерної інженерії, 2025 [10];
- Всеукраїнська науково-практична конференція «Theoretical and Applied Cybersecurity» (TACS-2024) [11];
- International Scientific Conference «ITSec», 2024 [12];
- Всеукраїнська науково-практична конференція з міжнародною участю. «Медико-технічна співпраця заради перемоги: актуальні завдання медичної, біологічної фізики та інформатики», 2024 [13];
- Міжнародна науково–практична конференція «Інформаційні технології та комп’ютерне моделювання», 2024[14];
- Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (ВНТКП ВНТУ), 2023 [15];
- Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, 2023 [16];
- Міжнародна науково–практична конференція «Інформаційні технології та комп’ютерне моделювання», 2022 [17].

1 АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ЗАХИСТУ МЕДИЧНИХ ДАНИХ

1.1 Аналіз нормативно-правової бази

Законодавство України про охорону здоров'я базується на Конституції України та охоплює Закон України «Основи законодавства України про охорону здоров'я» [18]. В цьому законі визначено поняття медичної таємниці та дані, що її становлять.

Відповідно до статті 40, що передбачена у цьому законі, існує поняття лікарської таємниці, яка передбачає в собі те, що: «медичні працівники та інші особи, яким у зв'язку з виконанням професійних або службових обов'язків стало відомо про хворобу, медичне обстеження, огляд та їх результати, інтимну і сімейну сторони життя громадянина, не мають права розголошувати ці відомості, крім передбачених законодавчими актами випадків» [18].

Також в Україні є чинним Закон України «Про захист персональних даних» [1], що передбачає право невтручання в особисте життя у зв’язку з обробкою персональних даних.

У різних країнах світу діють власні нормативно-правові акти у сфері захисту даних в охороні здоров’я, проте всі вони мають спільну мету — забезпечення конфіденційності та безпеки пацієнтів. У США акцент робиться на безпеці медичних записів і відповідальності за їх витоки, в ЄС — на правах суб’єктів даних і контролі за використанням інформації. Велика Британія адаптувала європейські правила під свої умови, Канада і Австралія приділяють увагу обмеженню збору та використання персональних даних, а Японія модернізувала свої норми відповідно до світових стандартів [19].

На сьогоднішній день, в Україні не передбачені штрафи за недотримання законів, однак у зв'язку з тим, що медична сфера вдосконалюється, це може стати звичним явищем, як і в інших державах. У зв’язку з інтеграцією до ЄС, в Україні Закон «Про захист персональних даних» заміниться на GDPR [19].

Результати аналізу нормативно-правових актів в різних країнах та в Україні доцільно представити у вигляді таблиці (табл. 1.1).

Таблиця 1.1 – Аналіз нормативно-правових актів

Країна/Регіон	Основний закон/акт	Рік ухвалення	Основний акцент	Санкції/штрафи
США	HIPAA	1996	Захист медичної інформації (PHI), права пацієнтів, конфіденційність	До $1,5 млн на рік за категорію порушення; кримінальна відповідальність
ЄС	GDPR	2016 (чинний з 2018)	Захист особистих даних, що стосуються громадян	До €20 млн або 4% глобального обороту
Велика Британія	UK GDPR, Data Protection Act	2018	Адаптація GDPR після брекзиту, регулювання NHS	До £17,5 млн або 4% обороту
Канада	PIPEDA (федеральний), PHIPA (Онтаріо)	2000 / 2004	Захист персональних і медичних даних у приватному секторі	До $100 тис. CAD
Австралія	Privacy Act, My Health Records Act	1988 / 2012	Конфіденційність, електронні медичні записи	До AUD $2,22 млн
Японія	APPI	2003	Захист персональних і медичних даних	До ¥100 млн (≈ $900 тис.)
Китай	PIPL, Cybersecurity Law	2021 / 2017	Аналог GDPR, особливий захист медичних даних	До ¥50 млн або 5% обороту
Індія	Digital Personal Data Protection Act	2023	Захист персональних даних	До 2,5 млрд INR (≈ $30 млн)
Бразилія	LGPD	2020	Аналог GDPR	До 2% обороту, макс. BRL 50 млн
Україна	Закон України «Про захист персональних даних»	2010	Захист персональних даних громадян України	До 34 тис. грн
Україна	Закон України «Про електронну систему охорони здоров’я» (eHealth)	2018	Регулювання електронних медичних записів	Адміністративна відповідальність

З аналізу нормативно-правової бази інших країн випливає, що існує тенденція у збільшенні регулювання процесів обробки інформації в медицині, що дозволяє підтвердити актуальність розробок в цій галузі.

1.2 Аналіз відомих засобів

Гомоморфне шифрування використовується в медичній сфері для підвищення захисту конфіденційності даних, з можливістю виконання обчислень на зашифрованих наборах даних без потреби в їх розшифровуванні.

Одним із прикладів є швейцарський проєкт MedCo [20], що є частиною ініціативи Data Protection and Personalized Health. MedCo використовує гомоморфне шифрування для безпечного обміну даними між лікарнями, без розкриття первинної інформації пацієнтів.

В Естонії широке поширення технології блокчейн в медицині дозволяє громадянам відстежувати свої медичні записи в системі E-Health, забезпечуючи цілісність даних і захист від внутрішніх загроз [21]. Однак, для запису всіх даних необхідно значні обчислювальні ресурси через велику кількість транзакцій.

Модель «Medichain» працює на основі блокчейну [22]. Ця модель використовує блокчейн як базу даних для зберігання повної інформації про пацієнта в блоці. Записи транзакцій гешуються для збереження отриманих геш-значень у дереві Меркля, щоб забезпечити безпеку даних і запобігти підробці, таким чином зменшуючи помилки під час прийняття клінічних рішень. Дерево Меркля вимагає повної перебудови при додаванні або видаленні будь-якого запису. Це може бути обтяжливим і вимагати значних обчислювальних ресурсів, особливо для великих обсягів даних. До того ж, для верифікації записів в дереві Меркля потрібно проводити багато операцій гешування, що може збільшувати вимоги до обчислювальних ресурсів [6].

Автори пропонують підхід до використання блокчейну для додавання медичних даних [23]. Коли пацієнт звертається до лікарні, лікар діагностує його стан і створює медичну картку. Для забезпечення можливості пацієнту повернутись до своїх даних у будь-який час, а лікарю – отримати необхідну

інформацію, всі медичні записи повинні бути збережені. Оскільки ці записи містять персональні дані пацієнта, їх необхідно шифрувати перед збереженням. Лікар шифрує та підписує медичну документацію, потім завантажує її в систему IPFS для зберігання, а також генерує індекси для ключових слів. IPFS повертає геш-адресу збереженого файлу лікарю [14]. Після цього лікар шифрує і гешує медичну документацію та її індекс, а потім зберігає геш-значення і зашифровану геш-адресу в блокчейні [17].

Автори роботи [24] пропонують багаторівневий підпис на основі атрибутів та блокчейну для взаємодії з медичними даними. Схема MA-ABS – це спосіб підписувати повідомлення, який не розголошує особисту інформацію пацієнта, але підтверджує певні атрибути, такі як статок або стаж, які пацієнт має. Ця техніка забезпечує приватність пацієнта та гарантує непідробність підпису. Однак, при збільшенні кількості пацієнтів та атрибутів система може зустрітися з обмеженнями масштабованості, що може вплинути на швидкодію та ефективність. Також, якщо будь-який з атрибутів порушиться або буде скомпрометований, це може поставити під загрозу конфіденційність даних та непідробність підписів.

Гомоморфне шифрування також сприяє безпечному аналізу даних про серцево-судинну систему, що дозволяє виявляти осіб із високим ризиком та прогнозувати розвиток захворювання без порушення конфіденційності [10]. Водночас, використання гомоморфного шифрування має недоліки: високе обчислювальне навантаження, що може знижувати продуктивність і масштабованість, особливо при обробці великих наборів даних [8].

Інше, не менш важливе застосування гомоморфного шифрування – це спільне дослідження ракових захворювань, де зашифровані дані з декількох лікарень аналізуються для визначення найбільш ефективних протоколів хіміотерапії, залежно від типу раку та особливостей пацієнтів [25, 26]. Однак це потребує додаткових обчислювальних ресурсів, пропускної здатності та місця для зберігання зашифрованих даних, що значно збільшує операційні витрати.

Ці методи мають значне обчислювальне навантаження, особливо під час обробки великих наборів даних або медичних зображень високої роздільної здатності, що призводить до затримок і зростання операційних витрат.

1.3 Аналіз методів гомоморфного шифрування

Гомоморфне шифрування – це форма шифрування, яка дозволяє виконувати обчислення над зашифрованими даними без необхідності їхнього розшифрування. Це особливо цінно в ситуаціях, коли конфіденційна інформація потребує обробки, але повинна залишатися захищеною. У гомоморфному шифруванні зашифрований вхід дає зашифрований вихід, який після розшифрування збігається з результатом операції, виконаної над відкритими даними.

Існують такі види гомоморфного шифрування:

- частково гомоморфне шифрування;
- повністю гомоморфне шифрування.

Частково гомоморфне шифрування дозволяє виконувати лише певні операції (додавання або множення) над зашифрованими даними. Частково гомоморфні схеми, серед яких варто виділити RSA [27], алгоритм Ель-Гамаля [28] та криптосистему Пальєра [29], забезпечують виконання лише одного типу операцій — додавання або множення — у необмеженій кількості.

Їх криптостійкість базується на складних математичних проблемах, таких як факторизація великих чисел чи обчислення дискретних логарифмів. Попри високу надійність, ці системи мають недоліки, зокрема значні обчислювальні витрати, необхідність використання великих ключів і збільшення обсягу зашифрованих даних. Тому практичне застосування частково-гомоморфних алгоритмів обмежене, хоча вони й залишаються важливою основою для розвитку сучасної криптографії та використовуються у формуванні електронного цифрового підпису й окремих механізмах захисту.

Прикладом частково гомоморфного шифрування є криптосистема Пальєра. Однією з переваг криптосистеми Пальєра є її гомоморфна властивість

у поєднанні з недетермінованим шифруванням завдяки використанню випадкових чисел.

Основна схема зашифрування складається з чотирьох кроків [30].

Крок 1. Генерування публічної пари ключів (*n, g*).

Для цього потрібно згенерувати великі прості числа p та q однакової довжини.

Обчислення:

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

Далі потрібно згенерувати g таким чином, щоб $g \in Z_{n^2}^*$.

Крок 2. Приватний ключ для розшифрування — (λ, μ).

Для цього потрібно обчислити λ як:

$$\lambda = lcm(p-1, q-1), \tag{1.2}$$

де *lcm(.)* — найменше спільне кратне.

Наступним етапом обчислюється модульний обернений елемент:

$$\mu = \left(L\left(g^{\lambda} mod n^2\right)\right)^{-1} mod\ n, \tag{1.3}$$

де функція $L(x) = \frac{(x-1)}{n}$ (цілочисельна частка від ділення).

Потрібно вибрати випадкове число *r* в межах $0 < r < n \wedge gcd(r, n) = 1$.

Крок 3. Для шифрування повідомлення (m), де $0 \leq m < n$, потрібно обчислити *c*, як:

$$c = g^m \cdot r^n, \tag{1.4}$$

де *c* — шифротекст.

Крок 4. Для розшифрування *m* потрібно виконати таке обчислення:

$$m = L\left(c^{\lambda} mod n^2\right) * \mu\ mod\ n, \tag{1.5}$$

де *c* — шифротекст, який треба розшифрувати і $c \in Z_{n^2}^*$.

Серед операцій, підтримуваних схемою Пальєра, є гомоморфне додавання (1.6) та множення (1.7).

Слід зазначити, що без знання приватного ключа неможливо обчислити добуток зашифрованих повідомлень.

Коли два шифротексти множаться, результат збігається із сумою їх відкритих значень.

Множення двох шифротекстів:

$$D\left(E_{pub}(m_1) * E_{pub}(m_2)\ mod\ n^2\right) = m_1 + m_2\ mod\ n, \quad (1.6)$$

де D — добуток, який необхідно розшифрувати.

Коли шифротекст підноситься до степеня відкритого значення, результат розшифровується до добутку цих двох відкритих значень:

$$D\ mod\ n^2 = m_1 * m_2\ mod\ n \quad (1.7)$$

Криптосистема Пальєра є обчислювально менш вимогливою порівняно з повністю гомоморфними схемами шифрування, такими як BFV [31], CKKS [32] або новітніми методами. Тому її можна віддати перевагу для реалізації в дослідженнях через меншу обчислювальну складність, що забезпечує швидше та менш обтяжливе для обчислень оновлення даних.

Повністю гомоморфне шифрування розширює можливості частково гомоморфного шифрування, дозволяючи виконувати довільні операції, включаючи як додавання, так і множення у зашифрованому вигляді. Повністю гомоморфні схеми шифрування можуть виконувати будь-які обчислення над зашифрованими даними, що робить їх достатньо потужними.

Серед прикладів повних гомоморфних схем є такі схеми, як BFV, BGV [33] та CKKS.

Схема BFV базується на припущенні про складність задачі решіток у кільцевому варіанті (RLWE) і дає змогу виконувати точні арифметичні операції додавання та множення.

Алгоритм BGV також базується на задачі решіток (LWE) та RLWE, але впроваджує техніку зміни модуля, яка забезпечує контроль зростання шуму й дозволяє зберігати коректність обчислень навіть після багатьох операцій.

Схема гомоморфного шифрування CKKS призначена для виконання обчислень над зашифрованими дійсними або комплексними числами з певним рівнем наближення. На відміну від точних схем, CKKS дозволяє працювати з

числовими значеннями з плаваючою комою, що робить її особливо корисною для задач аналізу даних і машинного навчання.

На початковому етапі виконується налаштування параметрів, які визначають безпеку та точність схеми. Обирається циклотомічне поліноміальне кільце певного ступеня n і модуль q, який зазвичай є степенем числа 2.

Після цього відбувається кодування — дійсні або комплексні числа перетворюються у поліноми відкритого тексту, які належать вибраному кільцю. Для цього використовується масштабування на великий коефіцієнт, що дозволяє зберегти наближену точність під час подальших обчислень.

Далі відбувається генерація ключів. Секретний ключ утворюється як випадковий поліном із малими коефіцієнтами, взятими з невеликого діапазону цілих чисел. Публічний ключ генерується на основі секретного ключа, вибраних параметрів та певної випадковості. Саме публічний ключ використовується для шифрування повідомлень.

Під час шифрування відкритий текст у вигляді полінома перетворюється на шифротекст. У процесі додається шум, який підсилює криптографічну стійкість, але водночас робить схему наближеною — тобто після розшифрування відновлюється лише приблизне значення початкового числа.

Гомоморфні операції у CKKS дозволяють виконувати арифметичні дії без розшифрування даних.

Додавання двох зашифрованих чисел реалізується шляхом додавання відповідних шифротекстів.

Множення двох шифротекстів виконується через операцію множення поліномів, після чого застосовується процес релінеаризації, який зменшує розмір і рівень шуму результату.

Для підтримання стабільності обчислень після множення також проводиться рескейлінг — масштабування шифротексту для контролю зростання шуму та збереження потрібного рівня точності.

На етапі розшифрування використовується секретний ключ, за допомогою якого шифротекст перетворюється назад у поліном відкритого тексту, що є

наближеним відображенням початкового повідомлення. Потім відбувається декодування, під час якого поліном перетворюється у дійсне або комплексне число, відновлюючи вихідні дані з урахуванням масштабованого коефіцієнта, використаного на етапі кодування.

На відміну від інших схем повного гомоморфного шифрування, CKKS орієнтований на роботу з дійсними та комплексними числами, допускаючи наближені обчислення, що робить його особливо придатним для застосувань у сфері машинного навчання, статистичного аналізу та обробки сигналів, де абсолютна точність не є обов'язковою.

Однак, є й недоліки та обмеження, а саме наявність шуму. Під час виконання кожної гомоморфної операції рівень шуму у шифротексті поступово зростає, і після певної кількості обчислень він може перевищити допустимий поріг. У такому випадку сигнал стає занадто спотвореним, і результат розшифрування втрачає зміст або стає зовсім некоректним.

Ще одним обмеженням є наближений характер схеми. Оскільки CKKS оперує не точними, а наближеними значеннями дійсних або комплексних чисел, у процесі обчислень можуть накопичуватися похибки. Це призводить до того, що кінцевий результат після кількох операцій може дещо відрізнятися від точного значення, що потребує ретельного вибору параметрів для забезпечення необхідної точності.

Таким чином, можна зробити проміжний висновок, що частково гомоморфне шифрування є доцільним для тих розробок, де достатньо виконання лише однієї операції – додавання або множення над зашифрованими даними. Це дає змогу знизити складність обчислень у порівнянні з повністю гомоморфними схемами шифрування, оскільки такі схеми мають простішу реалізацію та менші ресурсоємні витрати.

Повністю гомоморфне шифрування є доцільним для розробок, де необхідно виконувати широкий спектр операцій над зашифрованими даними, включаючи як додавання, так і множення. Це забезпечує можливість здійснювати повноцінні обчислення без розкриття конфіденційної інформації.

На відміну від частково гомоморфних схем, повністю гомоморфні алгоритми дають змогу будувати складні моделі й алгоритми без необхідності розшифровування даних, проте в свою чергу вони мають недолік, який полягає у високих ресурсоємних витратах.

У багатьох практичних випадках, наприклад при підрахунку сумарних значень або множенні на коефіцієнт, функціоналу частково гомоморфного шифрування цілком достатньо. Попри обмеження, частково гомоморфні алгоритми забезпечують високий рівень криптостійкості й можуть застосовуватися у сферах, де потрібен лише один тип операцій.

1.4 Постановка завдання дослідження

З аналізу нормативно-правової бази інших країн випливає, що існує тенденція у збільшенні регулювання процесів обробки інформації в медицині, що дозволяє підтвердити актуальність розробок в цій галузі.

Проведений аналіз відомих рішень показав, що ці методи мають значне обчислювальне навантаження, особливо під час обробки великих наборів даних або медичних зображень високої роздільної здатності, що призводить до затримок і зростання операційних витрат.

Використання лише шифрування не забезпечує комплексний захист даним, тому доцільно розробити такі метод та засіб захисту медичних даних на основі гомоморфного шифрування, який поєднає в собі використання ще децентралізованих сховищ даних, а саме технології блокчейн, який забезпечить прозорість та цілісність, а гомоморфне шифрування забезпечить цим даним захист конфіденційності під час розрахунків, менші обчислювальні витрати та масштабованість.

1.5 Висновки з розділу

Аналіз нормативно-правової бази показав, що захист персональних даних, зокрема медичних даних пацієнтів, є критично важливою частиною дослідження.

Аналіз відомих технологій обробки медичних даних показав, що традиційні платформи, які не використовують децентралізовані сховища, у них залишаються проблеми з цілісністю, водночас використання гомоморфного шифрування має перспективи, проте у відомих реалізаціях воно вимагає використання ресурсів, тому не набуло широкого використання.

Було розглянуто такі основні методи гомоморфного шифрування, як частково та повністю гомоморфне шифрування.

Так, частково гомоморфне шифрування виконує лише одну із операцій (додавання або множення) над зашифрованими даними, проте у порівнянні з повністю гомоморфними схемами шифрування, такі схеми мають простішу реалізацію та менші ресурсоємні витрати.

В той час як повністю гомоморфне шифрування забезпечує виконання двох операцій над зашифрованими даними, проте в свою чергу вони мають недолік, який полягає у високих ресурсоємних витратах, якщо порівнювати із частково гомоморфними схемами.

Використання лише блокчейну як основного механізму зберігання забезпечує незмінність та цілісність даних, але не гарантує їх конфіденційність.

Окрім цього, більшість відомих рішень мають проблеми з продуктивністю та масштабованістю.

Особливо це актуально для критичних напрямів, зокрема лікування онкохворих, де процес захисту потребує уваги.

2 МЕТОД ЗАХИСТУ МЕДИЧНИХ ДАНИХ

2.1 Узагальнений математичний опис завдання

Для того, щоб розробити математичну модель процесу захисту медичних даних, потрібно детально проаналізувати предметну область та кожен із атрибутів.

Для формалізації процесу захисту медичних даних обрано теоретико-множинну модель, яка є простою, зрозумілою та доречною в контексті цієї предметної області.

Нехай $\boldsymbol{D}$ – це дані пацієнтів, які зберігаються в їх медичній картці. Нехай $\boldsymbol{M}$ – це множина носіїв даних, які використовуються для зберігання $\boldsymbol{D}$, а $\boldsymbol{S}$ – множина даних, вже збережених на носіях з множини $\boldsymbol{M}$. Відповідно, можливі випадки, коли через недоступність або знищення носіїв певні дані можуть бути втрачені.

Процес зберігання певної кількості даних $d \in \boldsymbol{D}$ на носії $m \in \boldsymbol{M}$ формалізується таким чином:

$$store: \boldsymbol{D} \times \boldsymbol{M} \rightarrow \boldsymbol{S} \; or \; s_m = store(d, m) \tag{2.1}$$

У формулі (2.1) індекс m позначає, що s_m зберігається на носії m. Зворотний процес отримання збережених даних визначається так:

$$retrieve: \boldsymbol{S} \times \boldsymbol{M} \rightarrow \boldsymbol{D} \; or \; d = retrieve(s_m, m) \tag{2.2}$$

Збережені дані мають оновлюватися з часом. Відповідно, передбачається процес оновлення даних:

$$update: \boldsymbol{S} \rightarrow \boldsymbol{S} \; or \; s'_m = update(s_m) \tag{2.3}$$

Таким чином, математичний опис матиме наступний вигляд:

$$MathematicalDescription = \{D, S, M, \{store(d, m), update(s_m), retrieve(s_m, m)\}\} \tag{2.4}$$

Завдання дослідження полягає у розробці такого методу захисту даних, що відповідає таким умовам:

Умова 1. $\boldsymbol{M}$ повинна бути стійкою до атак типу відмови в обслуговуванні.

Умова 2. Знищення певного носія $m \in \boldsymbol{M}$ не повинно призводити до втрат даних у S.

Ці умови повинні бути врахованими під час розробки методу.

2.2 Узагальнене представлення методу захисту медичних даних

Для забезпечення конфіденційності, цілісності та доступності медичним даним розроблено узагальнений метод захисту даних, який описує послідовність дій від моменту збору даних до їх оновлення.

Метод базується на теоретико-множинному підході та передбачає виконання процесів зберігання, отримання та оновлення даних.

Основна ідея методу полягає у тому, що дані пацієнтів перед збереженням проходять процес попереднього зашифрування, після чого розподіляються між різними носіями, що підвищує їх стійкість до несанкціонованого доступу, втрат або атак на доступність.

Узагальнене представлення методу передбачає послідовність кроків, які визначають логіку безпечної обробки медичних даних — від моменту їх отримання до фіксації змін у захищеному середовищі.

Крок 1. Збір даних пацієнта (D).

До множини D входять персональні відомості пацієнта, зокрема, № медичної картки, № паспорта, а також інші показники, які становлять персональні дані пацієнта. Всі ці дані збираються з різних джерел: медичних карток, результатів обстежень, записів лікарів та історії лікування.

Крок 2. Визначення множини носіїв даних (M), на яких будуть зберігатися медичні дані, включно з локальними серверами, хмарними сховищами, блокчейном або іншими сховищами (на вибір).

Крок 3. Обчислення унікального ідентифікатора пацієнта (наприклад, геш-значення), що дає змогу ідентифікувати конкретного пацієнта в системі. Цей ідентифікатор згодом використовується як ключ для зберігання та пошуку даних у сховищах, забезпечуючи унікальність медичної інформації.

Крок 4. Виконати зашифрування даних D. Такий підхід забезпечує, що навіть у випадку несанкціонованого доступу до носіїв, зловмисник не зможе отримати інформацію у відкритому вигляді.

Крок 5. Виконати процес збереження $store(d, m)$для кожного елемента $d \in D$, тобто записати зашифровані дані на обрані носії $m \in M$.

Крок 6. У разі потреби отримання даних, здійснити процес $retrieve(s_m, m)$, який дозволяє користувачеві отримати доступ до відповідного елемента даних D або його відновленого представлення навіть при частковій недоступності носіїв.

Крок 7. Виконання операції $update(s_m)$, що забезпечує оновлення відомостей у зашифрованому вигляді без розкриття вихідних даних D, без прив'язки до процесів $store()$ та $retrieve()$. Такий підхід забезпечує конфіденційність даних навіть під час їх модифікації та виключає необхідність розкриття чутливої інформації при кожному оновленні.

Розроблений метод має узагальнений характер і може бути застосований для різних типів медичних даних та клінічних ситуацій. Однак для практичного застосування необхідно конкретизувати процеси зберігання, обробки та оновлення даних відповідно до специфіки конкретної предметної області.

У наступному підрозділі метод буде адаптовано та більш детально представлено для конкретної предметної області.

2.3 Метод захисту медичних даних на прикладі лікування саркоми Юінга

Оскільки математичний опис можливий для різних медичних даних. Тому для доведення концепції необхідно взяти конкретний процес. Тому нехай цим процесом буде обчислення дозування хіміотерапевтичних препаратів для пацієнтів, що мають діагноз саркома Юінга [4].

Для розрахунків дозування хіміотерапії основним фактором є площа поверхні тіла (BSA) [34]. Дозування на основі BSA використовується для

обчислення призначеної дози ліків, що забезпечує баланс між ефективністю лікування раку та токсичністю препарату.

Основне призначення BSA – визначення дози хіміотерапії для пацієнта.

Формула для розрахунку BSA:

$$BSA = \sqrt{\frac{height * weight}{3600}} \tag{2.5}$$

Для розрахунку кінцевої дози препарату буде використано формулу Мостеллера [35], яка має наступний вигляд:

$$Dose = BSA * ct, \tag{2.6}$$

де *ct* – це кількість препарату на одиницю BSA,

height – зріст пацієнта,

weight – вага пацієнта.

Згідно з методологією обчислення дозування (див. формулу 2.2), множина ***D*** є множиною векторів, де кожен вектор асоційований з конкретним пацієнтом і константою, пов'язаною з методикою хіміотерапії. Тому ***D*** визначається так:

$$\boldsymbol{D} = \{\{height, weight, chemotherapySessions\}, ct\}, \tag{2.7}$$

де *height* – зріст пацієнта,

weight – вага пацієнта,

chemotherapySessions – кількість сеансів хіміотерапії,

ct – кількість препарату на одиницю BSA.

Щоб виконати умови 1 і 2, визначені в попередньому підрозділі, було обрано блокчейн типу Ethereum [36] як середовище для зберігання даних. Використання кількох вузлів блокчейну, кожен з яких може бути використаний для доступу до збережених даних, забезпечує стійкість до атак відмови в обслуговуванні, тим самим виконуючи умову 1. Це дозволяє задовольнити умову 2, оскільки знищення одного вузла не призводить до втрати даних. Крім того, втрата даних при використанні блокчейну як середовища можлива тільки в разі знищення всіх вузлів, чого можна уникнути організаційно, наприклад, запустивши кілька вузлів поза зоною бойових дій (за кордоном або в

партнерських країнах) або знизивши ризик, розміщуючи вузли в захищених зонах.

Вибір цього типу блокчейну обґрунтований масштабованістю блокчейнів цього типу завдяки смарт-контрактам як структурі даних [16]. Отже, у разі змін методики розрахунку дозування хіміотерапії або застосування запропонованого методу, використання смарт-контрактів дозволяє адаптувати структуру даних цього середовища.

Таким чином, множина середовищ визначається як:

$$\boldsymbol{M = BlockchainNodes}, \tag{2.8}$$

де *BlockchainNodes* – це множина всіх вузлів блокчейну.

Зважаючи на відкритість даних, обумовлену вибраним середовищем, і необхідність виконання умови 3, збережені дані $\boldsymbol{S}$ повинні бути представлені у зашифрованому вигляді. Тому $\boldsymbol{S}$ визначається таким чином:

$$\boldsymbol{S} = \{\{height^e, weight^e, chemotherapySessions^e\}, ct^e\}. \tag{2.9}$$

де $height^e, weight^e, chemotherapySessions^e, ct^e$ – параметри, подані у зашифрованому вигляді.

Після визначення всіх даних, необхідно реалізувати процеси зберігання, отримання та оновлення даних. Запропоновано реалізувати процес зберігання даних таким чином:

– Зібрати ідентифікаційні дані пацієнта (наприклад, номер медичної картки) та відповідні параметри, такі як зріст та вага.
– Обчислити геш-значення ідентифікаційного номера пацієнта для захисту від витоку персональних даних. У випадку використання блокчейну типу Ethereum доцільно використовувати геш-функцію Keccak-256:

$$path = keccak(patientID) \tag{2.10}$$

– Використовуючи гомоморфне шифрування, зашифрувати параметри зросту та ваги за публічним ключем пацієнта, щоб отримати $height^e$ та $weight^e$ відповідно.

– Отримати або створити профіль пацієнта у смарт-контракті, що працює на блокчейні, використовуючи відображення виду $path \rightarrow \{height^e, weight^e, chemotherapySessions^e\}$.
– Зберегти отримані значення $height^e$, $weight^e$.

Збережені зашифровані параметри можна зчитувати безпосередньо з блокчейну у разі правильного обчислення параметра *path*. Тому процес отримання *retrieve ()* можна реалізувати таким чином:

– Зібрати ідентифікаційні дані пацієнта (*patientID*).
– Отримати значення *path* аналогічно до попереднього пункту.
– Викликати метод смарт-контракту, який обчислює в зашифрованому вигляді значення відповідної дози хіміотерапії, використовуючи гомоморфні перетворення, і отримати $dose^e$.
– Використовуючи приватний ключ пацієнта, розшифрувати $dose^e$ і отримати правильне значення дози препарату.

Після проведення хіміотерапії, значення $chemotherapySessions^e$ має бути оновлено. Для цього запропоновано реалізувати процес оновлення такими кроками:

– Зібрати ідентифікаційні дані пацієнта (*patientID*).
– Отримати значення *path*.
– Зашифрувати значення 1 за публічним ключем пацієнта:
– Отримати збережене значення $chemotherapySessions^e$.
– Використовуючи гомоморфне перетворення, додати зашифроване значення 1 до значення $chemotherapySessions^e$.
– Зберегти нове значення $chemotherapySessions^e$.

Цей метод доцільно представити у вигляді кроків:

Крок 1. Збір ідентифікаційних даних пацієнта, а саме – номер медичної картки та номер паспорта.

Крок 2. Збір параметрів пацієнта – зріст, вага, кількість проведених сесій хіміотерапії.

Крок 3. Обчислення геш-значення ідентифікатора пацієнта для формування шляху доступу (path) до його даних у блокчейні за допомогою функції Keccak-256.

Крок 4. Гомоморфне зашифрування параметрів зросту, ваги та кількості сесій хіміотерапії за публічним ключем.

Крок 5. Виклик методу смарт-контракту для створення або отримання профілю пацієнта в блокчейні за його унікальним ідентифікатором.

Крок 6. Зберігання зашифрованих даних в блокчейні.

Крок 7. Виклик методу смарт-контракту для отримання зашифрованих значень параметрів.

Крок 8. Використання гомоморфних перетворень для обчислення дози хіміотерапевтичного препарату.

Крок 9. Розшифрування значення дози за приватним ключем.

Крок 10. Для оновлення кількості сесій хіміотерапії зашифрувати значення «1» за публічним ключем та отримати попередньо збережене значення кількості сесій.

Крок 11. Виконання гомоморних перетворень для отримання оновленого значення кількості сесій.

Крок 12. Зберігання оновленого значення кількості сесій в блокчейні.

Після того, як було представлено метод захисту медичних даних, можна перейти до висновків з цього розділу.

2.4 Теоретичне оцінювання стійкості до атак

Метою цього експерименту є теоретична оцінка прямої грошової вартості атаки типу «заповнення блоків спамовими транзакціями» (gas-spam) в мережі Ethereum. Виконано аналітичне моделювання витрат за одиницю часу (година, доба) на основі параметрів мережі (ліміт gas на блок, середній інтервал блоку), ринкової ціни ETH та припущених рівнів завищеної ставки gas (в Gwei).

Отримані результати використовуються для інтерпретації економічної ефективності такої атаки та її порівняння з відомими спам-атаками на Bitcoin 2017 року.

Вихідні дані для проведення оцінювання (табл. 2.1).

Таблиця 2.1 – Вихідні дані для проведення оцінювання

Параметр	Значення
Ліміт газу на блок	45 000 000
Інтервал одного блока (с)	12
К-ть блоків за годину	300
Курс ETH ($)	3 296

Ціна газу, що ставиться зловмисником — розглядаються сценарії:

$$p \in \{50,100,150,200,300\}\ Gwei,$$

де:

$1\ Gwei = 1{\times}10^{-9}\ ETH$

Модель атаки: атакуючий прагне повністю заповнити кожен блок транзакціями власного походження, тому загальний спожитий gas за блок рівний G_{block}.

Вартість одного блоку в ETH:

$$C_{ETH}^{block}(p) = G_{block} \times p, \tag{2.11}$$

де p— ціна gas у Gwei.

Вартість одного блоку в USD:

$$C_{USD}^{block}(p) = C_{ETH}^{block}(p) \times P_{ETH}. \tag{2.12}$$

Вартість за годину:

$$C_{USD}^{hour}(p) = C_{USD}^{block}(p) \times N_h. \tag{2.13}$$

Вартість за добу:

$$C_{USD}^{day}(p) = C_{USD}^{hour}(p) \times 24. \tag{2.14}$$

Результати обчислень доцільно навести в таблиці 2.2.

Таблиця 2.2 – Результати оцінювання вартості атаки

Ціна газу p, Gwei	Вартість блока, USD	Вартість години, USD	Вартість доби, USD
50	7 416	2 224 800	53 395 200
100	14 832	4 449 600	106 790 400
150	22 248	6 674 400	160 185 600
200	29 664	8 899 200	213 580 800
300	44 496	13 348 800	320 371 200

Розрахунки показали, що навіть за найнижчого зі змодельованих тарифів — 50 Gwei — один повністю заповнений блок коштуватиме близько $7416. Це означає, що витрати за одну годину нападника становитимуть приблизно $2.22 млн, а за добу — понад $53 млн. Із підвищенням ставки gas витрати зростають лінійно: при 100 Gwei — це вже майже $4.45 млн на годину, при 200 Gwei — близько $8.9 млн на годину, а при 300 Gwei — понад $13 млн щогодини. Таким чином, добовий бюджет зловмисника при тарифі 300 Gwei перевищив би $320 млн.

Ці величини свідчать, що повноцінна газ-спам атака у 2025 році коштує порядків мільйонів доларів на годину, навіть за відносно «помірних» ставок gas. Такий рівень витрат створює значний економічний бар’єр для довготривалої реалізації атаки. Зловмиснику має бути зрозуміло, що подібні дії не окупляться, якщо він не переслідує надзвичайно вагомих політичних, економічних або демонстративних цілей.

Таким чином, проведене аналітичне оцінювання демонструє, що газ-спам атака у мережі Ethereum станом на 2025 рік є надзвичайно дорогою, економічно неефективною та технічно складною.

Після отриманих результатів можна перейти до формування висновків з 2 розділу.

2.5 Висновки з розділу

У цьому розділі було виконано математичний опис процесу захисту медичних даних, що включає формалізацію операцій зберігання, отримання та оновлення інформації. Було розроблено метод та визначено основні вимоги до нього, зокрема стійкість до атак відмови в обслуговуванні, відсутність втрат даних у разі знищення окремих носіїв, контрольований доступ до інформації та можливість оновлення даних без їх розшифрування.

Для реалізації цих вимог було обрано блокчейн типу Ethereum як середовище зберігання, що забезпечує стійкий захист даних та децентралізований підхід. Використання смарт-контрактів дозволяє адаптувати структуру даних під зміни методики розрахунку дозування хіміотерапії. Щоб забезпечити конфіденційність даних, запропоновано використовувати схему частково гомоморфного шифрування для збереження параметрів пацієнтів у зашифрованому вигляді. Також було проведено теоретичну оцінку стійкості, а саме оцінка вартості спамових транзакцій, за умови атаки.

Таким чином, було сформульовано метод захисту даних, який забезпечує безпечне зберігання та обробку інформації, що є важливим етапом для правильного дозування хіміотерапевтичних препаратів.

Наступним етапом є розробка засобу, який реалізує запропонований метод.

3 ЗАСІБ ЗАХИСТУ МЕДИЧНИХ ДАНИХ

3.1 Архітектура засобу

Узагальнена архітектура засобу є клієнт-серверною, оскільки відбувається зв’язок із сервером.

Відбувається розділення на серверну та клієнтську частину з метою поліпшення роботи окремих модулів програмного засобу, забезпечуючи їм незалежне один від одного проєктування.

Побудова такої архітектури сприяє кращому розумінню роботи програмного засобу, його структури та модулів. Кожен із блоків має свою окрему функцію, забезпечуючи кращий рівень безпеки, масштабованість та стійкість до відмов. Щодо останнього параметру, то стійкість до відмов забезпечить неперервну роботу засобу та стійкість до негараздів, які можуть спричинити відмову роботи програмного засобу [7].

Отже, засіб захищеного зберігання даних складається з 2 модулів – сервера та клієнта. Клієнт, в свою чергу взаємодіє із сервером.

Клієнтська частина відповідає за взаємодію користувача із системою. Через інтерфейс користувача (UI) виконується процес автентифікації користувача в системі.

Серверна частина виконує основні криптографічні та обчислювальні операції. На першому етапі відбувається введення даних, а саме номеру медичної картки та номеру паспорта для створення унікального ідентифікатора пацієнта в системі, а також параметри для розрахунку дози препарату і відповідно попередня кількість сесій хіміотерапії. Після чого виконується гомоморфне зашифрування даних. Наступним етапом є гомоморфні перетворення, під час яких сервер, а саме смарт-контракт та його методи, виконують необхідні математичні обчислення без розшифрування самих даних. Завдяки цьому зберігається конфіденційність — сервер не має доступу до вихідних значень.

Важливим компонентом системи є блокчейн, який використовується як середовище для зберігання зашифрованих даних і результатів обчислень. Його

застосування забезпечує децентралізоване зберігання, захист від змін і видалення, а також підвищує стійкість системи. У випадку збоїв або знищення окремих вузлів, дані залишаються доступними через інші копії в мережі. Також усувається єдина точка відмови, у разі кібератаки на систему в умовах воєнного стану.

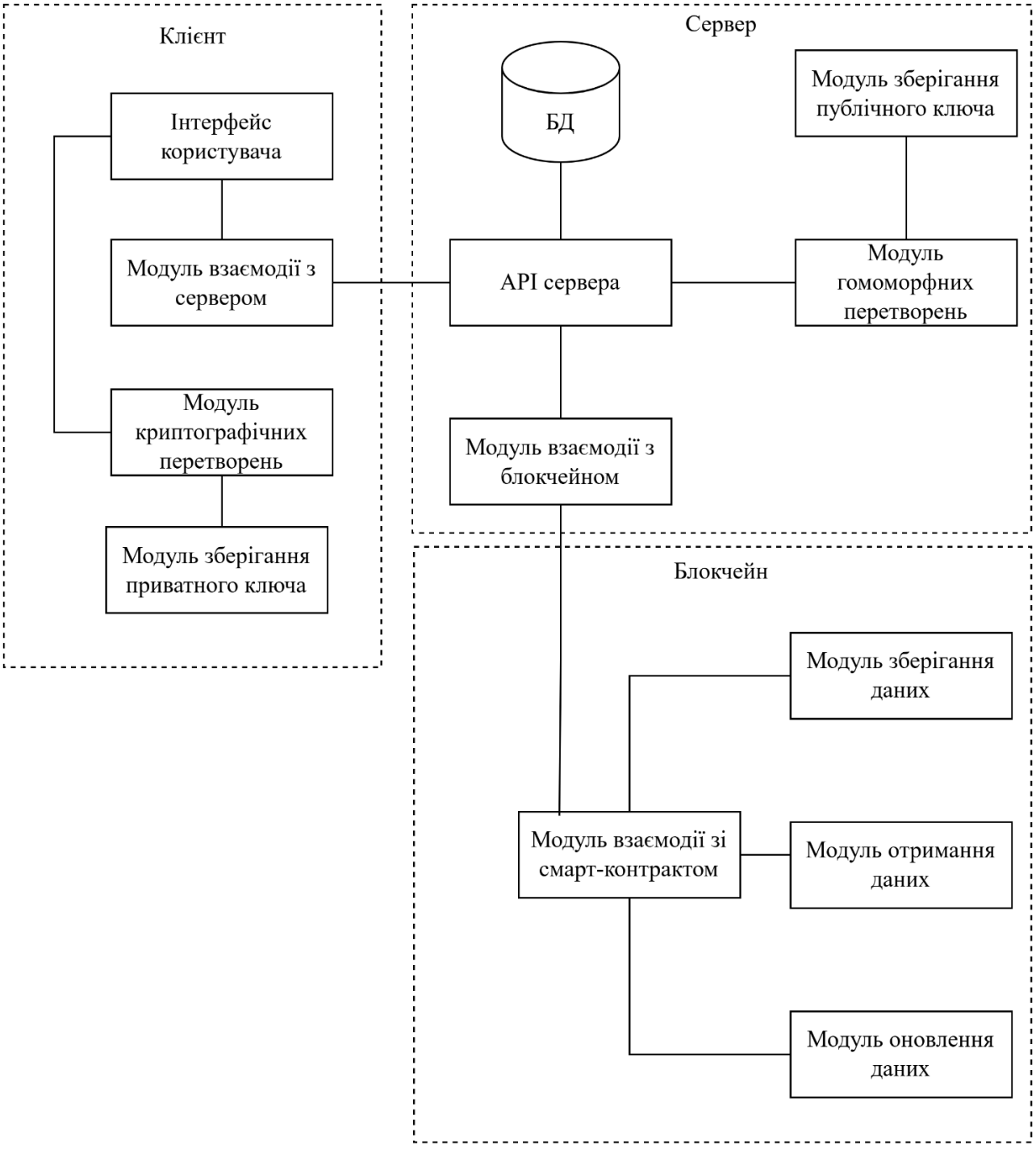


Рисунок 3.1 – Вигляд узагальненої архітектури програмного засобу

Оскільки в основному взаємодія відбувається поміж блокчейном (сервером) та клієнтською частиною, тому доцільно представити протокол взаємодії архітектурних складових, який побудований на основі архітектури (рис. 3.2).

Клієнт є початковою точкою взаємодії користувача з системою. Він відправляє запити на сервер для отримання необхідної інформації або для виконання певних дій. У цьому випадку клієнт ініціює запит на сервер, що може бути, наприклад, запитом на дані про транзакції або медичні дані.

Клієнтська частина представлена у вигляді вебсторінки, де користувачі, а саме в цьому випадку лікарі, можуть взаємодіяти із системою. Користувачі можуть виконувати необхідні запити для обробки медичних даних, такі як зберігання, отримання або оновлення інформації про пацієнтів.

Після введення запиту через візуальний інтерфейс, клієнтська частина ініціює гомоморфні операції за допомогою виклику методів смарт-контрактів з наданими даними, що забезпечують конфіденційність інформації. Наприклад, коли лікар хоче оновити кількість сеансів хіміотерапії, виконується гомоморфне зашифрування числа "1" і додавання його до вже зашифрованого значення кількості сеансів. Це дозволяє змінювати дані без їх розшифровування, зберігаючи конфіденційність медичних відомостей.

На сервері зберігається блокчейн, який обробляє запити. Сторона блокчейну перевіряє чи користувач має права доступу до необхідних даних, що забезпечує високий рівень безпеки та конфіденційності. Після верифікації запитів, блокчейн виконує операції з даними (зберігання, отримання, оновлення), надаючи результат виключно у зашифрованому вигляді, що забезпечує захист від несанкціонованого доступу.

Після обробки запиту блокчейн передає результат на сервер, який перевіряє відповіді та формує відповідь для клієнта. Результат може бути різним: це може бути підтвердження успішного виконання запиту або необхідна інформація, що була запитана користувачем (наприклад, оновлені медичні показники або нові дані про пацієнта).

Після цього сервер відправляє сформовану відповідь клієнту через веб-інтерфейс, де користувач може переглядати результати виконання запиту або продовжувати взаємодіяти з системою, залежно від наданої інформації. Вся комунікація між клієнтом, сервером та блокчейном здійснюється без розкриття чутливої медичної інформації завдяки використанню гомоморфного шифрування.

Деякі запити можуть бути виконані безпосередньо до реляційної бази даних. Наприклад, якщо запит стосується незначної кількості даних або потребує негайного доступу до них, сервер може виконати запит без звернення до блокчейну.

Після формування відповіді, сервер надсилає її клієнту через веб-інтерфейс. Користувач отримує результати свого запиту та може надалі взаємодіяти із системою відповідно до отриманої інформації.

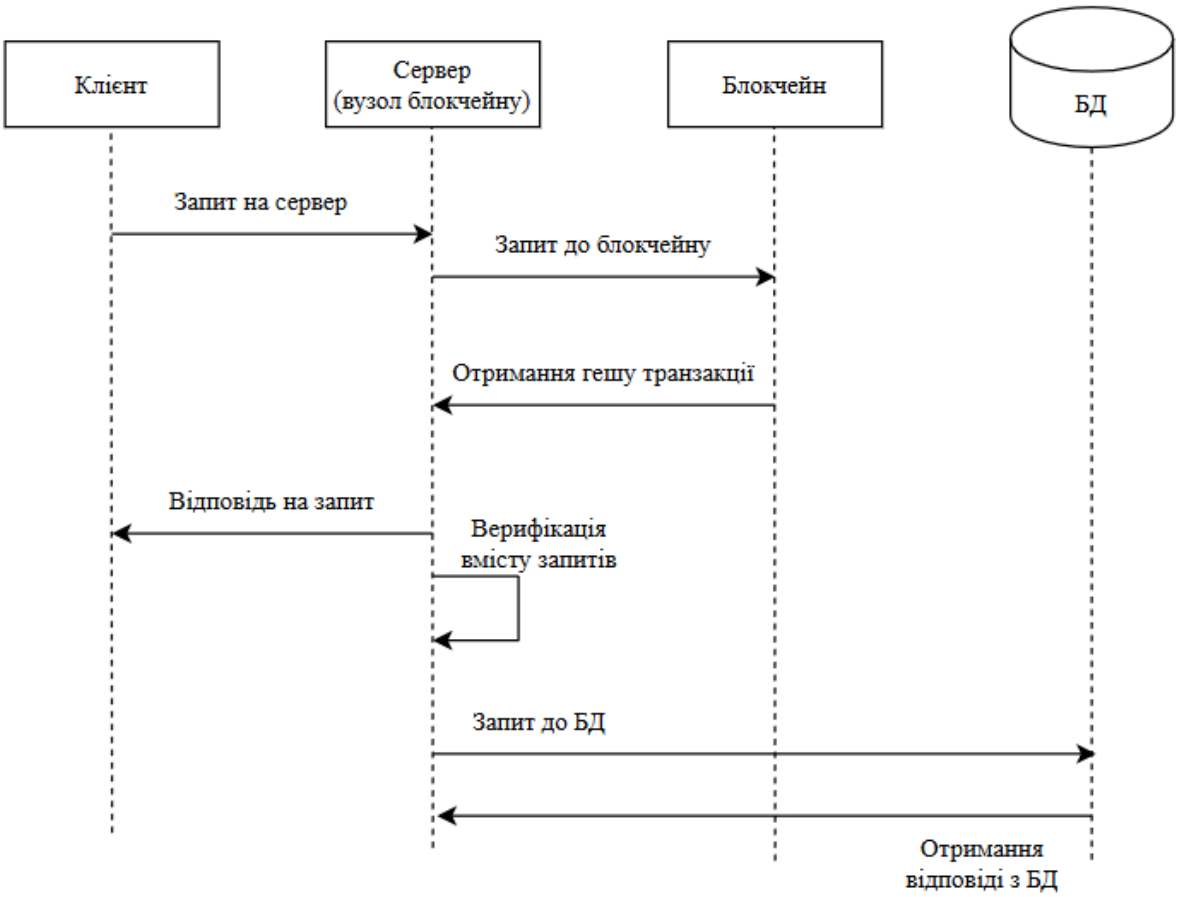


Рисунок 3.2 – Вигляд протоколу взаємодії архітектурних складових

Після формування відповіді, сервер надсилає її клієнту через веб-інтерфейс. Користувач отримує результати свого запиту та може надалі взаємодіяти із системою відповідно до отриманої інформації.

3.2 Смарт-контракти для взаємодії із даними

Необхідно реалізувати смарт-контракти, на основі яких буде формуватись унікальний ідентифікатор пацієнта, перевірка профілю пацієнта в блокчейні, а також відповідні методи, що дозволять виконувати маніпуляції із даними – зберігання, отримання та оновлення [16].

Ідентифікатор формується з комбінації унікальних даних пацієнта, а саме, з рядка, що містить № медичної картки пацієнта та № паспорта. Використовуючи геш-функцію, яка притаманна для блокчейну типу Ethereum – Keccak-256, обчислюється геш-значення та відповідно формується власне сам унікальний ідентифікатор пацієнта, який надалі є основним для успішного функціонування системи.

Метод *store()* є основною функцією для первинного внесення медичних даних пацієнта в блокчейн. Він приймає чотири параметри: ідентифікатор пацієнта, значення зросту, значення ваги та кількість сесій хіміотерапії. Перед збереженням метод виконує перевірку – він переконується, що пацієнт з таким ідентифікатором ще не існує в системі за допомогою перевірки поля exists у відповідній структурі PatientData, що запобігає випадковому або зловмисному перезапису існуючих даних.

Якщо перевірку пройдено успішно, метод створює нову структуру PatientData та заповнює її переданими зашифрованими значеннями Важливо відзначити, що всі дані зберігаються в зашифрованому вигляді.

Метод *retrieve()* призначений для читання збережених даних пацієнта з блокчейну і є view-функцією, що означає, що він не змінює стан контракту та не потребує витрати газу при виклику безпосередньо з вузла. Він приймає один параметр – ідентифікатор пацієнта *patientID*. Метод перевіряє, чи існує запис з таким ідентифікатором, звертаючись до поля exists відповідної структури в

mapping. Якщо пацієнт не знайдений, метод викликає помилку з повідомленням "Patient data not found", що запобігає зчитуванню порожніх або неініціалізованих даних.

Після успішної перевірки метод *retrieve()* у зашифрованому вигляді проводить необхідні гомоморфні перетворення для обчислення дози хіміотерапії. У результаті формується зашифроване значення дози, яке передається користувачу.

На останньому етапі, за допомогою приватного ключа, розшифровується отримане значення та отримується доза хіміотерапевтичного препарату.

Такий підхід передбачає, що усі обчислення виконуються без розкриття медичних даних, зберігаючи конфіденційність та безпеку пацієнта.

Останнім методом є *update()*, який оновлює кількість сесій хіміотерапії за допомогою виконання гомоморфних перетворень, а саме додавання зашифрованого значення одиниці до попередньо зашифрованого значення кількості сесій. Перед виконанням операції відбувається перевірка існування пацієнта.

Важливою особливістю є те, що під час всього процесу оновлення фактична кількість сесій залишається невідомою контракту та будь-якому спостерігачу блокчейну – оновлюються лише зашифровані значення, і тільки власник приватного ключа може розшифрувати результат та дізнатись актуальну кількість пройдених сесій.

Окрім методів роботи з даними пацієнтів, контракт містить функції для виконання обчислень над зашифрованими даними.

Таким чином, усі обчислення відбуваються на блокчейні, за допомогою смарт-контракту, що надає відстежуваність та цілісність даним, а гомоморфне шифрування забезпечує конфіденційність на усіх етапах виконання операцій.

3.3 Узагальнений алгоритм роботи засобу

Після того, як було сформовано узагальнену архітектуру засобу, спроєктовано модель бази даних та описано смарт-контракти, які призначені для

зберігання даних, можна представити узагальнений алгоритм роботи програмного засобу.

Вигляд узагальненого алгоритму роботи засобу, що реалізує метод, відображено на рисунку 3.4.

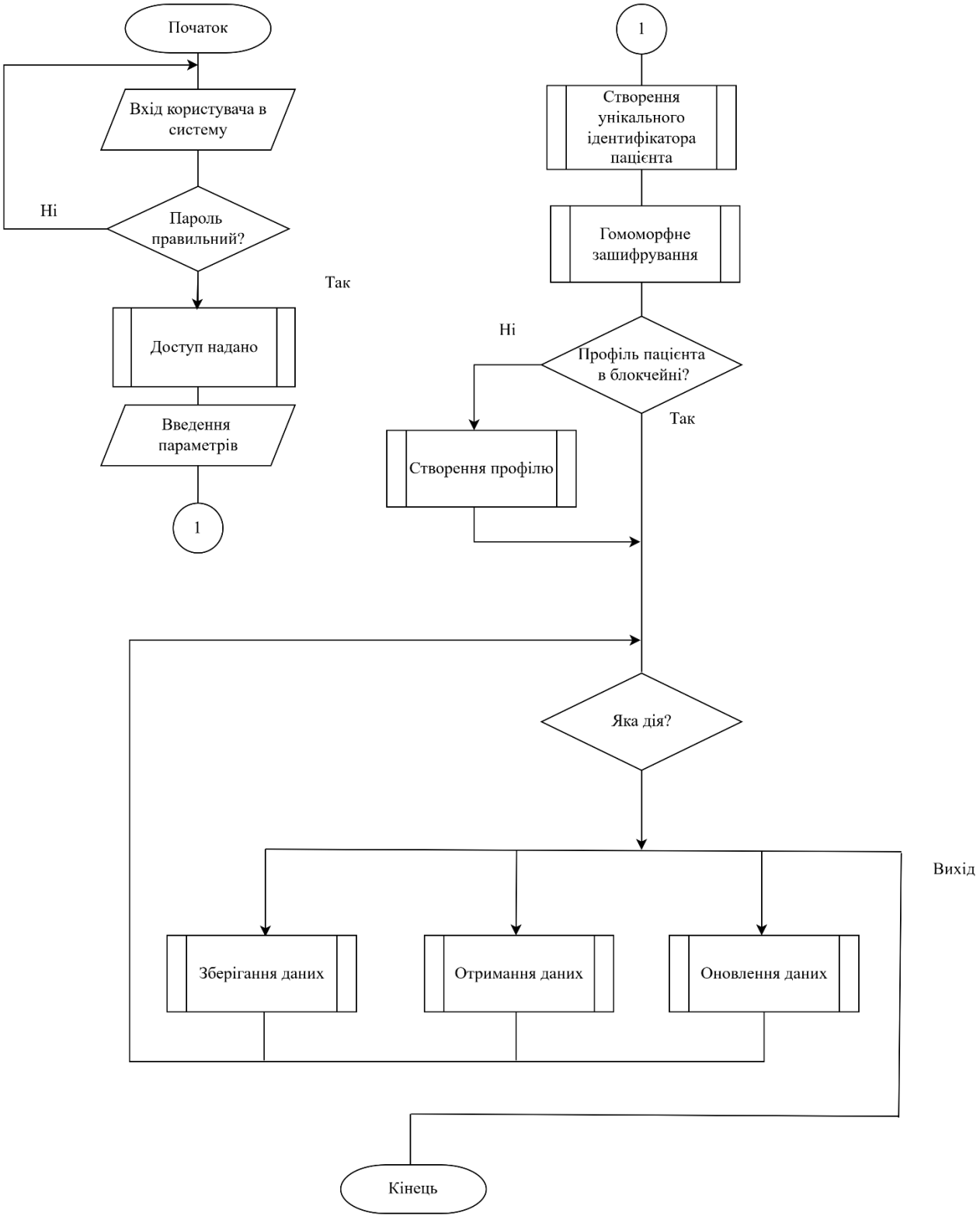


Рисунок 3.4 – Вигляд узагальненого алгоритму роботи засобу

На рисунку 3.5 зображено узагальнений алгоритм роботи процесу зберігання даних.

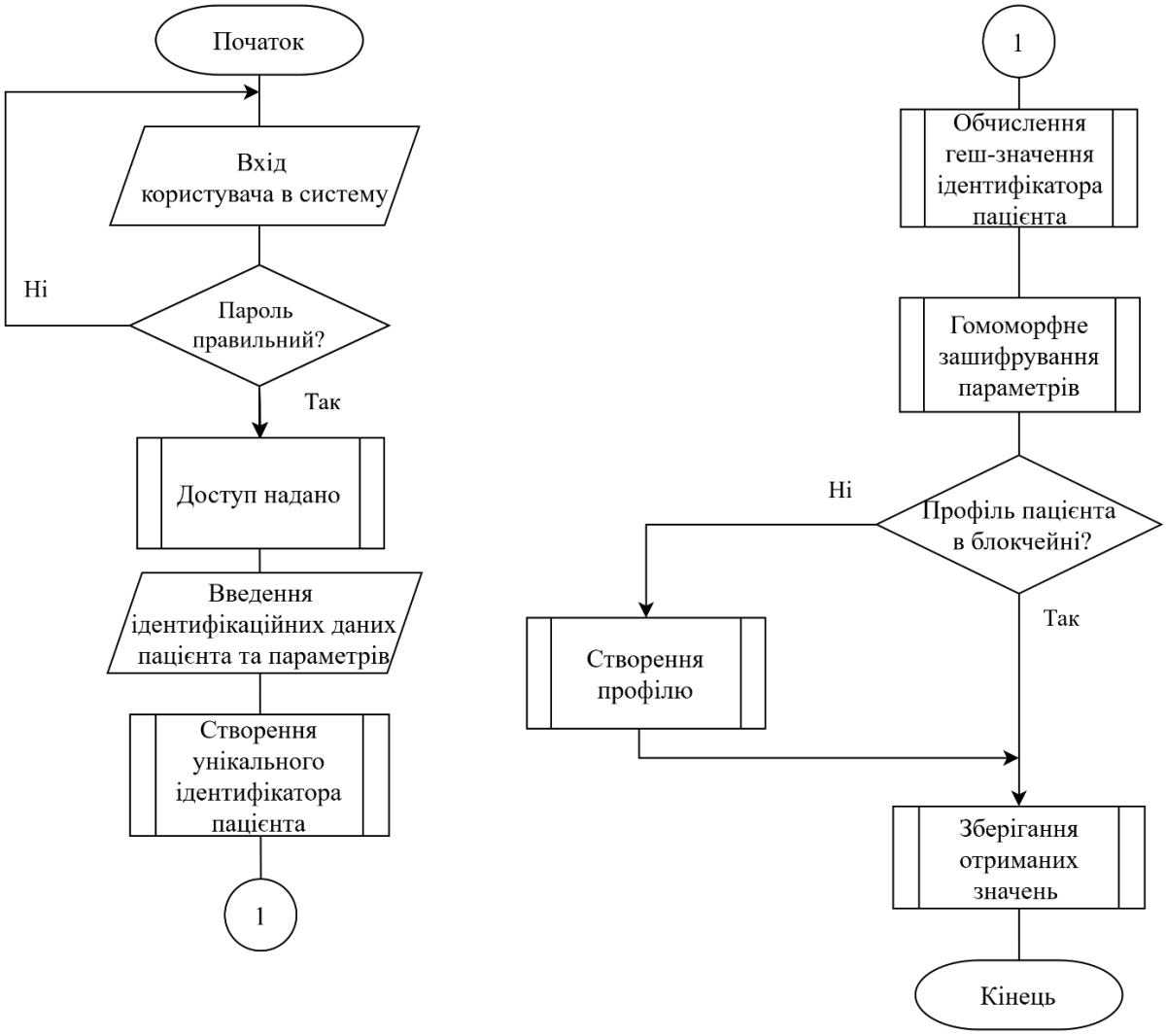


Рисунок 3.5 – Вигляд узагальненого алгоритму роботи процесу зберігання даних

Процес зберігання даних виконується таким чином. Спочатку збираються основні ідентифікаційні дані пацієнта, а саме номер медичної картки та номер паспорта, а також параметри, необхідні для подальших розрахунків дозування хіміотерапії – зріст та вага.

Щоб запобігти розкриттю персональних даних, ідентифікаційний номер пацієнта перетворюється на геш-значення за допомогою криптографічної функції keccak-256, яка природня для блокчейну типу Ethereum. Це геш-значення

використовується як унікальний ідентифікатор (*path*) для зберігання, отримання та оновлення даних пацієнта надалі.

Далі числові параметри, такі як зріст і вага, шифруються за допомогою гомоморфного шифрування з використанням публічного ключа. У результаті отримуються зашифровані значення параметрів, які дозволяють виконувати гомоморфні перетворення над даними без їх розшифрування.

Наступним етапом у смарт-контракті створюється або оновлюється профіль пацієнта, який містить відображення виду $path \rightarrow \{height^e, weight^e, chemotherapySessions^e\}$. На завершення зашифровані параметри $height^e$ і $weight^e$ зберігаються у цьому профілі, що забезпечує захист конфіденційності пацієнта.

Процес отримання даних пацієнта реалізується таким чином (рис. 3.6).

Спершу збираються ідентифікаційні дані пацієнта, на основі яких обчислюється геш-значення за тим самим принципом, що й під час процесу зберігання даних. Потім викликається метод смарт-контракту, який у зашифрованому вигляді проводить необхідні гомоморфні перетворення для обчислення дози хіміотерапії. У результаті формується зашифроване значення дози, яке передається користувачу.

На останньому етапі, за допомогою приватного ключа, розшифровується отримане значення та отримується доза хіміотерапевтичного препарату.

Такий підхід передбачає, щоб усі обчислення виконуються без розкриття медичних даних, зберігаючи конфіденційність та безпеку пацієнта.

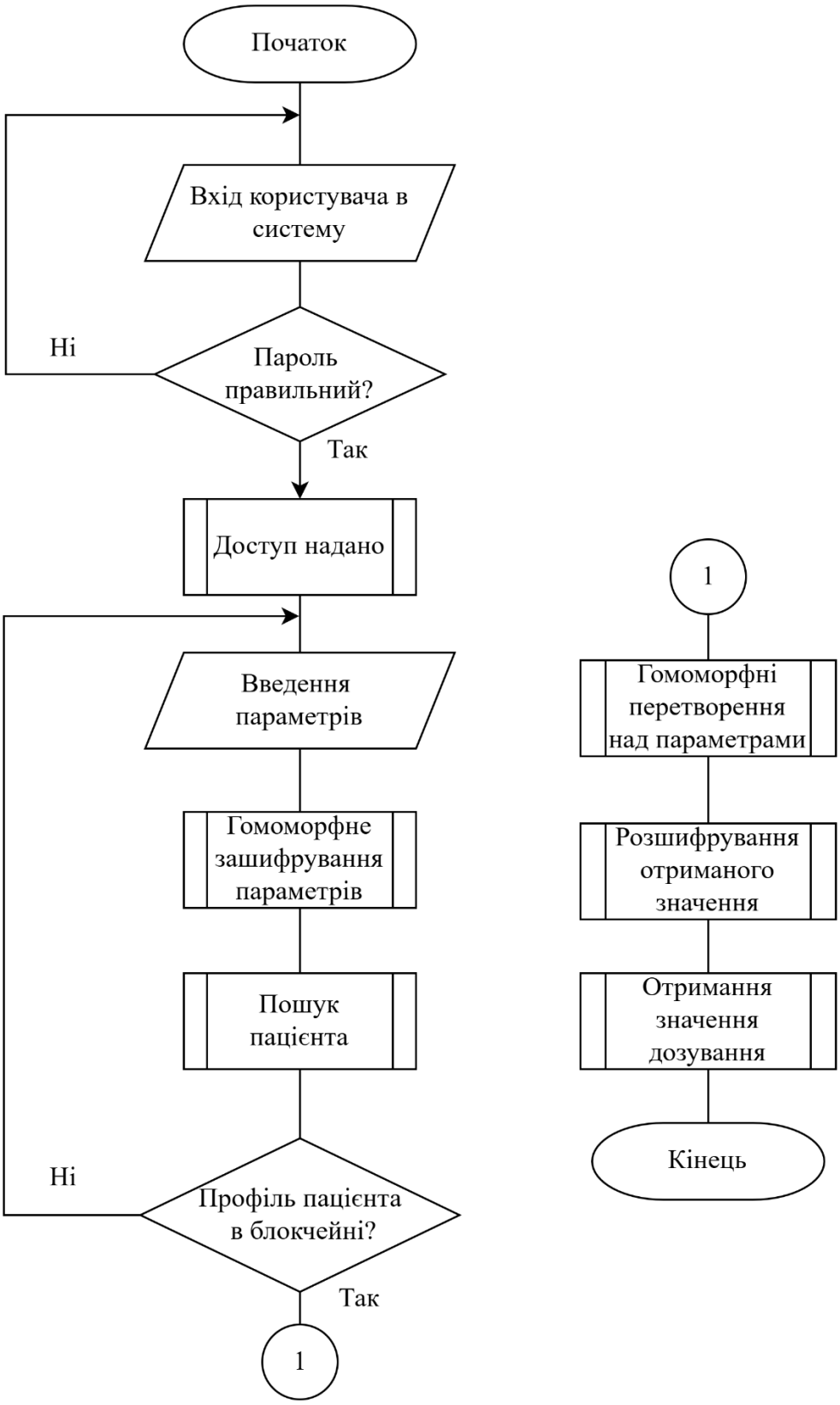


Рисунок 3.6 – Вигляд узагальненого алгоритму роботи процесу отримання даних

Останнім процесом є процес оновлення даних, а саме сесій хіміотерапії. Реалізовано цей процес наступним чином (рис. 3.7).

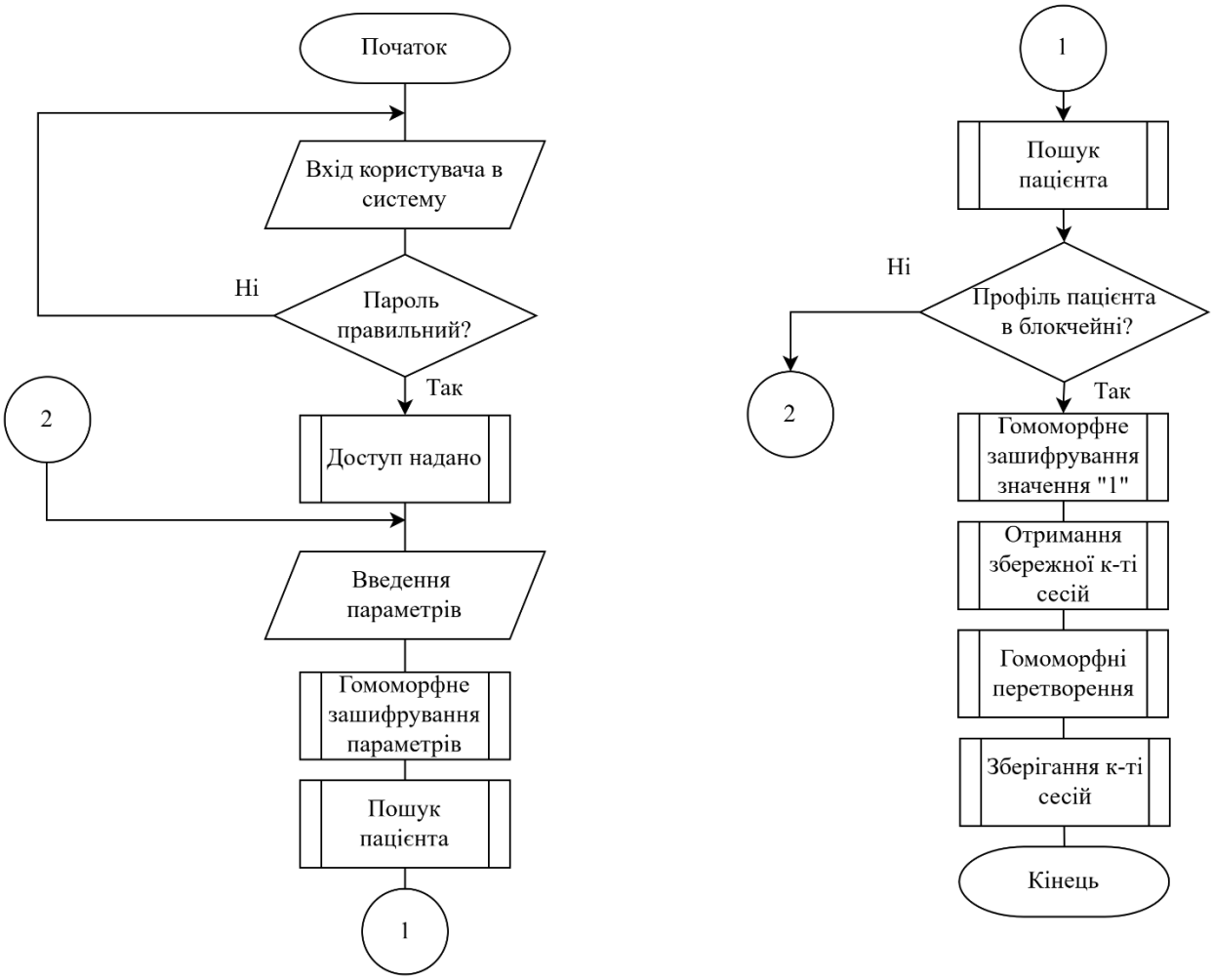


Рисунок 3.7 – Вигляд узагальненого алгоритму роботи процесу оновлення даних

Процес оновлення даних пацієнта реалізується наступним чином. Спершу збираються ідентифікаційні дані пацієнта та обчислюється геш-значення *path*, яке використовується для доступу до даних в смарт-контракті. Далі за допомогою публічного ключа пацієнта шифрується значення «1», що представляє одну нову сесію хіміотерапії. Потім із блокчейну отримується збережене зашифроване значення сесій хіміотерапії, після чого виконуються гомоморфні перетворення, а саме гомоморфне додавання, цього значення до зашифрованої одиниці. У результаті формується оновлене зашифроване значення сесій, яке зберігається в смарт-контракті.

Таким чином, кількість сесій хіміотерапії збільшується на 1, без потреби у розшифруванні даних, забезпечуючи цілісність і конфіденційність персональних даних пацієнта.

3.4 Алгоритм роботи клієнтської частини

Клієнтська частина програмного засобу забезпечує первинну взаємодію користувача із системою та виконує роль інтерфейсу для введення, обробки й відображення результатів. Основним завданням клієнта є збір і підготовка вхідних даних пацієнта, ініціація запитів до серверної частини, а саме до вузла блокчейна, та отримання результатів обчислень у захищеному вигляді (рис. 3.8).

Рисунок 3.8 – Вигляд узагальненого алгоритму роботи клієнтської частини засобу

На боці клієнта також відбувається процес автентифікації користувача в системі, а саме перевірка введених даних для входу. Якщо все введено коректно – доступ надано, в іншому випадку користувач повинен повторно ввести коректні дані, або здійснити реєстрацію кабінету (якщо це є перша взаємодія лікаря з пацієнтом).

Не менш важливою частиною програмного засобу є сервер. У наступному підрозділі буде відображено алгоритм роботи серверної частини.

3.5 Алгоритм роботи серверної частини

Під час створення алгоритму роботи серверної частини застосунку, необхідно звертати увагу на попередній підрозділ – алгоритм роботи модуля клієнта. Узагальнений алгоритм роботи модуля сервера представлений на рис. 3.9.

На узагальненому алгоритмі роботи модуля сервера видно, що спершу йде верифікація користувача, який увійшов до системи. Якщо такий користувач існує – робота модуля продовжується, в іншому випадку – припиняється. Наступним етапом є отримання HTTP(S)-запиту від користувача, це крок, на якому сервер отримує дані, надіслані користувачем. Далі відбувається створення та відповідно надсилання відповіді на модуль клієнта. Після проведених кроків, сервер припиняє свою роботу.

Рисунок 3.9 – Вигляд узагальненого алгоритму роботи модуля сервера

Узагальнений опис роботи модуля обробки запитів від користувача відображений на рисунку 3.10.

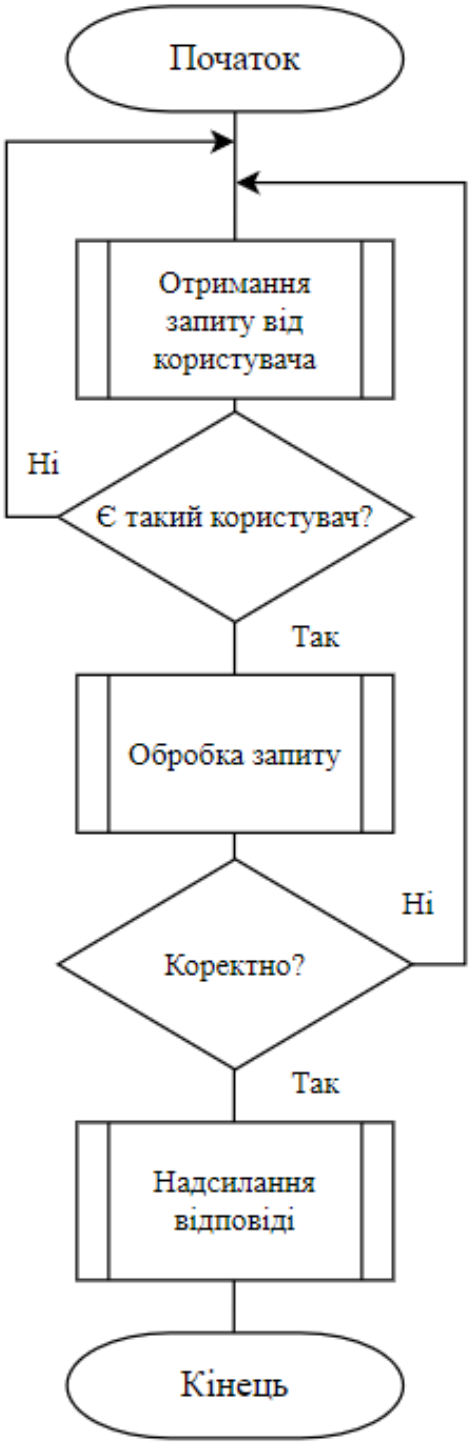


Рисунок 3.10 – Вигляд роботи модуля обробки запитів від користувача

Як видно з рисунку 3.10, одним із важливих моментів обробки запитів є все ж таки перевірка існування користувача в системі. Якщо користувач неіснуючий, робота засобу починається спочатку, допоки до системи не увійде справжній користувач. Таким чином, узагальнений алгоритм роботи модуля сервера починається власне з верифікації користувача.

Не менш важливим етапом є взаємодія з блокчейном (рис. 3.11). Адже саме на боці децентралізованої системи відбуаються гомоморфні перетворення над даними, обчислення результату та передача його на бік клієнтської частини.

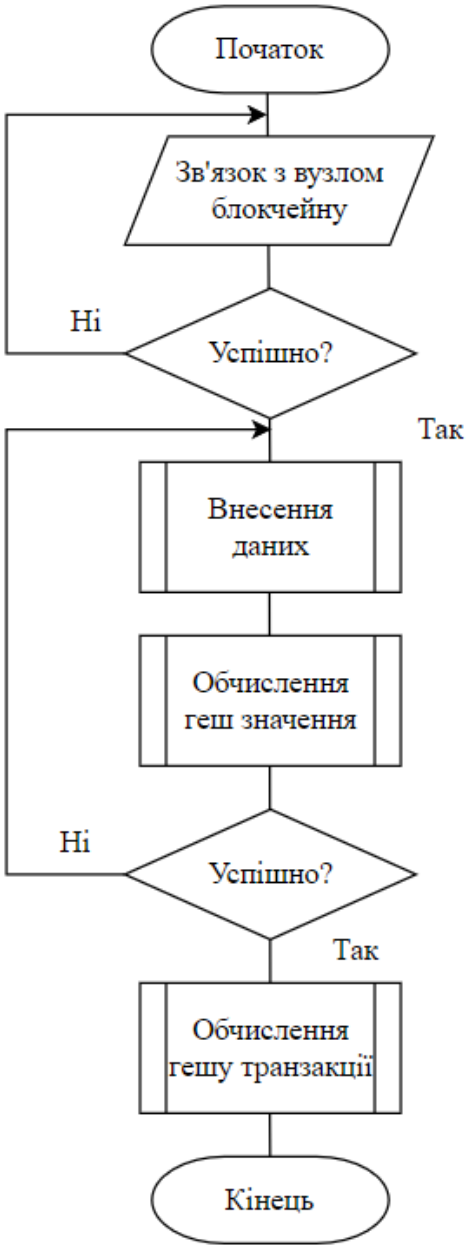


Рисунок 3.11 – Вигляд алгоритму роботи модуля взаємодії з блокчейном

Процес розпочинається зі встановлення з’єднання з вузлом блокчейну. У разі успішного підключення система переходить до наступного етапу, тоді як при невдалій спробі виконується повторне встановлення зв’язку. Після підтвердження з’єднання користувач вносить необхідні дані до системи. На основі введеної інформації виконується обчислення геш-значення. Якщо під час

цього етапу виникає помилка або результат є некоректним, система повертається до повторного введення даних і нового обчислення гешу.

У разі успішного формування геш-значення здійснюється обчислення гешу транзакції, після чого, за умови його коректного виконання, процес вважається завершеним.

3.6 Висновки з розділу

У цьому розділі було представлено архітектуру програмного засобу, яка складається із модулів клієнта та сервера. Сервер в свою чергу представляє вузол блокчейна, який взаємодіє з БД.

Модуль клієнта взаємодіє з користувачем, приймає його запити, перевіряє їх, створює HTTP(S) запити і обробляє отримані відповіді. Також саме на боці клієнта відбувається автентифікація користувача в системі.

Побудова алгоритму роботи серверної частини засобу базується на клієнтській частині, тому на основі попередніх алгоритмів було побудовано алгоритми роботи серверної частини, а саме узагальнений алгоритм роботи серверної частини та алгоритм обробки запитів від користувача. Кожен із процесів взаємодії із даними представлено у вигляді схем алгоритмів, для кращого розуміння структури роботи програмного засобу. Далі було було описано алгоритм роботи смарт-контрактів для взаємодії із даними.

Таким чином, на основі вищевказаного, можна перейти до експериментального дослідження та тестування роботи реалізованого програмного засобу.

4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ТА ТЕСТУВАННЯ

4.1 Обґрунтування вибору засобів розробки

Засіб, що реалізує метод, має клієнт-серверну архітектуру, тому вибір інструментів здійснювався з урахуванням вимог до розподіленості, взаємодії зі смарт-контрактами, високої безпеки даних та можливості швидкого тестування програмних компонентів.

У процесі аналізу різних середовищ було проведено порівняння альтернативних рішень, що дозволило обґрунтовано визначити найбільш доцільні інструменти. Серед середовищ програмування порівнювалися Visual Studio Code [37], IntelliJ IDEA/WebStorm [38, 39] та прості текстові редактори на кшталт Sublime Text [40].

IntelliJ IDEA та WebStorm забезпечують глибоку інтеграцію для окремих мов програмування та високу стабільність, проте є комерційними продуктами з обмеженими безкоштовними версіями, а їхня екосистема розширень є менш гнучкою для проєктів зі змішаними стеком технологій. Sublime Text та подібні редактори працюють швидко, але не надають повноцінної підтримки Solidity і вимагають ручного налаштування інструментів складання та аналізу. На цьому фоні Visual Studio Code вирізняється безкоштовністю, кросплатформністю, відкритою архітектурою та великою кількістю розширень, зокрема інтеграцією з Node.js [41], Git [42], Solidity [43] та інструментами тестування. Це робить його найбільш гнучким і придатним для багатомодульного проєкту з блокчейн- та криптографічними компонентами.

Під час аналізу інструментів розробки та тестування смарт-контрактів порівнювалися Truffle [44], Hardhat [45], Brownie [46] та Remix IDE [47]. Hardhat є сучасним фреймворком із широкими можливостями конфігурації, однак потребує складнішого налаштування та інтеграції додаткових плагінів. Brownie пропонує високий рівень автоматизації, але орієнтований на Python-середовище, що ускладнює роботу з Node.js-орієнтованим серверним кодом. Remix IDE зручний для швидких експериментів, але як браузерне середовище має обмежену

інтеграцію з багатомовними проєктами, не підходить для комплексних систем і не забезпечує гнучкості у взаємодії з серверною логікою чи складною файловою структурою. Truffle, натомість, пропонує повноцінну організацію проєкту, автоматизовані міграції, зручне тестування і глибоку інтеграцію з Visual Studio Code, що робить його якісним вибором для проєкту з чіткою структурою та поділом на модулі.

Окремо було розглянуто середовища для виконання смарт-контрактів під час розробки — публічні тестові мережі та локальні блокчейни. Використання мереж Goerli [48] чи Sepolia [49] дозволяє тестувати контракти в умовах, близьких до реальної мережі, однак потребує отримання тестових токенів, залежить від стабільності RPC-сервісів і значно повільніше через час підтвердження блоків. Локальний блокчейн Ganache [50] забезпечує повністю ізольоване середовище, миттєве виконання транзакцій, відсутність витрат на газ і можливість налаштування параметрів мережі, що значно прискорює процес налагодження. Саме тому Ganache було обрано як базовий інструмент для тестування.

Для статичного аналізу безпеки порівняно три популярні інструменти: Slither [51], Mythril [52] та Oyente [53]. Mythril використовує метод, за допомогою якого замість реальних значень змінні під час аналізу підміняються абстрактними (символічними)., що дає змогу знайти складні логічні вразливості, але робить аналіз повільнішим і ресурсозатратним. Oyente вважається одним із перших інструментів аналізу для Solidity, проте оновлюється нерегулярно та не підтримує сучасні версії мови. Slither, розроблений Trail of Bits, пропонує широку базу правил, швидку роботу, детальний аналіз структури контракту та підтримку актуальних версій Solidity. Завдяки цьому він є найбільш збалансованим інструментом за точністю, швидкістю та актуальністю.

На основі проведеного порівняння найбільш доцільними для проєкту були визначені: Visual Studio Code як основне середовище розробки, Truffle та Ganache як інструменти розгортання та тестування смарт-контрактів, а також Slither — як базовий інструмент статичного аналізу безпеки. Такий вибір поєднує

гнучкість, функціональність та зручність розробки для комплексної клієнт-серверної системи з використанням блокчейн-технологій.

Таким чином, після обґрунтування засобів для розробки можна перейти власне до експериментального дослідження схем гомоморфного шифрування.

4.2 Експериментальне дослідження схем гомоморфного шифрування

Для аналізу швидкості зашифрування та розшифрування було обрано 4 схеми гомоморфного шифрування – дві частково гомоморфні схеми, а також дві повністю гомоморфні схеми.

Аналіз було виконано використовуючи ключ довжиною в 2048 біт.

Вхідні дані для зашифрування: згенерований випадковим чином байтовий рядок.

Параметри комп’ютера, на якому виконувався аналіз, наведені нижче.

Процесор: 12th Gen Intel(R) Core(TM) i7-1255U (1.70 GHz).

ОЗП: 16,0 ГБ.

Тип системи: 64-розрядна операційна система, процесор на базі архітектури x64.

Для наочного представлення експериментального дослідження було побудовано порівняльну таблицю (табл. 4.1).

Таблиця 4.1 – Порівняльний аналіз схем гомоморфного шифрування

Алгоритм	Час генерування ключів, мс	Час зашифрування, мс	Час розшифрування, мс
Paillier	0.68	0.31	0.28
El-Gamal	0.37	0.63	0.42
BFV	0.66	0.65	0.13
BGV	0.72	0.34	0.13

Таким чином, середній час генерації ключів для схеми Пальєра становить 0.68 мс, що є середнім показником серед усіх алгоритмів. ElGamal демонструє найшвидшу генерацію ключів (0.37 мс, що на 45.6% швидше за схему Пальєра),

проте BGV має найповільнішу генерацію (0.72 мс, на 5.9% повільніше за Пальєра).

Щодо швидкості шифрування, Paillier показує найкращий результат з 0.31 мс, що на 50.8% швидше за ElGamal (0.63 мс), на 52.3% швидше за BFV (0.65 мс) та на 8.8% швидше за BGV (0.34 мс).

При розшифруванні даних Paillier потребує 0.28 мс, що швидше за ElGamal на 33.3% (0.42 мс), але повільніше за BFV та BGV (обидва по 0.13 мс) на 53.6%.

Схема Пальєра підтримує адитивне гомоморфне шифрування, що дозволяє виконувати додавання зашифрованих значень та множення зашифрованого значення на константу без розшифрування.

ElGamal забезпечує мультиплікативне гомоморфне шифрування для операцій множення над зашифрованими даними [9, 10].

BFV та BGV є повністю гомоморфними схемами, що теоретично підтримують необмежену кількість операцій додавання та множення, однак на практиці мають обмеження через накопичення шуму.

Використовуючи схему Пальєра, шифрування початкових даних займає 0.31 мс, гомоморфне додавання одиниці виконується за час множення двох чисел за модулем n^2, а розшифрування для перевірки займає 0.28 мс. Загальний час операції становить 0.59 мс.

При використанні ElGamal той самий процес потребує 0.63 мс на шифрування, але для додавання одиниці необхідно розшифрувати значення (0.42 мс), додати одиницю в відкритому вигляді та знову зашифрувати (0.63 мс), що дає загальний час 1.68 мс. Також постає питання щодо забезпечення повної конфіденційності даним на всіх етапах обчислень.

BFV та BGV можуть виконати гомоморфне додавання швидше, але їхній складний механізм управління шумом додає накладні витрати приблизно 0.3-0.5 мс на кожну операцію, роблячи загальний час однаковим або повільнішим.

Таким чином, можна зробити висновок, що для цієї розробки найкраще підійде частково гомоморфна схема Пальєра, яка підтверджує відповідність для криптографічних операцій, де швидкість є вагомим показником.

4.3 Модульне тестування

Основна ідея модульного тестування полягає в тому, щоб перевірити, чи працює окремий блок програмного коду (одиниця програмного коду) правильно і відповідно до очікувань.

Для тестування було обрано бібліотеку Jest [54], яка є зручною та доступною для використання.

Оскільки основним процесом є все ж взаємодія із гомоморфним шифруванням, тому було протестовано функції, що стосуються цього блоку розробки.

Тестування виконується за допомогою фреймворку Jest, який забезпечує модульний підхід до перевірки окремих компонентів системи. Тести поділені на кілька логічних груп, кожна з яких відповідає за конкретний аспект роботи алгоритму — від генерації ключів до перевірки гомоморфних властивостей. Така структура забезпечує системність і достовірність перевірки криптографічних функцій, що є важливим при розробці безпечних систем обробки медичних даних.

Перший блок тестів — «Paillier Key Generation Tests» перевіряє правильність створення ключової пари (рис. 4.1). У межах цього блоку тестів перевіряється, чи функція *generatePrime()* дійсно генерує прості числа необхідної довжини, чи є вони різними для кожного виклику та чи відповідають математичним вимогам простоти. Крім того, перевіряється, що функція *generateKeys()* створює коректну пару ключів — публічний *(n, g)* та приватний (λ, μ). Тести оцінюють відповідність довжини модуля n заданій розрядності, а також перевіряють математичні властивості ключів: $n = p \times q, g = n + 1,$ та $n^2 = n.pow(2)$. Додатковий тест підтверджує, що добуток $\mu * L(g\hat{}\lambda \, mod \, n^2)$ дійсно дорівнює одиниці за модулем n, що є ключовою умовою коректності схеми Пальєра.

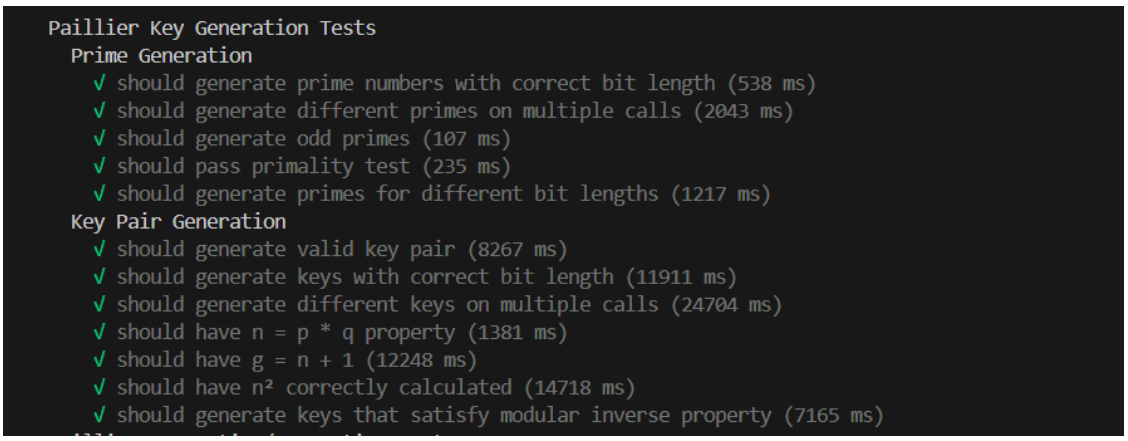


Рисунок 4.1 – Вигляд результату тестування блока «Paillier Key Generation Tests»

Другий блок — «Paillier Encryption/Decryption Tests» (рис. 4.2) спрямований на перевірку основної криптографічної функціональності: зашифрування та розшифрування.

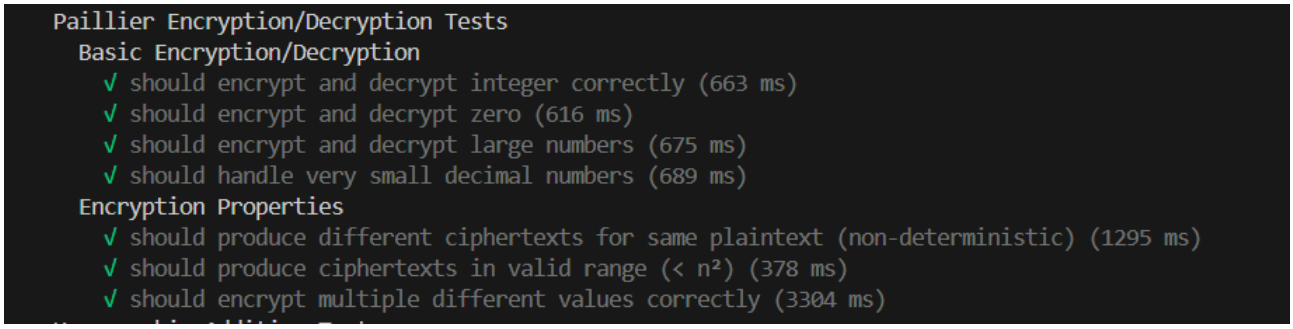


Рисунок 4.2 – Вигляд результату тестування блока «Paillier Encryption/Decryption Tests»

Третій блок тестів — «Homomorphic Addition Tests» (рис. 4.3) досліджує основну властивість гомоморфного шифрування схеми Пальєра: можливість виконання операцій додавання над зашифрованими даними без їх розшифрування.

У тестах перевіряється, що добуток двох шифротекстів $E(a) \times E(b)$ після розшифрування дає суму відкритих значень $(a + b)$. Тести охоплюють додавання звичайних чисел, випадок додавання нуля, а також послідовне додавання кількох зашифрованих значень.

Додатково перевіряється комутативність операції — тобто, $E(a) \times E(b)$ та $E(b) \times E(a)$ дають однаковий результат після розшифрування, що підтверджує коректність реалізації алгоритму.

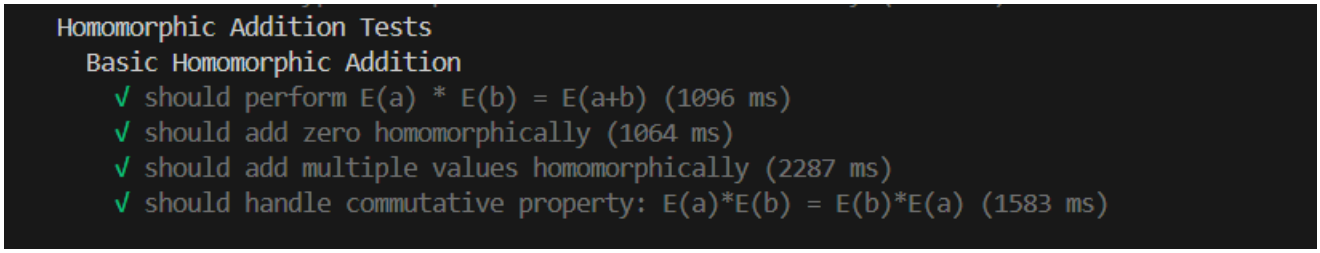


Рисунок 4.3 – Вигляд результату тестування блока «Homomorphic Addition Tests»

Таким чином, всього було створено 23 тести та усі вони успішно пройдені, що відображено на рисунку 4.4.

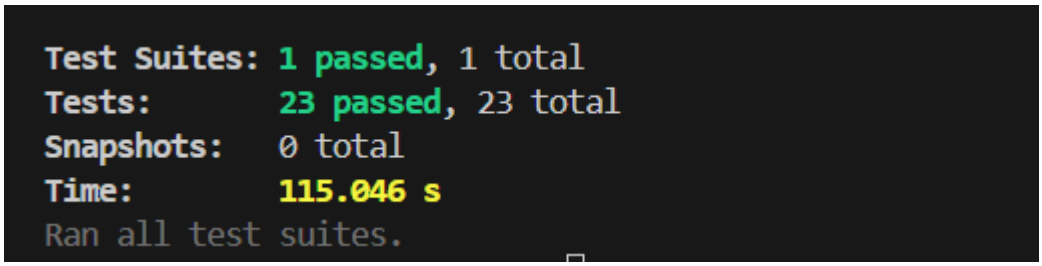


Рисунок 4.4 – Вигляд успішно пройдених тестів

Отже, можна висунути гіпотезу, що програмний засіб працює належним чином, однак слід перевірити це за допомогою інтеграційного тестування.

4.4 Статичне тестування безпеки

Статичне тестування безпеки (SAST) відіграє важливу роль у процесі розробки смарт-контрактів, забезпечуючи раннє виявлення потенційних загроз безпеці. SAST проводить глибокий аналіз вихідного коду без його виконання, виявляючи такі вразливості, як помилки в контролі доступу, недоліки в захисті даних і можливі витоки інформації.

Використання цього підходу дозволяє розробникам вчасно виправляти проблеми, що знижує ризик атак і неправильного використання контрактів у майбутньому.

Для тестування безпеки смарт-контрактів було обрано фреймворк Slither, вихідний код якого написаний на мові Python. Даний фреймворк дозволяє виконати детальний аналіз коду смарт-контракту, до того ж, на відміну від інших фреймворків, Slither після тестування дає вказівки для покращення коду та пояснює причину виявленого недоліку.

Результат тестування безпеки смарт-контрактів наведений на рис. 4.5.

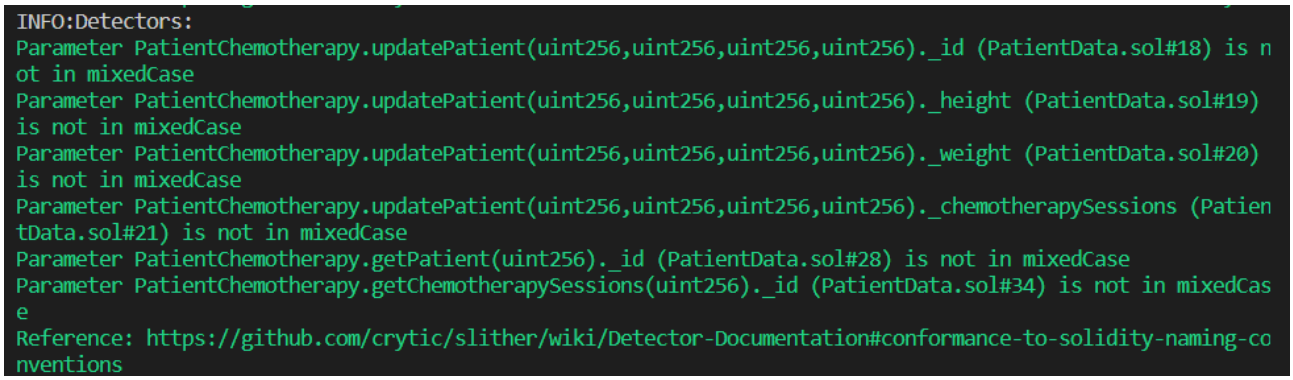


Рисунок 4.5 – Вигляд результату статичного тестування безпеки смарт-контрактів

У межах статичного аналізу безпеки застосунку було виконано додаткове комплексне SAST-тестування та перевірку безпеки вихідного коду за допомогою платформи Aikido Security [55]. Аналіз охопив декілька ключових напрямів: контроль залежностей відкритого коду, моніторинг можливих витоків секретів, аналіз ліцензій програмних компонентів, перевірку конфігурацій Infrastructure as Code, пошук шкідливого коду в залежностях, а також оцінку налаштувань доступу до репозиторію.

Платформа Aikido автоматично проаналізувала залежності проєкту, перевіривши їх на наявність відомих вразливостей у сторонніх бібліотеках. Жодних ризикових або застарілих компонентів виявлено не було, що свідчить про актуальний та безпечний стек технологій. Моніторинг секретів також не засвідчив наявності у вихідному коді конфіденційних даних, таких як токени, паролі, ключі доступу чи інші чутливі змінні. Це вказує на належний підхід до керування секретами та уникнення практик, що можуть спричинити витік.

Перевірка ліцензій підтвердила, що всі використані залежності мають допустимі та сумісні умови використання, тобто проєкт не порушує вимог ліцензування та може безперешкодно масштабуватися або комерціалізуватися. Аналіз конфігурацій IaC також не виявив неправильних або небезпечних налаштувань інфраструктури. Перевірки на наявність шкідливого ПЗ в залежностях показали відсутність компонентів, що містять потенційно небезпечні або компрометовані пакети.

Статичне тестування вихідного коду не зафіксувало жодних вразливостей у логіці застосунку, структурі коду чи способах обробки даних, що свідчить про загалом безпечне програмування та відсутність критичних помилок, пов’язаних із валідацією, автентифікацією, обробкою вхідних даних чи правами доступу (рис. 4.6).

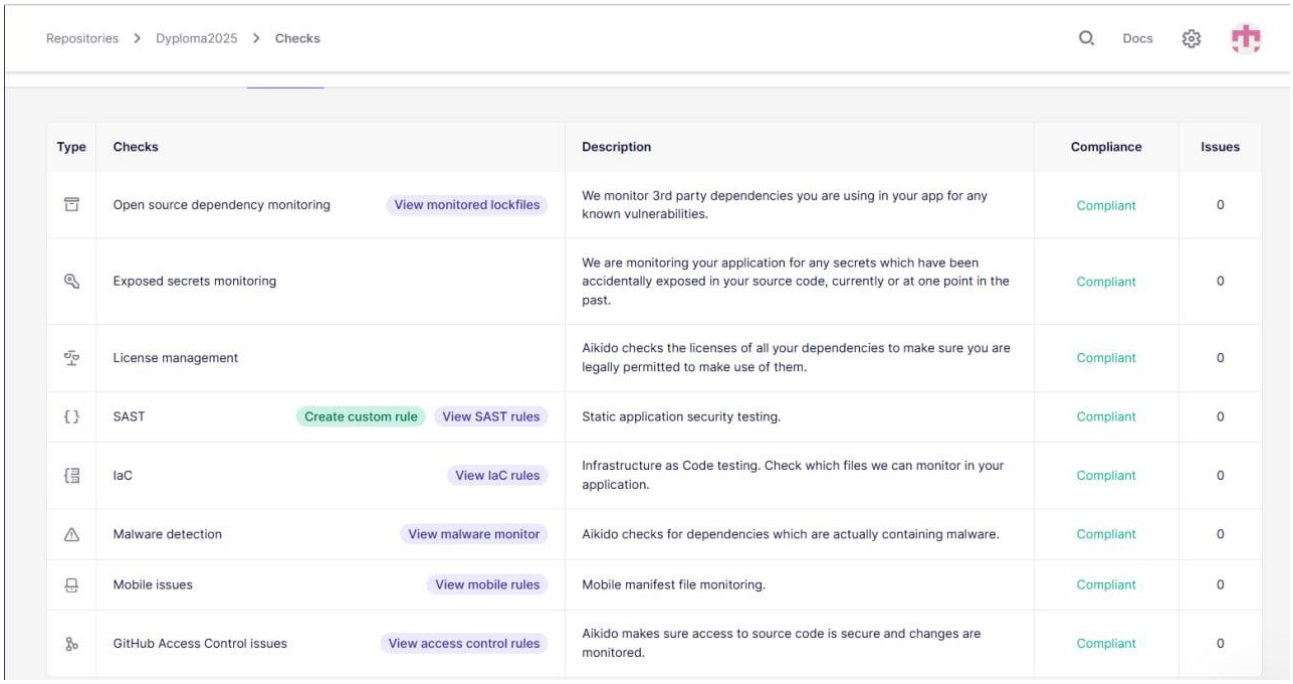


Рисунок 4.6 – Вигляд звіту зі статичного тестування безпеки проєкту за допомогою платформи Aikido Security

Додатково Aikido перевірив налаштування GitHub-доступу, підтвердивши, що контроль доступу до репозиторію налаштований коректно і не створює загрози компрометації коду.

Таким чином, статичний аналіз безпеки за допомогою фреймворку Slither та Aikido Security продемонстрував повну відповідність проєкту вимогам безпеки, відсутність проблем у залежностях, конфігураціях, ліцензіях та самому коді. Отримані результати свідчать про високий рівень базової безпеки та якісну організацію процесів розробки.

Отже, можна перейти до динамічного тестування безпеки, яке також є не менш важливим елементом.

4.5 Динамічне тестування безпеки

У ході динамічного тестування веб-застосунку за допомогою інструмента OWASP ZAP [56] було виконано автоматичне сканування за вказаною адресою https://localhost:8081 (адреса, за якою розгорнуто розроблений програмний засіб).

Сканування включало використання стандартного павука, а також AJAX-spider для глибшого обходу сторінок. На етапі аналізу ZAP перевірив засіб на наявність типових веб-вразливостей, зокрема SQL-ін’єкцій, XSS-атак, помилок конфігурації сервера, виконання стороннього коду, включення файлів, обходу каталогів, ін’єкцій у командний рядок, небезпечних XML-обробників, CRLF-ін’єкцій, уразливостей SOAP/XPath тощо (рис. 4.7, 4.8).

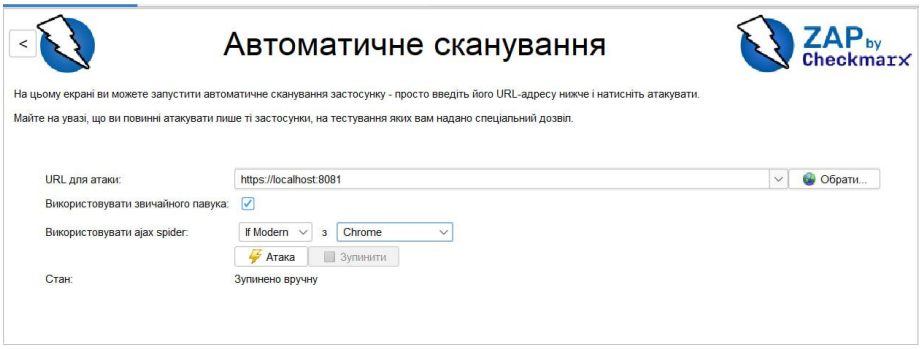
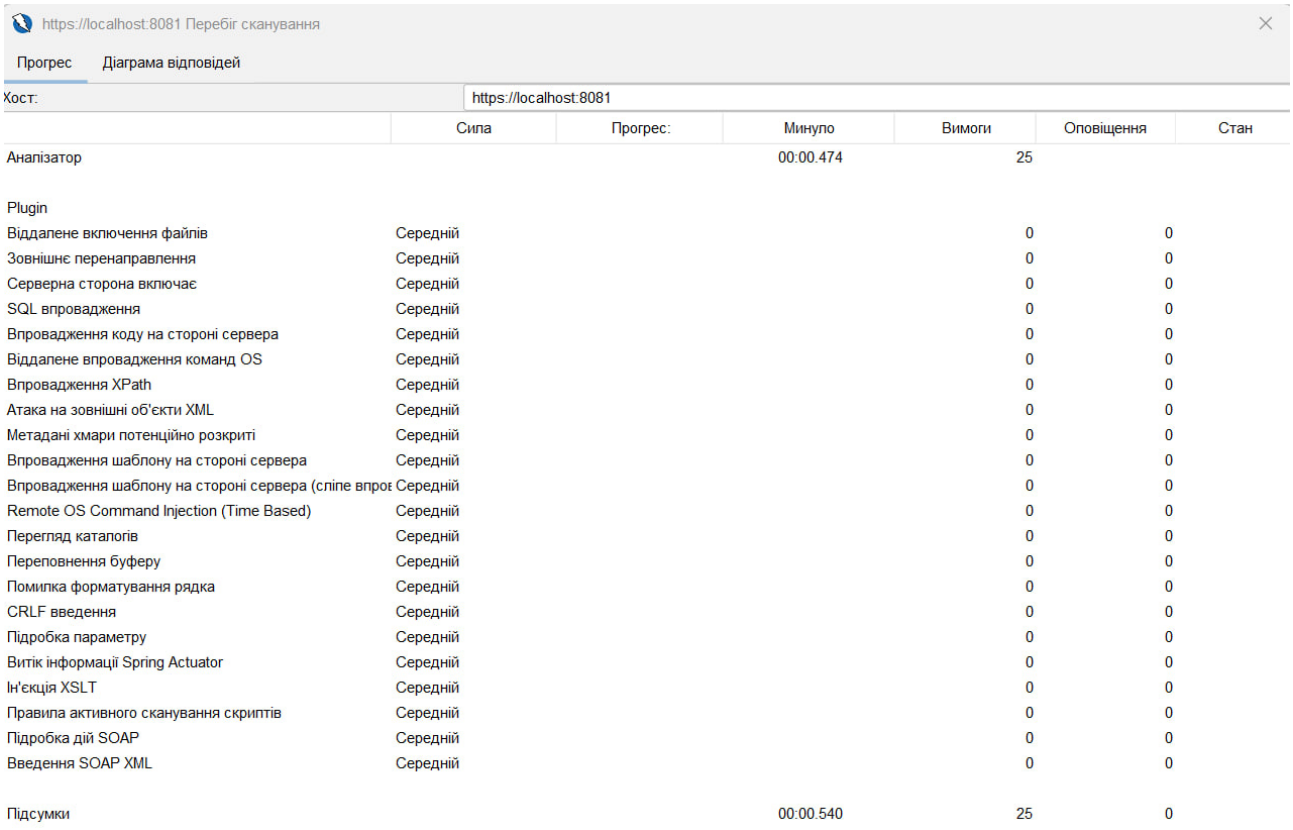


Рисунок 4.7 – Вигляд вікна з налаштуванням сканування


https://localhost:8081 Перебіг сканування

Прогрес | Діаграма відповідей

Хост: https://localhost:8081

	Сила	Прогрес:	Минуло	Вимоги	Оповіщення	Стан
Аналізатор			00:00.474	25		
Plugin						
Віддалене включення файлів	Середній			0	0	
Зовнішнє перенаправлення	Середній			0	0	
Серверна сторона включає	Середній			0	0	
SQL впровадження	Середній			0	0	
Впровадження коду на стороні сервера	Середній			0	0	
Віддалене впровадження команд OS	Середній			0	0	
Впровадження XPath	Середній			0	0	
Атака на зовнішні об'єкти XML	Середній			0	0	
Метадані хмари потенційно розкриті	Середній			0	0	
Впровадження шаблону на стороні сервера	Середній			0	0	
Впровадження шаблону на стороні сервера (сліпе впрос	Середній			0	0	
Remote OS Command Injection (Time Based)	Середній			0	0	
Перегляд каталогів	Середній			0	0	
Переповнення буферу	Середній			0	0	
Помилка форматування рядка	Середній			0	0	
CRLF введення	Середній			0	0	
Підробка параметру	Середній			0	0	
Витік інформації Spring Actuator	Середній			0	0	
Ін'єкція XSLT	Середній			0	0	
Правила активного сканування скриптів	Середній			0	0	
Підробка дій SOAP	Середній			0	0	
Введення SOAP XML	Середній			0	0	
Підсумки			00:00.540	25	0	

Рисунок 4.8 – Вигляд звіту з проведеного сканування

За результатами сканування всі перевірки були завершені коректно, при цьому система не виявила жодних критичних чи середніх вразливостей. Кількість оповіщень дорівнює нулю, що свідчить про відсутність типових загроз або некоректної поведінки застосунку під час взаємодії зі сканером. Отримані дані можуть свідчити про достатній базовий рівень безпеки тестованого середовища, коректну маршрутизацію запитів, відсутність відкритих ендпоінтів, що дозволяють виконати шкідливий код, та загалом про безпечну обробку введених даних.

4.6 Інтеграційне тестування

Інтеграційне тестування полягає в тому, що відбувається перевірка наскільки добре взаємодіють різні блоки програми.

Спершу необхідно протестувати модуль авторизації користувача в системі.

Важливим моментом є також розмежування прав доступу, тому було здійснено поділ на такі ролі, як лікар та медсестра. Лікарю доступні усі етапи

взаємодії з даними пацієнтів, водночас медсестра має доступ лише до оновлення сесій.

У ролі користувача виступає лікар, який авторизується в системі, після запуску сервера він переходить за посиланням:

http://127.0.0.1:8081/login.html

Перед лікарем відображається форма авторизації в системі (рис. 4.9).

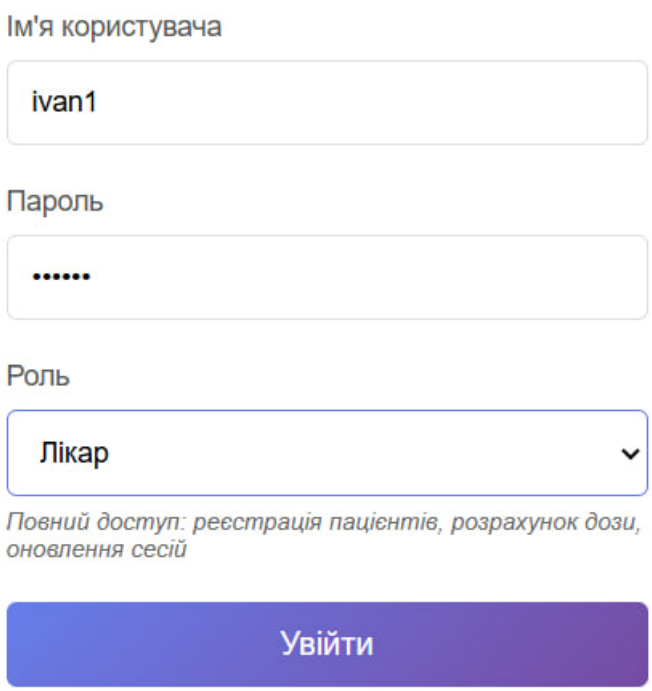


Рисунок 4.9 – Вигляд вікна авторизації для ролі лікаря

Після успішної авторизації лікаря направляє на наступне вікно, де відображено три дії, які він може виконати в системі:

– реєстрація;
– розрахунок;
– сесії.

Варто розпочати із першої дії, що стосується реєстрації пацієнтів, а саме виконання процесу *store()*, про який вказано в методі. Перед лікарем відображена форма, яка має наступні поля:

– № паспорта;
– № медичної картки;

– зріст;
– вага;
– кількість сесій.

Для формування унікального ідентифікатора необхідно заповнити в формі поля № паспорта та № медичної картки. За допомогою конкатенації рядків та після обчислення геш-значення в блокчейн буде записано унікальниц ідентифікатор пацієнта (patientID).

Наступним кроком варто заповнити поля, що стосуються параметрів пацієнта, які знадобляться для обчсилення дозування препарату хіміотерапії, а саме зріст та вага, які виражені в сантиметрах та кілограмах відповідно.

Останнім полем в формі є введення кількості проведених сесій хіміотерапії, тобто мається на увазі, що варто внести вже ту кількість сесій, яка була проведена пацієнту до цього моменту. Після успішного заповнення форми та натискання кнопки Зберегти (рис. 4.10) пацієнт зберігається в системі, про що свідчить відповідна транзакція в тестовому середовищі Ganache (рис. 4.11).

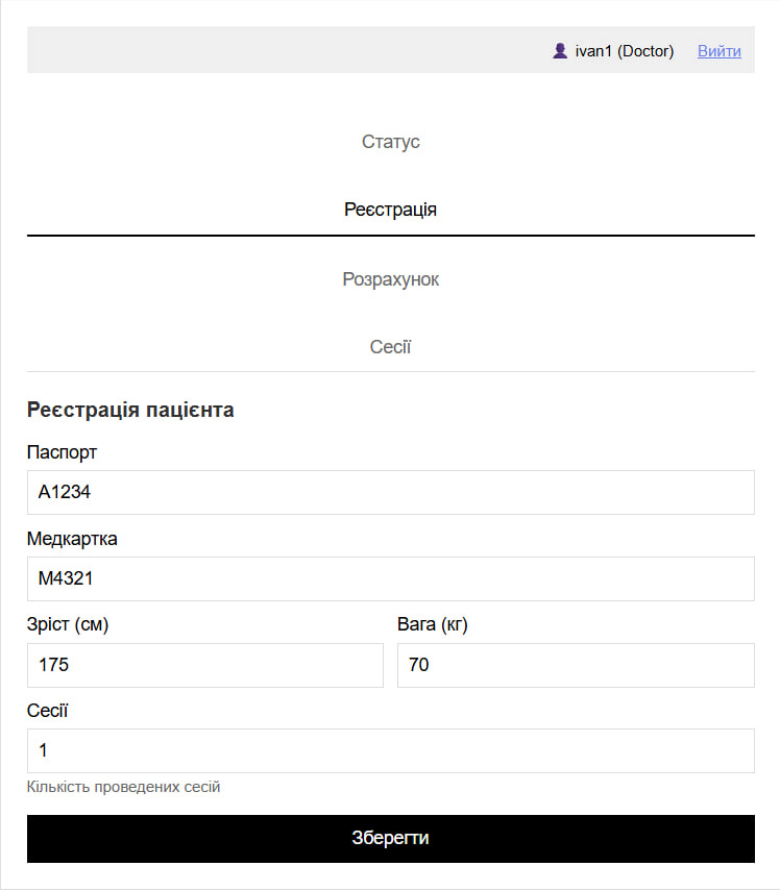


Рисунок 4.10 – Вигляд успішно заповненої форми зберігання даних пацієнта

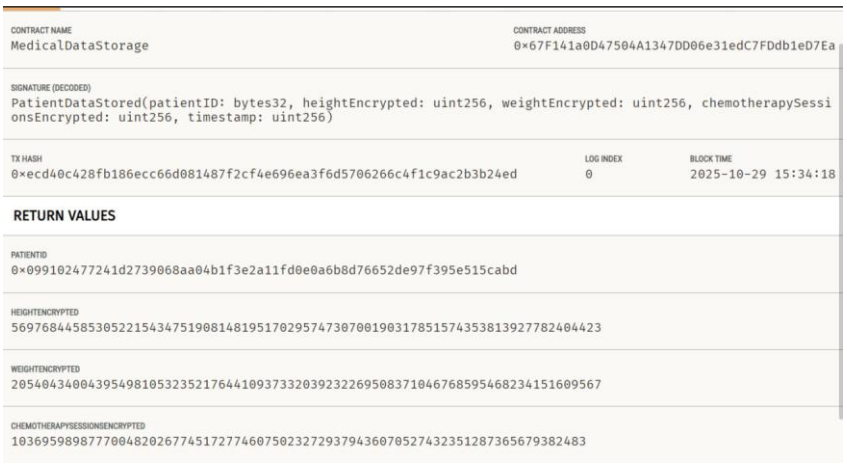


Рисунок 4.11 – Вигляд успішної транзакції в тестовому середовищі Ganache

Наступним етапом, після того, як пацієнта було збережено в системі, доцільно перейти до наступного кроку, а саме розрахунку дозування препарату хіміотерапії.

Після введення значень № медичної картки та № паспорта в систему, формується унікальний ідентифікатор пацієнта, та якщо такий пацієнт вже збережений, відповідно обчислюється дозування препарату за формулами (2.5), (2.6). Результат успішного розрахунку дозування препарату хіміотерапії для конкретного пацієнта наведено на рисунку 4.12.

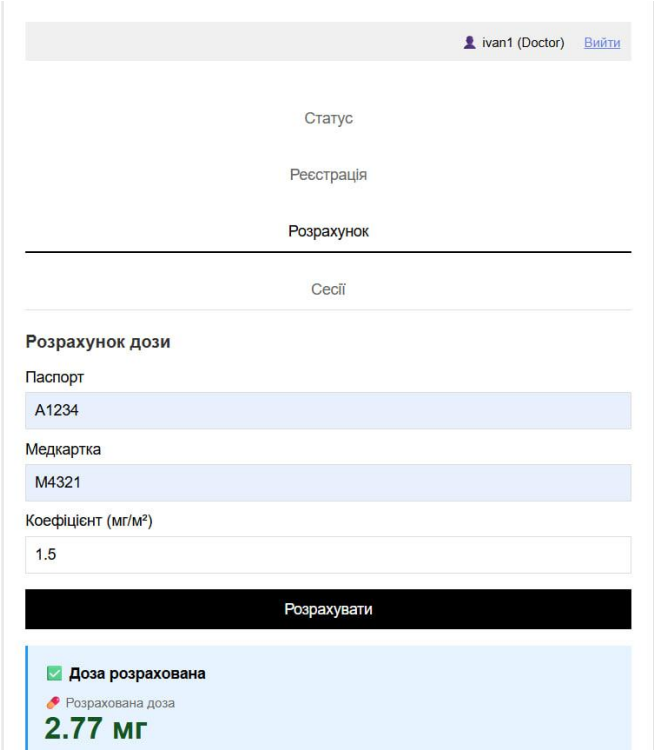
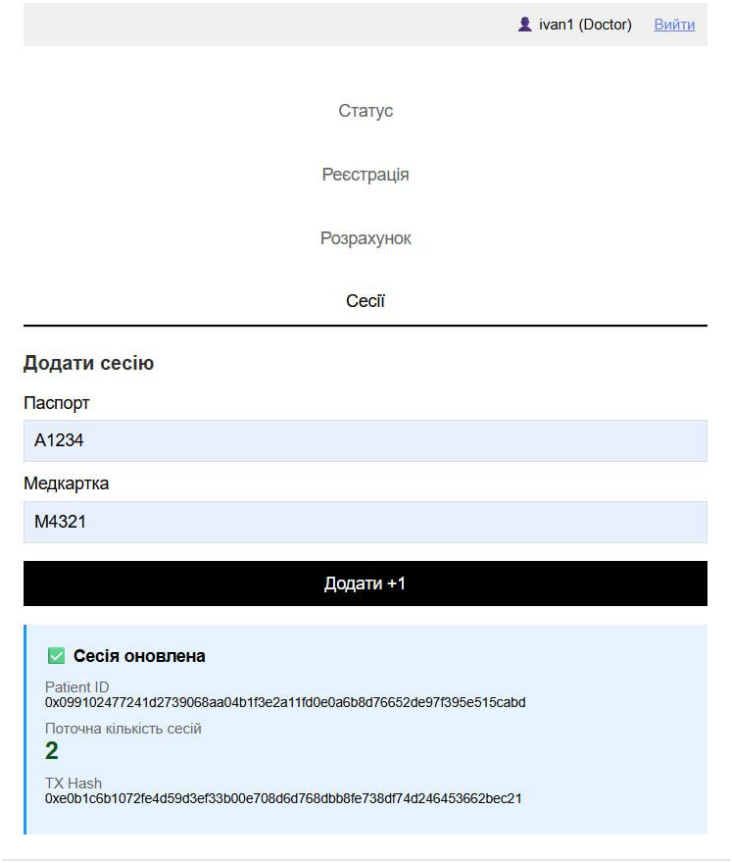


Рисунок 4.12 – Результат успішного розрахунку дозування

Наступним етапом можна перейти до оновлення кількості сесій хіміотерапії. Після натискання на розділ Сесії перед лікарем відображається аналогічна форма для введення № паспорта та № медичної картки. Після натискання на оновлення, викликається метод смарт-контракту, який відповідно оновлює сесії хіміотерапії згідно із кроками методу для процесу *update()*. Результат оновлення та відповідна транзакція наведені на рисунках 4.13, 4.14.



Риунок 4.13 – Вигляд успішного оновлення кількості сесій хіміотерапії

Рисунок 4.14 – Вигляд успішної транзакції в тестовому середовищі Ganache

Після того, як лікар виконав усі маніпуляції із даними, можна протестувати роль медсестри, а саме оновлення кількості сесій хіміотерапії.

Авторизація медсестри в системі відображена на рисунку 4.15.

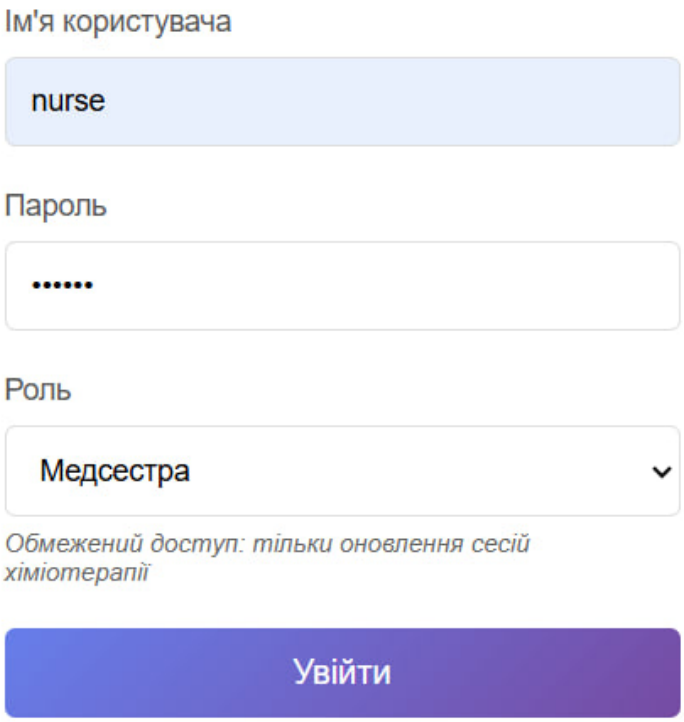


Рисунок 4.15 – Вигляд вікна авторизації для ролі медсестра

Після того, як медсестра успішно увійшла в систему, перед нею відображено можливість підключення до системи, а також оновлення сесій. Усі інші дії для неї недоступні. Таким чином, можна протестувати оновлення кількості сесій (рис. 4.16).

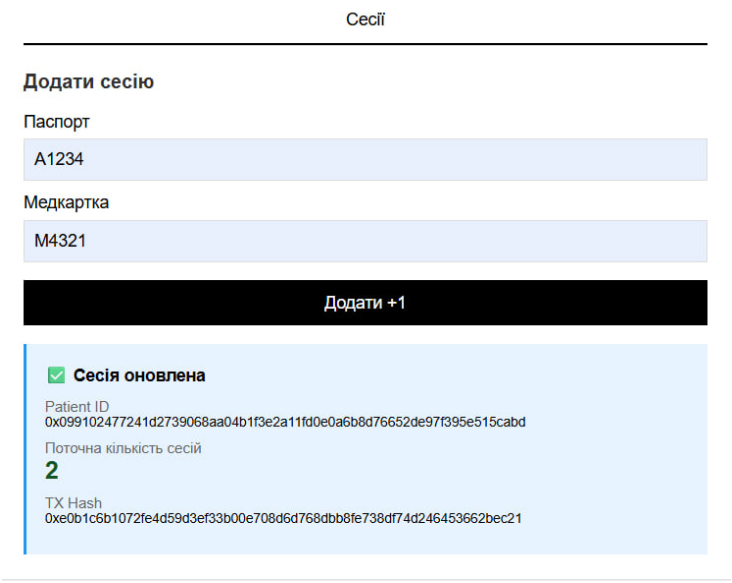


Рисунок 4.16 – Вигляд успішного оновлення кількості сесій

Таким чином, можна зробити висновок, що розроблений програмний засіб виконує усі поставлені задачі та відповідає вимогам методу.

Наступним етапом доцільно провести експериментальні дослідження витрат ресурсів в смарт-контрактах.

4.7 Висновки з розділу

Під час реалізації засобу необхідно звертати увагу на засоби розробки, таким чином, після аналізу було обрано мову програмування JavaScript, яка є скриптовою мовою та зручною для написання як клієнтської, так і серверної частин застосунку.

Для реалізації смарт-контрактів для блокчейну Ethereum було застосовано мову Solidity, яка забезпечує статичну типізацію, підтримує механізми успадкування та використання різних бібліотек.

Експериментальне дослідження схем гомоморфного шифрування дозволило зробити вибір схеми, яка буде використовуватись у процесі розробки, а саме схема Пальєра.

Статичне тестування безпеки необхідне для того, щоб перевірити чи код написаний коректно та чи не містить він суттєвих вразливостей, таким чином, за

результатами тестування смарт-контрактів за допомогою фреймворку Slither, суттєвих недоліків не було виявлено. До того ж було здійснено статичний аналіз усього проєкту, використавши платформу Aikido.

Динамічне тестування безпеки також не виявило недоліків у розробленому програмному засобі, тестування проводилось за допомогою OWASP ZAP.

Інтеграційне та модульне тестування показали, що розроблений засіб працює належним чином та має достатній функціонал.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Оскільки результати МКР на тему «Метод та засіб захисту медичних даних на основі гомоморфного шифрування» мають перспективу комерційного використання та можуть бути інтегровані у програмні рішення інших розробників чи компаній, виникає потреба у проведенні комерційного аудиту цієї науково-технічної розробки.

Основною метою такого аудиту є оцінювання науково-технічного потенціалу створеного програмного продукту, визначення його конкурентоспроможності та ринкових перспектив, а також формування економічного обґрунтування, яке може бути використане для залучення інвестицій і подальшої комерціалізації розробки.

У оцінюванні науково-технічного рівня і комерційного потенціалу розробки взяли участь три експерти: Войтович О. П., к.т.н., доц. кафедри захисту інформації, Лукічов В. В., к.т.н., доц. кафедри захисту інформації, Гарнага В. А., к.т.н., доц. кафедри захисту інформації.

Оцінки було проставлено згідно з рекомендованими критеріями, які зображені в додатку Г.

Результати оцінювання зображено в табл. 5.1.

Середньоарифметична сума балів розраховується за формулою:

$$\text{СБ} = \frac{\sum_{i=1}^{3} \text{СБ}_i}{3} \tag{5.1}$$

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Бали		
	Войтович О. П.	Лукічов В. В.	Гарнага В. А.
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	3	2
Ринкові переваги (ціна продукту)	4	4	4
Ринкові переваги (технічні властивості)	4	4	4
Ринкові переваги (експлуатаційні витрати)	3	4	4
Ринкові переваги (розмір ринку)	2	3	3
Ринкові переваги (конкуренція)	3	3	3
Практична здійсненність (наявність фахівців)	4	3	3
Практична здійсненність (наявність фінансів)	4	4	4
Практична здійсненність (необхідність нових матеріалів)	4	4	4
Практична здійсненність (термін реалізації)	4	4	4
Практична здійсненність (розробка документів)	4	4	4
Сума балів	41	43	42
Середньоарифметична сума балів, $СБ_c$	42		

Отже, за результатами аналізу (табл. 5.1) можна зробити висновок, що науково-технічний рівень та комерційний потенціал запропонованої розробки є високими. Такий показник було досягнуто завдяки значному підвищенню

точності медичних обчислень, безпеки обробки чутливих даних та покращенню процесу розрахунків доз хіміотерапевтичних препаратів.

Підвищення рівня захищеності забезпечується тим, що система дає змогу виконувати обчислення над зашифрованими медичними параметрами без їх розкриття, що суттєво зменшує ризики витоку інформації та знижує навантаження на інфраструктуру. Використання блокчейну гарантує незмінність даних і прозорість процесу доступу до них.

Підвищення рівня захищеності досягнуто завдяки усуненню ключових недоліків традиційних методів зберігання та обробки медичної інформації. Хоча запропонований підхід може потребувати додаткових обчислювальних ресурсів та зберігання допоміжних криптографічних даних, ці витрати є незначними й можуть бути ефективно оптимізовані за рахунок налаштування параметрів шифрування та інфраструктури.

За результатами розрахунків, наведених в таблиці 5.1, зроблено висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки.

При цьому доцільно використати рекомендації, наведені в табл. 5.2.

Таблиця 5.2 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вищий середнього
21…30	Середній
11…20	Нижчий середнього
0…10	Низький

Також доцільно провести аналіз рівня конкурентоспроможності запропонованої розробки.

Конкурентоспроможність визначається через сукупність якісних, технологічних та економічних показників, що характеризують безпечність та практичну цінність рішення у порівнянні з існуючими підходами.

Оцінювання рівня конкурентоспроможності науково-технічної розробки передбачає кілька послідовних етапів. У цьому випадку порівняльний аналіз буде здійснюватися відносно традиційних систем розрахунку доз хіміотерапевтичних препаратів, які працюють у звичайному режимі обробки, а саме з медичними даними, які представлені у відкритому вигляді.

Зокрема, порівняння проводитиметься зі стандартною моделлю використання медичних інформаційних систем, що використовують центральну базу даних для взаємодії із даними, але не забезпечують можливості виконання обчислень у зашифрованому вигляді та зберігання даних, забезпечуючи їм цілісність.

На відміну від таких рішень, запропонований підхід на основі частково гомоморфного шифрування та блокчейну дозволяє виконувати математичні операції над зашифрованими параметрами пацієнтів без їх розкриття, що істотно підвищує рівень конфіденційності та безпеки, а також мінімізує ризики несанкціонованого доступу чи модифікації даних.

5.1.1 Розрахунок одиничних параметритичних індексів

Якщо збільшення величини параметра свідчить про підвищення якості нової розробки, одиничний параметричний індекс розраховується за формулою:

$$q_i = \frac{P_i}{P_{\text{баз}i}}, \tag{5.2}$$

де q_i – одиничний параметричний індекс, розрахований за i-м параметром;

P_i – значення i-го параметра розробки;

$P_{\text{баз}i}$ – аналогічний параметр базової розробки-аналога, з якою проводиться порівняння.

Тобто для технічного показника, наприклад, продуктивності роботи, розрахунок буде такий:

$$q_1 = \frac{5}{4} = 1{,}25.$$

Для економічного показника розрахунок буде такий:

$$q_1 = \frac{5000}{7000} = 0{,}75.$$

Технічні та економічні параметри аналога та нової науково-технічної розробки доцільно подати у вигляді таблиці 5.3.

Таблиця 5.3 – Технічні та економічні параметри аналога нової науково-технічної розробки

Параметр	Одиниця виміру	Аналог	Нова розробка	Індекс зміни значення параметра	Коефіцієнт вагомості
Технічні					
Продуктивність	5-бальна шкала	4	5	1,25	0,25
Сумісність	5-бальна шкала	4	4	1	0,05
Функціональність	5-бальна шкала	5	5	1	0,05
Безпека	5-бальна шкала	5	5	1	0,25
Масштабованість	5-бальна шкала	4	5	1,25	0,4
Економічні					
Вартість інтеграції	грн	7000	5000	0,75	0,3
Вартість підтримки	грн	5000	4000	0,8	0,3
Час впровадження	години	10	8	0,8	0,4

Таким чином, можна перейти до розрахунку групових параметритичних індексів.

5.1.2 Розрахунок групових параметритичних індексів

Значення групового параметричного індексу за технічними параметрами визначається з урахуванням вагомості (частки) кожного параметра за формулою:

$$I_{\text{ТП}} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \qquad (5.3)$$

де $I_{\text{ТП}}$ – груповий параметричний індекс за технічними показниками (порівняно з аналогом);

q_i – одиничний параметричний показник *i*-го параметра;

α_i – вагомість *i*-го параметричного показника, $\sum_{i=1}^{n} \alpha_i = 1$;

n – кількість технічних параметрів, за якими оцінюється конкурентоспроможність.

Тобто розрахунок групового параметричного індексу за технічними параметрами матиме такий вигляд:

$$I_{\text{ТП}} = 1{,}25 \cdot 0{,}25 + 0{,}8 \cdot 0{,}05 + 1 \cdot 0{,}05 + 1 \cdot 0{,}25 + 1{,}25 \cdot 0{,}4 = 1{,}16$$

Груповий параметричний індекс за економічними параметрами (за ціною споживання) розраховується за формулою:

$$I_{\text{ЕП}} = \sum_{i=1}^{m} q_i \cdot \beta_i, \qquad (5.4)$$

де $I_{\text{ЕП}}$ – груповий параметричний індекс за економічними показниками;

q_i – економічний параметр *i*-го виду;

β_i – частка *i*-го економічного параметра, $\sum_{i=1}^{m} \beta_i = 1$;

m – кількість економічних параметрів, за якими здійснюється оцінювання.

Тобто розрахунок групового параметричного індексу за економічними параметрами матиме вигляд:

$$I_{\text{ЕП}} = 0{,}75 \cdot 0{,}3 + 0{,}8 \cdot 0{,}3 + 0{,}8 \cdot 0{,}4 = 0{,}785$$

Далі варто перейти до розрахунку інтегрального показника.

5.1.3 Розрахунок інтегрального показника

На основі групових параметричних індексів за нормативними, технічними та економічними показниками розраховують інтегральний показник конкурентоспроможності за формулою:

$$K_{\text{ІНТ}} = I_{\text{НП}} \cdot \frac{I_{\text{ТП}}}{I_{\text{ЕП}}}, \tag{5.5}$$

$$K_{\text{ІНТ}} = 1 \cdot \frac{1{,}16}{0{,}785} = 1{,}47.$$

На основі інтегрального показника $K_{\text{ІНТ}} = 1{,}47$ можна стверджувати, що розробка має високий рівень конкурентоспроможності та може бути успішно представлена на ринку. Таке значення показника пояснюється тим, що технологічні характеристики запропонованої системи перевищують показники традиційних рішень, а економічні витрати, навпаки, виявилися істотно нижчими. Саме оптимальне поєднання підвищених технічних параметрів та економічної ефективності забезпечило конкурентну перевагу розробки.

5.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати на здійснення науково-дослідної роботи групуються за такими статтями:

- витрати на оплату праці;
- відрахування на соціальні заходи;
- матеріали;
- паливо та енергія для науково-виробничих цілей;
- витрати на службові відрядження;
- спецустаткування для наукових (експериментальних) робіт;
- програмне забезпечення для наукових (експериментальних) робіт;
- витрати на роботи,які виконують сторонні підприємства, установи і організації;
- інші витрати;
- накладні (загальновиробничі) витрати.

Варто розглянути кожну статтю окремо.

5.2.1 Витрати на оплату праці

Для початку, варто розрахувати основну заробітну плату дослідників. Для цього необхідно використати формулу:

$$З_o = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.6)$$

де k – кількість посад дослідників, залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – кількість днів роботи конкретного дослідника, дн.;

T_p – середня кількість робочих днів в місяці, T_p = 22 дні.

Для зручності варто винести результати розрахунків у таблицю 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Кількість днів роботи	Витрати на заробітну плату, грн
Керівник проєкту	29800	1354,54	12	16254,48
Криптограф	25000	1136,36	10	11363,6
Розробник модуля взаємодії з блокчейном	22000	1000	17	17000
Тестувальник	19000	863,63	5	4318,15
Всього				48936,23

Основна заробітна плата робітників розраховується за формулою:

$$З_{\text{р}} = \sum_{i=1}^{n} C_i \cdot t_i, \quad (5.7)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{\text{ЗМ}}}, \quad (5.8)$$

де M_M – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн – наразі ММ = 8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середня кількість робочих днів в місяці, приблизно T_p = 22 дні;

$t_{\text{ЗМ}}$ – тривалість зміни, год.

$$C_i = \frac{8000 \cdot 1{,}1 \cdot 1{,}8}{22 \cdot 8} = 90 \text{ грн.}$$

$$З_{\text{р}} = 90 \cdot 14 = 1260 \text{ грн.}$$

Для зручності всі витрати варто винести в таблицю 5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Дослідження відомих засобів	14	2	1,1	90	1260
Розробка методу захисту медичних даних	25	7	2,2	180	4500

Розробка архітектури програмного засобу	4	3	1,35	110,45	441,8
Побудова алгоритмів роботи програмного засобу	8	3	1,35	110,45	883,63
Програмна реалізація засобу	50	6	2,0	163,63	8181,5
Оптимізація засобу	39	5	1,7	139,09	5424,51
Тестування засобу	40	3	1,35	101,25	4045
Експериментальне дослідження витрат ресурсів смарт-контрактів	17	6	2,0	163,63	2781,71
Виправлення виявлених помилок	18	3	1,35	110,45	1998,1
Збірка засобу	1	1	1,0	81,81	81,81
Всього					29598,06

Додаткова заробітна плата робітників розраховується за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.9)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати.

Нехай це буде 11%.

$$З_{дод} = (48936{,}23 + 29598{,}06) \cdot \frac{11\%}{100\%} = 8638{,}77.$$

Наступним етапом доцільно перейти до наступної статті – відрахування на соціальні заходи.

5.2.2 Відрахування на соціальні заходи

До статті «Відрахування на соціальні заходи» належать відрахування внеску на загальнообов’язкове державне соціальне страхування та для

здійснення заходів щодо соціального захисту населення (ЄСВ – єдиний соціальний внесок).

Нарахування на заробітну плату розраховується як 22% від суми основної та додаткової заробітної плати за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (5.10)$$

де $Н_{зп}$ – норма нарахування на заробітну плату, тобто $Н_{зп}$ = 22%.

$$З_н = (48936{,}23 + 29598{,}06 + 8638{,}77) \cdot \frac{22\%}{100\%} = 19178{,}07.$$

Далі варто розрахувати вартість сировини та матеріалів.

5.2.3 Сировина та матеріали

Всі процеси відбуваються у електронному форматі, працівники мають власні інструменти для досліджень та розробки, та все ж є необхідність у записниках.

Витрати на матеріали у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$M = \sum_{j-1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (5.11)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$– вартість матеріалу j-го найменування, грн/кг;

K_j– коефіцієнт транспортних витрат, (K_j= 1,1);

B_j – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Результати всіх розрахунків зображено у таблиці 5.6.

Найменування матеріалу, марка, тип, сорт	Ціна за 1 шт., грн.	Норма витрат, шт	Величина відходів, шт	Ціна відходів, грн/шт	Вартість витраченого матеріалу, грн
Блокнот формату А5, Kite	54	4	0	0	237,6
Папір офісний формату А4	12	2000	0	0	26400
USB-накопичувач	300	3	0	0	990
Ручка кулькова, чорна, Kite	22	4	0	0	96,8
Всього					27724,4

Наступним етапом варто розрахувати витрати на комплектуючі та спецустаткування для наукових робіт.

5.2.4 Розрахунок витрат на комплектуючі та спецустаткування для наукових робіт

Витрати на комплектуючі та спецустаткування для наукових робіт не передбачені, оскільки у них немає потреби.

5.2.5 Програмне забезпечення для наукових (експериментальних) робіт

Балансова вартість програмного забезпечення розраховується за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_j, \quad (5.12)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу цього виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_j – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (Кі = 1,10);

k – кількість найменувань програмних засобів.

Отримані результати винесені у таблицю 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
IDE VS Code Professional	1	1890	1890
Ganache	1	0	0
OWASP ZAP	1	0	0
Всього			1890

Наступним етапом варто розрахувати витрати на амортизацію обладнання, програмних засобів та приміщень.

5.2.6 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з викням прямолінійного методу амортизації за формулою (5.13).

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12}, \quad (5.13)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$Т_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Результати розрахунків варто винести в таблицю 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Кількість, шт	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Приміщення	1	120000	20	1	500,00
Ноутбук Asus	7	11000	2	1	1833,33
	Всього				2333,33

Отже, сума амортизації — 2333,33 грн.

5.2.7 Палива та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($В_е$) можна розрахувати за формулою:

$$В_e = \sum_{i=1}^{n} W_{yi} \cdot t_i \cdot Ц_e \cdot \frac{K_{вэ}}{\eta_i}, \quad (5.14)$$

де W_{yi} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн, $Ц_e$ = 10,5 грн;

$K_{вэ}$ – коефіцієнт, що враховує використання потужності, $K_{вэ} < 1$, $K_{вэ} = 0{,}38$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$, $\eta_i = 0{,}9$.

Результати проведених розрахунків занесено до таблиці 4.9.

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук Asus №1	0,025	40	10,28
Ноутбук Asus №2	0,04	56	23,03
Ноутбук Asus №3	0,045	120	55,52
Ноутбук Asus №4	0,050	40	20,56

Ноутбук Asus №5	0,025	30	7,71
Ноутбук Asus №6	0,025	20	5,14
Ноутбук Asus №7	0,04	26	10,69
Всього			132,93

Отже, загальна вартість становить 132,93 грн.

5.2.8 Службові відрядження

У службових відрядженнях немає потреби, тому кошти на них також не виділяються.

5.2.9 Витрати на роботи, які виконують сторонні підприємства, установи та організації

Зазначені витрати не передбачені.

5.2.10 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_{\text{в}} = (З_{\text{о}} + З_{\text{р}}) \cdot \frac{H_{\text{ів}}}{100\%}, \quad (5.15)$$

$$I_{\text{в}} = (27724{,}4 + 29598{,}06) \cdot \frac{50\%}{100\%} = 28661{,}23.$$

Отже, інші витрати становлять 28661,23 грн.

5.2.11 Накладні (загальновиробничі витрати)

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{зв}}{100\%}, \quad (5.16)$$

де $Н_{зв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = (27724{,}4 + 29598{,}06) \cdot \frac{100\%}{100\%} = 57322{,}46.$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}, \quad (5.17)$$

$$В_{заг} = 48936{,}23 + 27724{,}4 + 29598{,}06 + 8638{,}77 + 1890{,}00 + 2333{,}33 + 132{,}93 + 0{,}00 + 0{,}00 + 28661{,}23 + 57322{,}46 = 205237{,}41.$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.18)$$

де η – коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Оскільки наукова робота знаходиться на стадії розробки промислового зразка, то $\eta = 0{,}7$.

$$ЗВ = \frac{205237{,}41}{0{,}7} = 293196{,}3$$

Отже, загальновиробничі витрати становлять 293196,3 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки від її впровадження безспосередньо замовником

У цій МКР здійснено розробку спеціалізованого програмного забезпечення для забезпечення захищеного зберігання чутливих даних, що може бути впроваджене у виробничу та інформаційну інфраструктуру підприємства.

Можливе збільшення чистого прибутку підприємства ΔПі для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П_i = (\pm \Delta П_я \cdot N + П_я \cdot \Delta N_i), \quad (5.19)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження на підприємстві результатів науково-технічної розробки в аналізованому році;

N – основний кількісний показник, який визначає обсяг діяльності підприємства у році до впровадження результатів нової науково-технічної розробки;

$П_я$ – основний якісний показник, який визначає результати діяльності підприємства у кожному із років після впровадження науково-технічної розробки;

ΔN – зміна основного кількісного показника діяльності підприємства в результаті впровадження науково-технічної розробки в аналізованому році.

$$\Delta П_1 = 0{,}2 \cdot 800 + 2{,}5 \cdot 50 = 285 \text{ тис. грн.}$$

$$\Delta П_2 = \Delta П_3 = \Delta П_4 = \Delta П_5 = 0{,}12 \cdot 800 + 2{,}5 \cdot 50 = 223{,}4 \text{ тис. грн.}$$

Далі потрібно розрахувати приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати замовник від можливого впровадження науково-технічної розробки на власному підприємстві. Розрахунок відбувається за формулою:

$$\text{ПП} = \sum_{i=1}^{T} \frac{\Delta\text{П}_i}{(1+\tau)^t}, \tag{5.20}$$

де $\Delta\text{П}_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0{,}1$;

t – період часу (в роках) від моменту початку впровадження науковотехнічної розробки до моменту отримання підприємством збільшеної величини чистого прибутку в аналізованому році.

$$\text{ПП} = \frac{285}{(1+0{,}1)^1} + \frac{223{,}4}{(1+0{,}1)^2} + \frac{223{,}4}{(1+0{,}1)^3} + \frac{223{,}4}{(1+0{,}1)^4} + \frac{223{,}4}{(1+0{,}1)^5}$$
$$= 902{,}83 \text{ тис. грн.}$$

Далі потрібно розрахувати величину початкових інвестицій PV, які замовник має вкласти для здійснення науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{\text{розр}} \cdot \text{ЗВ}, \tag{5.21}$$

де розр k – коефіцієнт, що враховує витрати розробника (замовника) на впровадження науково-технічної розробки. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові

заходи тощо; зазвичай $k_{\text{розр}} = 2 \ldots 5$, але може бути і більшим. У випадку даного дослідження та розробки $k_{\text{розр}} = 2$;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 252454{,}73 = 504909{,}46 \text{ грн.}$$

Абсолютний економічний ефект $E_{\text{абс}}$ або чистий приведений дохід (NPV, Net Present Value) для розробника (замовника) від можливого впровадження науково-технічної розробки можна розрахувати за формулою:

$$E_{\text{абс}} = \text{ПП} - PV, \tag{5.22}$$

де ПП – приведена вартість збільшення всіх чистих прибутків від можливого впровадження науково-технічної розробки, грн;

PV – теперішня вартість початкових інвестицій, грн.

$$E_{\text{абс}} = 902{,}83 - 504{,}909 = 397{,}920 \text{ тис. грн.}$$

Для остаточного прийняття рішення необхідно розрахувати внутрішню економічну дохідність $E_{\text{в}}$ або показник внутрішньої норми дохідності (IRR, Internal Rate of Return) вкладених замовником коштів. Внутрішня економічна дохідність інвестицій $E_{\text{в}}$, які можуть бути вкладені замовником у впровадження науково-технічної розробки, розраховується за формулою:

$$E_{\text{в}} = \sqrt[\text{Тж}]{1 + \frac{E_{\text{абс}}}{PV}} - 1, \tag{5.23}$$

де $E_{\text{абс}}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, роки.

$$E_в = \sqrt[1]{1 + \frac{397,920}{504,909}} - 1 = 0,78.$$

Далі розраховується період окупності інвестицій $T_{ок}$ (DPP, Discounted Payback Period), які можуть бути вкладені розробником (замовником) у впровадження та комерціалізацію науково-технічної розробки. Розрахунок відбувається за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.24)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{0,78} = 1,28.$$

Оскільки $T_{ок} < 3$ років, можна зробити висновок, що впровадження науково-технічної розробки замовником є економічно ефективним.

5.4 Висновки з розділу

Отже, для повного дослідження економічної ефективності було проведено оцінювання науково-технічного рівня і комерційного потенціалу розробки трьома експертами за визначеними критеріями. Результат розрахунку середнього балу становить 42, що означає, науково технічний рівень та комерційний потенціал розробки є високим. Також було проведено аналіз рівня конкурентоспроможності розробки, який дозволив роз’яснити переваги відповідних технічних показників та економічних характеристик. Проведений

аналіз показав, що розробка є конкурентоспроможною та її можна виводити на ринок.

Крім того, було проведено розрахунок витрат на здійснення науково-дослідної роботи та розрахунок економічної ефективності науково-технічної розробки від її впровадження на підприємстві. Проведені розрахунки показали, що впровадження розробки є економічно ефективним.

ВИСНОВКИ

Аналіз нормативно правової бази показав актуальність захисту даних пацієнтів, особливо це важливо для критичних напрямів, зокрема лікування онкохворих, де процес захисту потребує уваги.

Захист даних пацієнтів регулюється суворими нормами урядів і привертає підвищену увагу з боку суспільства. Це пояснюється як критичністю медичної інфраструктури, так і великою кількістю особистих даних, що обробляються в цій сфері.

Відомі роботи з теми використання гомоморфного шифрування для захисту медичних даних застосовують його для великих масивів даних, що впливає на продуктивність через обчислювально важкі перетворення, які використовуються в процесі шифрування. Це негативно впливає на доступність даних і можливості серверів для одночасної обробки кількох запитів користувачів. Крім того, дані, представлені в зашифрованому вигляді для таких алгоритмів, значно більші за початкові до шифрування.

Формалізація задачі дозволила визначити параметри і обмеження задачі, зокрема набір даних, який має бути захищений за допомогою гомоморфного шифрування та основні операції обробки даних, які має враховувати метод захисту медичних даних. Це дозволило виконати узагальнене подання методу. В подальшому метод був адаптований до обраної в магістерській кваліфікаційній роботі області — захисту даних, які використовуються під час лікування саркоми Юінга. Аналіз методики лікування саркоми Юінга дозволив визначити, що частково гомоморфне шифрування, яке є менш вимогливим до обчислювальних ресурсів і має менший вплив на збільшення розміру даних після шифрування, є достатнім для цієї задачі, оскільки потреби в оновленні даних можна задовольнити лише за допомогою адитивних операцій, що робить застосування як схеми повністю гомоморфного, так і частково гомоморфного видів шифрування адекватними для використання в цій предметній області..

Ключовими особливостями запропонованого методу захисту є одночасне застосування гомоморфного шифрування і розподіленого сховища, а саме блокчейну типу Ethereum. Завдяки властивості гомоморфізму шифрування дані можна обробляти та оновлювати в зашифрованій формі в блокчейні, що дозволяє усунути недолік його відкритості і відсутності вбудованих механізмів захисту конфіденційності даних, що в ньому зберігаються. Водночас відкритість самого блокчейну та розподілення даних через численні вузли забезпечує підвищений захист цілісності, доступності та відстежуваності методу. Обчислені в роботі теоретичні оцінки складності зламу дозволили визначити ресурси, які потрібні для зламу методу і таким чином довести його стійкість.

Для реалізації методу було розроблено архітектуру засобу, до якої входить три основні частини: клієнтський застосунок, серверний застосунок та смарт-контракт, який виконується в блокчейні. Завдяки такому рішення вдалося уникнути єдиної точки відмови, притаманної більшості аналогів, що мають клієнт-серверну архітектуру, оскільки навіть при відмові серверної частини, користувачі можуть продовжувати звертатись до смарт-контракту. Останнє забезпечить виконання функціональних завдань засобу, хоча й з погіршеними характеристиками. Створена архітектура стала основою для декомпозиції задачі проєктування, внаслідок чого були розроблені алгоритми роботи окремих архітектурних складових.

Обґрунтування вибору засобів розроблення програми дозволило спростити процес реалізації. На основі експериментальних досліджень було обґрунтовано використання схеми Пальєра для реалізації гомоморфного шифрування. Для доведення коректності прийнятих технічних рішень в роботі було проведено модульне та інтеграційне тестування. Статичне та динамічне тестування безпеки показали відсутність критичних для безпеки вад в кодовій базі засобу.

Оцінювання показників економічної ефективності дозволило визначити перспективність та новизну запропонованого засобу. Крім того, були визначені

собівартість розробки та показники окупності, які показали економічну доцільність впровадження.

Подальший розвиток досліджень, виконаних в магістерській кваліфікацій роботі перспективно виконувати шляхом масштабування запропонованого методу для інших предметних областей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Про захист персональних даних: Закон України від 01.06.2010 р. № 2297-VI. Дата оновлення: 27.04.2024. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text (дата звернення: 01.09.2025).
2. General Data Protection Regulation (GDPR), *Official Journal of the European Union* L 119, 04.05.2016; with cor. L 127, 23.05.2018. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (accessed: 01.09.2025).
3. U.S. Department of Health and Human Services. Health Insurance Portability and Accountability Act (HIPAA). 1996. URL: https://www.hhs.gov/hipaa/index.html (дата звернення: 03.09.2025).
4. Baryshev Y., Lanova V.. Method of Patients` Data Protection on the Instance of Chemotherapy Dosing Data for Ewing`s Sarcoma Treatment. *Proceedings of the 7th International Conference on Informatics & Data-Driven Medicine* Birmingham, United Kingdom, November 14-16, 2024. Pp. 81-91. URL: https://ceur-ws.org/Vol-3892/ (accessed: 03.09.2025).
5. Баришев Ю. В., Ланова В. С. Інформаційна технологія захищеного зберігання результатів академічної успішності. *Інформаційні технології та комп`ютерна інженерія.* Т. 60. № 2. 2024. С. 17-30. DOI: 10.31649/1999-9941-2024-60-2-17-30 (дата звернення: 03.09.2025).
6. Баришев Ю. В., Ланова В. С. Метод захищеного зберігання медичних даних на основі реляційної бази даних та блокчейну. *Наукові праці ВНТУ.* № 3. 8 с. DOI: 10.31649/2307-5376-2023-3-9-17 (дата звернення: 03.09.2025).
7. Спосіб захисту інформації в мережах передавання даних: пат. 157759 Україна, МПК G06K 19/06. № u 2024 01843; заявл. 10.04.2024 ; опубл. 20.11.2024, Бюл. № 47. 5 с.
8. Lanova V., Klysh V., Baryshev Y. Method of applying homomorphic encryption for protecting private data. *Міжнародна конференція SMICS-2025 «Безпека*

сучасних інформаційно-комунікаційних систем». м. Львів, 16-18 жовтня 2025 року. URL: https://smics.lnu.edu.ua/uk/programa/ (accessed: 03.09.2025).

9. Lanova V., Baryshev Y. Comparative Analysis of Curves Performance of ElGamal Algorithm. *Міжнародна науково–практична конференція «Інформаційні технології та комп'ютерне моделювання»,* м. Івано-Франківськ, 2025, С. 169-170.URL: https://journal.comp-sc.if.ua/test/index.php/ITCM/article/view/648 (accessed: 03.09.2025).
10. Ланова В. С. , Баришев Ю. В.. Аналіз вразливостей кривих як криптографічних примітивів. *LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп`ютерної інженерії.* м. Вінниця, 2025. 3 с. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/23740/19611 (дата звернення: 03.09.2025).
11. Ланова В. С., Баришев Ю. В. Модуль видавання електронних направлень в медичних закладах. *Всеукраїнська науково-практична конференція «Theoretical and Applied Cybersecurity» (TACS-2024)*, м Київ, 30-31 травня. С. 133-136. URL: http://www.is.ipt.kpi.ua/pdf/T24.pdf (дата звернення: 03.09.2025).
12. Ланова В. С., Баришев Ю. В. Метод захищеного зберігання медичних даних за допомогою розмежування прав доступу та блокчейну. *The 13th International Scientific Conference «ITSec»,* м. Львів, 9-11 травня 2024 року. С. 115-116. (дата звернення: 03.09.2025).
13. Ланова В. С., Баришев Ю. В. Модель даних автоматизованої системи видавання електронних направлень на додаткові обстеження. *III Всеукраїнська науково-практична конференція з міжнародною участю. «Медико-технічна співпраця заради перемоги: актуальні завдання медичної, біологічної фізики та інформатики»,* м. Вінниця, 5-6 квітня 2024 року. С. 159-161.
14. Ланова В. С., Баришев Ю. В. Підхід до застосування IPFS для захисту медичних даних. *Інформаційні технології та комп'ютерне моделювання:*

матеріали статей Міжнародної науково-практичної конференції, м. Івано-Франківськ, 21-24 травня 2024 року. URL: https://journal.comp-sc.if.ua/test/index.php/ITCM/article/view/586 (дата звернення: 03.09.2025).

15. Ланова В. С., Баришев Ю. В. Аналіз геш-функцій для захисту цілісності чутливих даних. *Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (ВНТКП ВНТУ).* Вінниця, ВНТУ 2023. С. 501-503. URL: https://press.vntu.edu.ua/index.php/vntu/catalog/view/904/1576/2888-1 (дата звернення: 03.09.2025).
16. Ланова В. С., Баришев Ю. В. Смарт-контракти для розподіленого зберігання медичних даних. *LII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії.* Вінниця, ВНТУ 2023. URL: https://ir.lib.vntu.edu.ua/handle/123456789/39338 (дата звернення: 03.09.2025).
17. Baryshev Y., Lanova V. Database Structure for Medical Data Protection Based on Blockchain. *Інформаційні технології та комп'ютерне моделювання: матеріали статей Міжнародної науково-практичної конференції*, м. Івано-Франківськ, 15-16 грудня 2022 року. Івано-Франківськ: п. Голіней О.М., 2022. С. 90-91. URL: https://journal.comp-sc.if.ua/test/index.php/ITCM/article/view/275 (дата звернення: 03.09.2025).
18. Основи законодавства України про охорону здоров'я: Закон України від 19.11.1992 р. № 2801-XII. Дата оновлення: 19.04.2024. URL: https://zakon.rada.gov.ua/laws/show/2801-12 (дата звернення: 03.09.2025).
19. HIPAA vs Healthcare Laws and Regulations in Canada, the UK, Australia, and MENA Countries Source. URL: https://yalantis.com/blog/hipaa-vs-healthcare-laws-in-other-countries/ (accessed: 03.09.2025).
20. MedCo | Collective protection of medical data. URL: https://medco-ch.github.io/index.html (accessed: 03.09.2025).
21. e-Health Record – e-Estonia. URL: https://e-estonia.com/solutions/healthcare/e-health-records/ (accessed: 03.09.2025).

22. Katile U. MediChain: Medical Record Management System. *International Journal for Research in Applied Science and Engineering Technology*. 2023. Vol. 11, Pp. 3414 – 3417. DOI: 10.22214/ijraset.2023.52365 (accessed: 03.09.2025).
23. Sun J., Yao X., Wu Y. Blockchain-Based Secure Storage and Access Scheme For Electronic Medical Records in IPFS. *IEEE Access*. 2020. Vol. 8. Pp. 59389 – 59401. DOI: 10.1109/access.2020.2982964 (accessed: 03.09.2025).
24. Guo R., Shi H., Zhao Q. Secure Attribute-Based Signature Scheme With Multiple Authorities for Blockchain in Electronic Health Records Systems. *IEEE Access*. 2018. Vol. 6. Pp. 11676–11686. DOI: 10.1109/access.2018.2801266 (accessed: 10.09.2025).
25. Homomorphic encryption in healthcare. URL: https://www.inno-boost.com/blog/homomorphic-encryption-in-healthcare/ (accessed: 03.09.2025).
26. Adhikary S., Dutta S., Dwivedi A. D. Secret Learning for Lung Cancer Diagnosis - A Study with Homomorphic Encryption, Texture Analysis and Deep Learning. *Biomedical Physics & Engineering Express*. 2023. DOI: 10.1088/2057-1976/ad0b4b (accessed: 10.09.2025).
27. Diffie W., Hellman M. E. New Directions in Cryptography. *Secure Communications and Asymmetric Cryptosystems.* Routledge, 1976, Pp. 143–180 (accessed: 10.09.2025).
28. ElGamal T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 1985. Vol. 31(4), Pp. 469–472 (accessed: 12.09.2025).
29. Paillier P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, Pp. 223–238 (accessed: 18.09.2025).
30. Fan J., Vercauteren F. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*, 2012 (accessed: 18.09.2025).
31. Cheon J. H., Kim A., Kim M., et al. Homomorphic Encryption for Arithmetic of Approximate Numbers. *In: International Conference on the Theory and*

Application of Cryptology and Information Security. Springer, 2017, Pp. 409–437 (accessed: 18.09.2025).

32. Brakerski Z., Gentry C., Vaikuntanathan V. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 2014. Vol. 6(3), Pp. 1–36 (accessed: 19.09.2025).
33. Body Surface Area. URL: https://www.ncbi.nlm.nih.gov/books/NBK559005/ (accessed: 25.09.2025).
34. Kouno T. Standardization of the Body Surface Area (BSA) Formula to Calculate the Dose of Anticancer Agents in Japan. *Japanese Journal of Clinical Oncology*. 2003. Vol. 33, no. 6. Pp. 309–313. DOI:10.1093/jjco/hyg062 (accessed: 25.09.2025).
35. Mosteller R. Simplified Calculation of Body-Surface Area. *N Engl J Med*. 1987. Vol. 317(17):1098. DOI:10.1056/NEJM198710223171717 (accessed: 25.09.2025).
36. Ethereum whitepaper. URL: https://ethereum.org/uk/whitepaper/ (accessed: 28.09.2025).
37. Visual Studio Code. URL: https://code.visualstudio.com/docs (accessed: 28.09.2025).
38. IntelliJ IDEA Documentation. URL: https://www.jetbrains.com/help/idea/getting-started.html (accessed: 28.09.2025).
39. WebStorm. URL: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html (accessed: 28.09.2025).
40. Sublime Text. URL: https://www.sublimetext.com/docs/ (accessed: 25.09.2025).
41. Node.js. URL: https://nodejs.org/docs/latest/api/ (accessed: 28.09.2025).
42. Git. URL: https://git-scm.com/docs (accessed: 28.09.2025).
43. Solidity. URL: https://docs.soliditylang.org/en/v0.8.31/ (accessed: 28.09.2025).
44. Truffle. URL: https://archive.trufflesuite.com/docs/ (accessed: 28.09.2025).
45. Hardhat. URL: https://hardhat.org/docs/getting-started (accessed: 02.10.2025).
46. Brownie. URL: https://eth-brownie.readthedocs.io/en/stable/ (accessed: 02.10.2025).

47. Remix IDE. URL: https://remix-ide.readthedocs.io/en/latest/ (accessed: 02.10.2025).
48. Goerli. URL: https://www.alchemy.com/docs/wallets/reference/account-kit/core/variables/goerli (accessed: 05.10.2025).
49. Sepolia. URL: https://eth-sepolia.blockscout.com/api-docs (accessed: 05.10.2025).
50. Ganache. URL: https://archive.trufflesuite.com/docs/ganache/ (accessed: 07.10.2025).
51. Slither. URL: https://slither.readthedocs.io/en/latest/ (accessed: 15.10.2025).
52. Mythril. URL: https://mythril-classic.readthedocs.io/en/master/ (accessed: 15.10.2025).
53. Oyente. URL: https://lity.readthedocs.io/en/latest/oyente-integration.html (accessed: 15.10.2025).
54. Jest. URL: https://jestjs.io/docs/getting-started (accessed: 18.10.2025).
55. Aikido Security. URL: https://help.aikido.dev/ (accessed: 18.10.2025).
56. OWASP ZAP. URL: https://www.zaproxy.org/docs/ (accessed: 22.10.2025).

ДОДАТКИ

97

Додаток А. **ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ**

Назва роботи: Метод та засіб захисту медичних даних на основі гомоморфного шифрування

Автор роботи: Ланова Владислава Сергіївна

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ , група 1 БС-24м

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism **1, 32%**

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

√ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

В. о. зав. кафедри ЗІ д. т. н., проф. _____ Володимир ЛУЖЕЦЬКИЙ

Гарант освітньої програми «Безпека інформаційних і комунікаційних систем» к.т.н., доцент _____ Олеся ВОЙТОВИЧ

Особа, відповідальна за перевірку _____ Валентина КАПЛУН

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Юрій БАРИШЕВ

Здобувач _____ Владислава ЛАНОВА

Додаток Б. ТЕКСТ КЛІЄНТСЬКОЇ ЧАСТИНИ ПРОГРАМИ

index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Медична система</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
        }

        body {
            font-family: Arial, sans-serif;
            background: #f0f0f0;
            padding: 20px;
        }

        .container {
            max-width: 600px;
            margin: 0 auto;
            background: white;
            padding: 20px;
            border: 1px solid #ddd;
        }

        h1 {
            font-size: 20px;
            margin-bottom: 20px;
            color: #333;
        }

        .tabs {
            border-bottom: 1px solid #ddd;
            margin-bottom: 20px;
        }

        .tab {
            padding: 10px 15px;
            background: none;
            border: none;
            cursor: pointer;
            color: #666;
        }

        .tab.active {
            color: #000;
            border-bottom: 2px solid #000;
        }

        .tab-content {
            display: none;
        }

        .tab-content.active {
            display: block;
        }

        h2 {
            font-size: 16px;
            margin-bottom: 15px;
            color: #333;
        }
```

```
.status-list {
    margin-bottom: 20px;
}

.status-item {
    padding: 8px;
    margin-bottom: 5px;
    background: #f9f9f9;
    display: flex;
    justify-content: space-between;
}

.badge {
    padding: 2px 8px;
    font-size: 12px;
}

.badge-success {
    background: #d4edda;
    color: #155724;
}

.badge-error {
    background: #f8d7da;
    color: #721c24;
}

label {
    display: block;
    margin: 10px 0 5px 0;
    font-size: 14px;
}

input {
    width: 100%;
    padding: 8px;
    border: 1px solid #ddd;
    font-size: 14px;
}

input:focus {
    outline: none;
    border-color: #000;
}

.row {
    display: flex;
    gap: 10px;
}

.row > div {
    flex: 1;
}

button {
    width: 100%;
    padding: 10px;
    margin-top: 15px;
    background: #000;
    color: white;
    border: none;
    cursor: pointer;
    font-size: 14px;
}

button:hover:not(:disabled) {
    background: #333;
}

button:disabled {
```

```
            background: #999;
            cursor: not-allowed;
        }

        .info {
            padding: 10px;
            margin: 15px 0;
            background: #ffffcc;
            border-left: 3px solid #ffeb3b;
            font-size: 13px;
        }

        .result {
            margin-top: 15px;
            padding: 15px;
            background: #e7f3ff;
            border-left: 3px solid #2196f3;
        }

        .result h3 {
            font-size: 14px;
            margin-bottom: 10px;
        }

        .result-item {
            margin: 8px 0;
            font-size: 13px;
        }

        .result-label {
            color: #666;
            font-size: 12px;
        }

        .result-value {
            color: #000;
            word-break: break-all;
        }

        small {
            display: block;
            margin-top: 3px;
            color: #666;
            font-size: 12px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1></h1>

        <div class="tabs">
            <button class="tab active"
onclick="switchTab('status')">Статус</button>
            <button class="tab"
onclick="switchTab('register')">Реєстрація</button>
            <button class="tab"
onclick="switchTab('calculate')">Розрахунок</button>
            <button class="tab" onclick="switchTab('sessions')">Сесії</button>
        </div>

<div id="status" class="tab-content active">
    <h2>Статус системи</h2>

    <div class="status-list">
        <div class="status-item">
            <span>Блокчейн</span>
            <span id="blockchain-status" class="badge badge-
error">Відключено</span>
        </div>
        <div class="status-item">
```

```
            <span>Ключі</span>
            <span id="keys-status" class="badge badge-error">Немає</span>
        </div>
        <div class="status-item">
            <span>Контракт</span>
            <span id="contract-status" class="badge badge-error">Немає</span>
        </div>
    </div>

    <button onclick="initializeSystem()">Ініціалізувати</button>

    <button onclick="resetSystem()" style="background: #dc3545; margin-top:
10px;">
        Очистити ключі і перезапустити
    </button>

    <div class="info">
        <strong>Налаштування:</strong><br>
        RPC: http://127.0.0.1:7545<br>
        Шифрування: Paillier<br>
        Хеш: Keccak-256<br><br>
        <strong>Важливо:</strong> Ключі зберігаються в localStorage браузера
    </div>
</div>
        <!-- Реєстрація -->
        <div id="register" class="tab-content">
            <h2>Реєстрація пацієнта</h2>

            <form onsubmit="storePatientData(event)">
                <label>Паспорт</label>
                <input type="text" id="passport" required>

                <label>Медкартка</label>
                <input type="text" id="medicalCard" required>

                <div class="row">
                    <div>
                        <label>Зріст (см)</label>
                        <input type="number" id="height" required min="50"
max="300">
                    </div>
                    <div>
                        <label>Вага (кг)</label>
                        <input type="number" id="weight" required min="20"
max="300">
                    </div>
                </div>

                <label>Сесії</label>
                <input type="number" id="initialSessions" required value="0"
min="0" max="100">
                <small>Кількість проведених сесій</small>

                <button type="submit" id="store-btn">Зберегти</button>
            </form>

            <div id="store-result" style="display: none;"></div>
        </div>

        <!-- Розрахунок -->
        <div id="calculate" class="tab-content">
            <h2>Розрахунок дози</h2>

            <form onsubmit="calculateDose(event)">
                <label>Паспорт</label>
                <input type="text" id="calcPassport" required>

                <label>Медкартка</label>
                <input type="text" id="calcMedicalCard" required>

                <label>Коефіцієнт (мг/м²)</label>
```

```
                <input type="number" id="doseCoefficient" required value="50"
step="0.1">

                <button type="submit" id="calc-btn">Розрахувати</button>
            </form>

            <div id="calc-result" style="display: none;"></div>
        </div>

        <!-- Сесії -->
        <div id="sessions" class="tab-content">
            <h2>Додати сесію</h2>

            <form onsubmit="updateSessions(event)">
                <label>Паспорт</label>
                <input type="text" id="sessionPassport" required>

                <label>Медкартка</label>
                <input type="text" id="sessionMedicalCard" required>

                <button type="submit" id="session-btn">Додати +1</button>
            </form>

            <div id="session-result" style="display: none;"></div>
        </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/big-
integer@1.6.52/BigInteger.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/ethers@6.7.1/dist/ethers.umd.min.js"></script>
    <script>
        const authToken = localStorage.getItem('authToken');
        const userRole = localStorage.getItem('userRole');
        const username = localStorage.getItem('username');

        if (!authToken) {
            window.location.href = 'login.html';
        } else {
            console.log('Користувач:', username, 'Роль:', userRole);

            document.addEventListener('DOMContentLoaded', () => {
                const registerTab =
document.querySelector('.tab[onclick*="register"]');
                const calculateTab =
document.querySelector('.tab[onclick*="calculate"]');

                if (userRole === 'Nurse') {
                    if (registerTab) {
                        registerTab.style.display = 'none';
                    }
                    if (calculateTab) {
                        calculateTab.style.display = 'none';
                    }

                    switchTab('sessions');
                }

                if (userRole === 'Auditor') {
                    if (registerTab) registerTab.style.display = 'none';
                    if (calculateTab) calculateTab.style.display = 'none';
                    document.querySelector('.tab[onclick*="sessions"]').style.disp
lay = 'none';
                }

                const header = document.querySelector('.container');
                if (header) {
                    const userInfo = document.createElement('div');
                    userInfo.style.cssText = 'padding: 10px; background: #f0f0f0;
text-align: right; font-size: 12px;';
                    userInfo.innerHTML = `
```

```
                        ${username} (${userRole})
                        <a href="#" onclick="logout()" style="margin-left: 15px;
color: #667eea;">Вийти</a>
                    `;
                    header.insertBefore(userInfo, header.firstChild);
                }

                if (userRole === 'Nurse') {
                    const nurseInfo = document.getElementById('nurse-info');
                    if (nurseInfo) {
                        nurseInfo.style.display = 'block';
                    }
                }
            });
        }
    </script>
    <script src="js/paillier.js"></script>
    <script src="js/blockchain.js"></script>
    <script src="js/medical-system.js"></script>
    <script src="js/app.js"></script>
</body>
</html>
```

paillier.js

```
const STORAGE_KEY_PUBLIC = 'paillier_public_key_v1';
const STORAGE_KEY_PRIVATE = 'paillier_private_key_v1';

let globalPublicKey = null;
let globalPrivateKey = null;

function savePaillierKeysToStorage(publicKey, privateKey) {
    try {
        const publicData = {
            n: publicKey.n.toString(),
            _n2: publicKey._n2.toString(),
            g: publicKey.g.toString(),
            precision: publicKey.precision.toString()
        };

        const privateData = {
            lambda: privateKey.lambda.toString(),
            mu: privateKey.mu.toString()
        };

        localStorage.setItem(STORAGE_KEY_PUBLIC, JSON.stringify(publicData));
        localStorage.setItem(STORAGE_KEY_PRIVATE, JSON.stringify(privateData));

        console.log('Ключі збережено в localStorage');
        return true;
    } catch (error) {
        console.error('Помилка збереження ключів:', error);
        return false;
    }
}

function loadPaillierKeysFromStorage() {
    try {
        const publicRaw = localStorage.getItem(STORAGE_KEY_PUBLIC);
        const privateRaw = localStorage.getItem(STORAGE_KEY_PRIVATE);

        if (!publicRaw || !privateRaw) {
            console.log('Ключі не знайдено в localStorage');
            return false;
        }

        const publicData = JSON.parse(publicRaw);
        const privateData = JSON.parse(privateRaw);
```

```
        const n = bigInt(publicData.n);
        const g = bigInt(publicData.g);

        globalPublicKey = new PaillierPublicKey(n, g);
        globalPublicKey._n2 = bigInt(publicData._n2);
        globalPublicKey.precision = bigInt(publicData.precision);

        const lambda = bigInt(privateData.lambda);
        const mu = bigInt(privateData.mu);

        globalPrivateKey = new PaillierPrivateKey(lambda, mu, null, null,
globalPublicKey);

        console.log('Ключі завантажено з localStorage');
        console.log('Public key n:', globalPublicKey.n.toString().substring(0, 30)
+ '...');

        return true;
    } catch (error) {
        console.error('Помилка завантаження ключів:', error);
        localStorage.removeItem(STORAGE_KEY_PUBLIC);
        localStorage.removeItem(STORAGE_KEY_PRIVATE);
        return false;
    }
}

function clearPaillierKeys() {
    localStorage.removeItem(STORAGE_KEY_PUBLIC);
    localStorage.removeItem(STORAGE_KEY_PRIVATE);
    globalPublicKey = null;
    globalPrivateKey = null;
    console.log('Ключі видалено');
}
const seed = 12345;
bigInt.rand = (function(seed) {
    return function(bitLength) {
        seed = (seed * 16807) % 2147483647;
        return bigInt(seed).mod(bigInt(2).pow(bitLength));
    };
})(seed);

bigInt.prime = function (bitLength) {
    const start = Date.now();
    let attempts = 0;
    const maxAttempts = 1000;

    while (attempts < maxAttempts) {
        const candidate = bigInt.randBetween(
            bigInt(2).pow(bitLength - 1),
            bigInt(2).pow(bitLength).subtract(1)
        ).or(1);

        attempts++;

        if (candidate.isProbablePrime(15)) {
            return candidate;
        }
    }
    throw new Error(`Failed to generate prime after ${maxAttempts} attempts`);
};

function L(u, n) {
    return u.subtract(1).divide(n);
}

const generateRandomKeys = function (bitLength = 128) {
    console.log(`Генерація ключів ${bitLength} біт...`);

    let p = bigInt.prime(bitLength / 2);
    let q = bigInt.prime(bitLength / 2);
```

```
        while (p.equals(q)) {
            q = bigInt.prime(bitLength / 2);
        }

        const n = p.multiply(q);
        const n2 = n.pow(2);
        const g = n.add(1);
        const lambda = bigInt.lcm(p.subtract(1), q.subtract(1));

        const gLambda = g.modPow(lambda, n2);
        console.log('g^lambda mod n²:', gLambda.toString().substring(0, 40) + '...');

        const L_gLambda = gLambda.subtract(1).divide(n);
        console.log('L(g^lambda):', L_gLambda.toString().substring(0, 40) + '...');
        console.log('n:', n.toString().substring(0, 40) + '...');

        const gcd = bigInt.gcd(L_gLambda, n);
        console.log('gcd(L(g^lambda), n):', gcd.toString());

        if (!gcd.equals(1)) {
            console.warn('L(g^lambda) та n не взаємно прості! Регенеруємо ключі...');
            return generateRandomKeys(bitLength);
        }

        const mu = L_gLambda.modInv(n);
        console.log('mu:', mu.toString().substring(0, 40) + '...');

        const check = L_gLambda.multiply(mu).mod(n);
        console.log('Перевірка (L * mu) mod n:', check.toString());

        if (!check.equals(1)) {
            console.error('mu обчислено неправильно! Регенеруємо...');
            return generateRandomKeys(bitLength);
        }

        const maxUint256 = bigInt(2).pow(256).subtract(1);
        if (n2.greater(maxUint256)) {
            return generateRandomKeys(bitLength - 16);
        }

        const publicKey = new PaillierPublicKey(n, g);
        const privateKey = new PaillierPrivateKey(lambda, mu, p, q, publicKey);

        console.log('\nТестування...');
        const testVal = 1;
        const enc = publicKey.encrypt(testVal);
        const dec = privateKey.decrypt(enc);
        console.log(`Тест: ${testVal} -> ${dec}`);

        if (Math.abs(dec - testVal) > 0.01) {
            console.error('Тест провалено! Регенеруємо ключі...');
            return generateRandomKeys(bitLength);
        }

        console.log('Ключі працюють!\n');
        return { publicKey, privateKey };
    };

    class PaillierPublicKey {
        constructor(n, g) {
            this.n = bigInt(n);
            this._n2 = this.n.pow(2);
            this.g = bigInt(g);
            this.precision = bigInt(1); // Було 1e6, стало 1
        }

        encrypt(m) {
            const scaledM = this._scaleUp(m);

            console.log('=== ENCRYPT DEBUG ===');
            console.log('Original value:', m);
```

```
        console.log('Scaled value:', scaledM.toString());

        let r;
        do {
            r = bigInt.randBetween(1, this.n);
        } while (r.leq(1) || r.geq(this.n));

        const result = this.g.modPow(scaledM, this._n2)
            .multiply(r.modPow(this.n, this._n2))
            .mod(this._n2);

        console.log('Encrypted result:', result.toString().substring(0, 30) +
'...');

        return result;
    }

    add(c1, c2) {
        return bigInt(c1).multiply(bigInt(c2)).mod(this._n2);
    }

    multiply(c, k) {
        console.log('=== MULTIPLY DEBUG ===');
        console.log('Encrypted value c:', c.toString().substring(0, 30) + '...');
        console.log('Scalar k:', k);

        const result = bigInt(c).modPow(bigInt(k), this._n2);

        console.log('Result:', result.toString().substring(0, 30) + '...');

        return result;
    }

    _scaleUp(value) {
        if (typeof value === 'number') {
            return bigInt(Math.round(value * this.precision.toJSNumber()));
        }
        if (typeof value === 'string') {
            return bigInt(value).multiply(this.precision);
        }
        if (bigInt.isInstance(value)) {
            return value.multiply(this.precision);
        }
        throw new Error(`Unsupported type for scaling: ${typeof value}`);
    }
}

class PaillierPrivateKey {
    constructor(lambda, mu, p, q, publicKey) {
        this.lambda = this._toBigInt(lambda);
        this.mu = this._toBigInt(mu);
        this._p = this._toBigInt(p);
        this._q = this._toBigInt(q);
        this.publicKey = publicKey;
    }

    _toBigInt(value) {
        const valueStr = value.toString();
        if (!/^-?\d+$/.test(valueStr)) {
            throw new RangeError(`Invalid value for BigInt: ${value}`);
        }
        return bigInt(valueStr);
    }

    decrypt(c) {
        c = this._toBigInt(c);

        // m = L(c^lambda mod n²) * mu mod n
        const cLambda = this.modPow(c, this.lambda, this.publicKey._n2);
        const L_cLambda = cLambda.subtract(bigInt(1)).divide(this.publicKey.n);
        const m = L_cLambda.multiply(this.mu).mod(this.publicKey.n);
```

```
        console.log('=== DECRYPT DEBUG ===');
        console.log('c^lambda mod n²:', cLambda.toString().substring(0, 30) +
'...');
        console.log('L(c^lambda):', L_cLambda.toString().substring(0, 30) +
'...');
        console.log('m (raw):', m.toString());
        console.log('m / precision:',
m.divide(this.publicKey.precision).toString());

        return this._scaleDown(m);
    }

    _scaleDown(value) {
        const result = value.divide(this.publicKey.precision);
        console.log('_scaleDown result:', result.toString());
        return result.toJSNumber();
    }

    decryptInteger(c) {
        c = this._toBigInt(c);
        const x = this.modPow(c, this.lambda, this.publicKey._n2);
        const L = x.subtract(bigInt(1)).divide(this.publicKey.n);
        const m = L.multiply(this.mu).mod(this.publicKey.n);

        const result = m.divide(this.publicKey.precision);
        return parseInt(result.toString());
    }

    modPow(base, exponent, modulus) {
        base = this._toBigInt(base);
        exponent = this._toBigInt(exponent);
        modulus = this._toBigInt(modulus);

        if (modulus.eq(bigInt(1))) return bigInt(0);
        let result = bigInt(1);
        base = base.mod(modulus);

        while (exponent.greater(bigInt(0))) {
            if (exponent.and(bigInt(1)).eq(bigInt(1))) {
                result = result.multiply(base).mod(modulus);
            }
            exponent = exponent.shiftRight(1);
            base = base.square().mod(modulus);
        }
        return result;
    }
}

function L(u, n) {
    return u.subtract(1).divide(n);
}
```

Додаток В. ТЕКСТ СЕРВЕРНОЇ ЧАСТИНИ ПРОГРАМИ

server.js

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const bcrypt = require('bcryptjs');
const path = require('path');
const odbc = require('odbc');
const { requireAuth, requireRole, generateToken } = require('./routes/auth.js');

const app = express();
const PORT = process.env.PORT || 8081;

app.use(cors({
    origin: [
        'http://127.0.0.1:8080',
        'http://localhost:8080',
        'http://127.0.0.1:8081',
        'http://localhost:8081'
    ],
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization']
}));

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', req.headers.origin || '*');
    res.header('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS');
    res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization');
    res.header('Access-Control-Allow-Credentials', 'true');

    if (req.method === 'OPTIONS') {
        return res.sendStatus(200);
    }
    next();
});

const connectionString = `
Driver={ODBC Driver 17 for SQL Server};
Server=DESKTOP-N103H81\\SQLEXPRESS;
Database=Medical_Blockchain;
Trusted_Connection=Yes;
`;

async function getConnection() {
    try {
        const connection = await odbc.connect(connectionString);
        console.log("Підключено до бази даних!");
        return connection;
    } catch (err) {
        console.error("Помилка підключення до БД:", err);
        throw err;
    }
}
app.get('/api', (req, res) => {
    res.json({ message: 'API працює!' });
});

app.get('/api/test', async (req, res) => {
    try {
        const db = await getConnection();
        const result = await db.query('SELECT @@VERSION AS version');
```

```
        await db.close();
        res.json({ success: true, data: result });
    } catch (err) {
        res.status(500).json({ success: false, error: err.message });
    }
});

// Реєстрація
app.post('/api/auth/register', async (req, res) => {
    try {
        const { fullName, username, email, password, role } = req.body;

        if (!fullName || !username || !email || !password) {
            return res.status(400).json({
                success: false,
                error: 'Всі поля обов\'язкові'
            });
        }

        const db = await getConnection();

        const existingUser = await db.query(
            `SELECT * FROM Users WHERE Username = ? OR Email = ?`,
            [username, email]
        );

        if (existingUser.length > 0) {
            await db.close();
            return res.status(400).json({
                success: false,
                error: 'Користувач з таким логіном або email вже існує'
            });
        }

        const hashedPassword = await bcrypt.hash(password, 10);

        await db.query(`
            INSERT INTO Users (Username, PasswordHash, Email, FullName, Role,
CreatedAt)
            VALUES (?, ?, ?, ?, ?, GETDATE())
        `, [username, hashedPassword, email, fullName, role || 'Doctor']);

        const result = await db.query(
            'SELECT * FROM Users WHERE Username = ?',
            [username]
        );

        await db.close();

        const user = result[0];

        res.json({
            success: true,
            message: 'Користувач успішно зареєстрований',
            user: {
                id: user.UserID,
                username: user.Username,
                email: user.Email,
                role: user.Role
            }
        });
    } catch (error) {
        console.error('Register error:', error);
        res.status(500).json({
            success: false,
            error: 'Помилка сервера при реєстрації'
        });
    }
});

// Вхід
```

```
app.post('/api/auth/login', async (req, res) => {
    try {
        const { username, password } = req.body;

        if (!username || !password) {
            return res.status(400).json({
                success: false,
                error: 'Логін та пароль обов\'язкові'
            });
        }

        const db = await getConnection();

        const result = await db.query(
            'SELECT * FROM Users WHERE Username = ?',
            [username]
        );

        if (result.length === 0) {
            await db.close();
            return res.status(401).json({
                success: false,
                error: 'Невірний логін або пароль'
            });
        }

        const user = result[0];
        const isValid = await bcrypt.compare(password, user.PasswordHash);

        if (!isValid) {
            await db.close();
            return res.status(401).json({
                success: false,
                error: 'Невірний логін або пароль'
            });
        }

        await db.query(
            'UPDATE Users SET LastLogin = GETDATE() WHERE UserID = ?',
            [user.UserID]
        );

        await db.close();

        const token = generateToken(user);

        res.json({
            success: true,
            token,
            user: {
                id: user.UserID,
                username: user.Username,
                email: user.Email,
                fullName: user.FullName,
                role: user.Role,
                ethereumAddress: user.EthereumAddress
            }
        });
    } catch (error) {
        console.error('Login error:', error);
        res.status(500).json({
            success: false,
            error: 'Помилка сервера при вході'
        });
    }
});

app.get('/api/auth/verify', requireAuth, (req, res) => {
    res.json({
        success: true,
        user: req.user
```

```
    });
});

// Створення пацієнта
app.post('/api/patients', requireAuth, async (req, res) => {
    try {
        const { passport, medicalCard, heightEncrypted, weightEncrypted,
                sessionsEncrypted, patientID, txHash, blockNumber } = req.body;

        console.log('=== РЕЄСТРАЦІЯ ПАЦІЄНТА В БД ===');
        console.log('PatientID:', patientID);
        console.log('Passport:', passport);
        console.log('MedicalCard:', medicalCard);

        if (!patientID || !passport || !medicalCard || !heightEncrypted ||
!weightEncrypted || !sessionsEncrypted) {
            return res.status(400).json({
                success: false,
                error: 'Всі поля обов\'язкові'
            });
        }

        const db = await getConnection();

        const existing = await db.query(
            'SELECT PatientID FROM Patients WHERE PatientID = ?',
            [patientID]
        );

        if (existing.length > 0) {
            await db.close();
            return res.status(400).json({
                success: false,
                error: 'Пацієнт з таким ID вже існує'
            });
        }

        await db.query(`
            INSERT INTO Patients
            (PatientID, Passport, MedicalCard, HeightEncrypted, WeightEncrypted,
             ChemotherapySessionsEncrypted, TxHash, BlockNumber, CreatedBy,
CreatedAt)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, GETDATE())
        `, [patientID, passport, medicalCard, heightEncrypted, weightEncrypted,
            sessionsEncrypted, txHash || null, blockNumber || 0, req.user.id]);

        console.log('INSERT виконано');

        const result = await db.query(
            'SELECT * FROM Patients WHERE PatientID = ?',
            [patientID]
        );

        await db.close();

        console.log('Пацієнт збережено:', result[0]);

        res.json({
            success: true,
            patient: result[0]
        });
    } catch (error) {
        console.error('Store patient error:', error);
        console.error('ODBC Errors:', error.odbcErrors);
        res.status(500).json({
            success: false,
            error: error.message
        });
    }
});
```

```
app.put('/api/patients/:patientID', requireAuth, async (req, res) => {
    try {
        const { patientID } = req.params;
        const { txHash, blockNumber } = req.body;

        const db = await getConnection();

        await db.query(`
            UPDATE Patients
            SET TxHash = ?,
                BlockNumber = ?,
                UpdatedAt = GETDATE()
            WHERE PatientID = ?
        `, [txHash, blockNumber || 0, patientID]);

        const result = await db.query(
            'SELECT * FROM Patients WHERE PatientID = ?',
            [patientID]
        );

        await db.close();

        if (result.length === 0) {
            return res.status(404).json({
                success: false,
                error: 'Пацієнт не знайдений'
            });
        }

        res.json({
            success: true,
            patient: result[0]
        });
    } catch (error) {
        console.error('Update patient error:', error);
        res.status(500).json({
            success: false,
            error: error.message
        });
    }
});

// Збереження розрахованої дози
app.post('/api/patients/:patientID/dose', requireAuth, async (req, res) => {
    try {
        const { patientID } = req.params;
        const { doseEncrypted, doseDecrypted, txHash, doseCoefficient } =
req.body;

        console.log('=== ЗБЕРЕЖЕННЯ ДОЗИ ===');
        console.log('PatientID:', patientID);
        console.log('DoseDecrypted:', doseDecrypted);
        console.log('DoseCoefficient:', doseCoefficient);

        if (isNaN(parseFloat(doseDecrypted))) {
            return res.status(400).json({
                success: false,
                error: 'Invalid DoseDecrypted value'
            });
        }

        const db = await getConnection();

        const patientCheck = await db.query(
            'SELECT PatientID FROM Patients WHERE PatientID = ?',
            [patientID]
        );

        if (patientCheck.length === 0) {
            await db.close();
            return res.status(404).json({
```

```
                success: false,
                error: 'Пацієнт не знайдений в БД'
            });
        }
        await db.query(`
            INSERT INTO DoseCalculations
            (PatientID, DoseEncrypted, DoseDecrypted, DoseCoefficient,
             TxHash, CalculatedBy, CalculatedAt)
            VALUES (?, ?, ?, ?, ?, ?, GETDATE())
        `, [
            patientID,
            doseEncrypted || null,
            parseFloat(doseDecrypted),
            parseFloat(doseCoefficient),
            txHash || null,
            req.user.id
        ]);

        const result = await db.query(`
            SELECT TOP 1 * FROM DoseCalculations
            WHERE PatientID = ?
            ORDER BY CalculatedAt DESC
        `, [patientID]);

        await db.close();

        console.log('Доза збережена:', result[0]);

        res.json({
            success: true,
            dose: result[0]
        });

    } catch (error) {
        console.error('Помилка при збереженні дози:', error);
        res.status(500).json({
            success: false,
            error: error.message
        });
    }
});

// Оновлення сесій
app.put('/api/patients/:patientID/sessions', requireAuth, async (req, res) => {
    try {
        const { patientID } = req.params;
        const { sessionsEncrypted, txHash } = req.body;

        const db = await getConnection();

        await db.query(`
            UPDATE Patients
            SET ChemotherapySessionsEncrypted = ?,
                UpdatedAt = GETDATE()
            WHERE PatientID = ?
        `, [sessionsEncrypted, patientID]);

        const result = await db.query(
            'SELECT * FROM Patients WHERE PatientID = ?',
            [patientID]
        );

        await db.query(`
            INSERT INTO AuditLog (UserID, Action, Details, Timestamp)
            VALUES (?, ?, ?, GETDATE())
        `, [req.user.id, 'UPDATE_SESSIONS', JSON.stringify({ patientID, txHash
})]);

        await db.close();

        if (result.length === 0) {
```

```
            return res.status(404).json({
                success: false,
                error: 'Пацієнт не знайдений'
            });
        }

        res.json({
            success: true,
            patient: result[0]
        });
    } catch (error) {
        console.error('Update sessions error:', error);
        res.status(500).json({
            success: false,
            error: error.message
        });
    }
});

app.use(express.static(path.join(__dirname, 'public')));

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.get('/login', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'login.html'));
});

app.get('/register', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'register.html'));
});

app.listen(PORT, async () => {
    console.log(`Сервер запущено на http://127.0.0.1:${PORT}`);
    console.log(`API доступне на http://127.0.0.1:${PORT}/api`);

    try {
        await getConnection();
    } catch (err) {
        console.error('Не вдалося підключитися до БД');
    }
});

process.on('SIGINT', async () => {
    console.log('\nЗупинка сервера...');
    process.exit(0);
});
```

Додаток Г. ТЕКСТ СМАРТ-КОНТРАКТУ

MedicalStorage.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

library BigInt2048 {
    uint256 internal constant K = 8;
    uint256 internal constant K2 = 16;

    function fullMul(uint256 a, uint256 b) internal pure returns (uint256 low,
uint256 high) {
        unchecked {
            uint256 mm = mulmod(a, b, type(uint256).max);
            low = a * b;
            high = mm - low;
            if (mm < low) high -= 1;
        }
    }

    function addWithCarry(uint256 a, uint256 b, uint256 carryIn) internal pure
returns (uint256 sum, uint256 carryOut) {
        unchecked {
            uint256 s = a + b + carryIn;
            if (s < a || (carryIn > 0 && s == a)) carryOut = 1;
            else carryOut = 0;
            sum = s;
        }
    }

    function subWithBorrow(uint256 a, uint256 b, uint256 borrowIn) internal pure
returns (uint256 res, uint256 borrowOut) {
        unchecked {
            uint256 tmp = a - b;
            uint256 r = tmp - borrowIn;
            if (a < b || tmp < borrowIn) borrowOut = 1;
            else borrowOut = 0;
            res = r;
        }
    }

    function mul2048(uint256[K] memory A, uint256[K] memory B) internal pure
returns (uint256[K2] memory P) {
        for (uint i = 0; i < K; i++) {
            uint256 carry = 0;
            for (uint j = 0; j < K; j++) {
                (uint256 low, uint256 high) = fullMul(A[i], B[j]);
                uint idx = i + j;

                (uint256 s1, uint256 c1) = addWithCarry(P[idx], low, 0);
                (uint256 s2, uint256 c2) = addWithCarry(s1, carry, 0);
                P[idx] = s2;
                carry = high + c1 + c2;
            }
            uint idx2 = i + K;
            while (carry != 0) {
                (uint256 s3, uint256 c3) = addWithCarry(P[idx2], carry, 0);
                P[idx2] = s3;
                carry = c3;
                idx2++;
            }
        }
    }

    function lt(uint256[K] memory a, uint256[K] memory b) internal pure returns
(bool) {
        for (uint i = K; i > 0; i--) {
```

```
            uint idx = i - 1;
            if (a[idx] < b[idx]) return true;
            if (a[idx] > b[idx]) return false;
        }
        return false;
    }

    function sub(uint256[K] memory a, uint256[K] memory b) internal pure returns
(uint256[K] memory res) {
        uint256 borrow = 0;
        for (uint i = 0; i < K; i++) {
            (uint256 d, uint256 bo) = subWithBorrow(a[i], b[i], borrow);
            res[i] = d;
            borrow = bo;
        }
    }

    function barrettLikeReduce(uint256[K2] memory x, uint256[K] memory m) internal
pure returns (uint256[K] memory r) {
        for (uint i = 0; i < K; i++) r[i] = x[i];
        while (!lt(r, m)) {
            r = sub(r, m);
        }
    }

    function mulMod(uint256[K] memory a, uint256[K] memory b, uint256[K] memory m)
internal pure returns (uint256[K] memory) {
        uint256[K2] memory prod = mul2048(a, b);
        return barrettLikeReduce(prod, m);
    }

    function copyK(uint256[K] memory src) internal pure returns (uint256[K] memory
dst) {
        for (uint i = 0; i < K; i++) dst[i] = src[i];
    }

    function one() internal pure returns (uint256[K] memory o) {
        o[0] = 1;
        for (uint i = 1; i < K; i++) o[i] = 0;
    }
}

contract MedicalDataStorageBig {
    using BigInt2048 for *;

    uint256 internal constant K = 8;
    uint256 internal constant K2 = 16;

    struct PatientDataStorage {
        uint256[K2] heightEncrypted;
        uint256[K2] weightEncrypted;
        uint256[K2] chemotherapySessionsEncrypted;
        bool exists;
        uint256 timestamp;
    }

    uint256[K] public n;
    uint256[K] public nSquared;
    uint256[K2] public n2_raw;

    mapping(bytes32 => PatientDataStorage) private patients;

    address public owner;

    event PaillierParamsSet();
    event PatientDataStored(bytes32 indexed patientID);
    event ChemotherapySessionUpdated(bytes32 indexed patientID);
    event DoseEncryptedReturned(bytes32 indexed patientID);

    modifier onlyOwner() {
        require(msg.sender == owner, "only owner");
```

```
        _;
    }

    constructor() {
        owner = msg.sender;
    }

    function setPaillierParams(uint256[K] calldata _n, uint256[K2] calldata
_n2_raw) external onlyOwner {
        for (uint i = 0; i < K; i++) {
            n[i] = _n[i];
        }
        for (uint i = 0; i < K2; i++) {
            n2_raw[i] = _n2_raw[i];
        }
        for (uint i = 0; i < K; i++) {
            nSquared[i] = _n2_raw[i];
        }

        emit PaillierParamsSet();
    }

    function storePatientData(
        bytes32 patientID,
        uint256[K2] calldata heightEncrypted,
        uint256[K2] calldata weightEncrypted,
        uint256[K2] calldata chemotherapySessionsEncrypted
    ) external {
        require(!patients[patientID].exists, "Patient already exists");

        for (uint i = 0; i < K2; i++) patients[patientID].heightEncrypted[i] =
heightEncrypted[i];
        for (uint i = 0; i < K2; i++) patients[patientID].weightEncrypted[i] =
weightEncrypted[i];
        for (uint i = 0; i < K2; i++)
patients[patientID].chemotherapySessionsEncrypted[i] =
chemotherapySessionsEncrypted[i];

        patients[patientID].exists = true;
        patients[patientID].timestamp = block.timestamp;

        emit PatientDataStored(patientID);
    }

    function retrievePatientData(bytes32 patientID)
        external
        view
        returns (uint256[K2] memory heightEncrypted, uint256[K2] memory
weightEncrypted, uint256[K2] memory chemotherapySessionsEncrypted, uint256
timestamp)
    {
        require(patients[patientID].exists, "Patient not found");
        PatientDataStorage storage p = patients[patientID];
        for (uint i = 0; i < K2; i++) heightEncrypted[i] = p.heightEncrypted[i];
        for (uint i = 0; i < K2; i++) weightEncrypted[i] = p.weightEncrypted[i];
        for (uint i = 0; i < K2; i++) chemotherapySessionsEncrypted[i] =
p.chemotherapySessionsEncrypted[i];
        timestamp = p.timestamp;
    }

    function paillierAdd(uint256[K2] calldata c1_raw, uint256[K2] calldata c2_raw)
public view returns (uint256[K2] memory) {

        uint256[K] memory a;
        uint256[K] memory b;
        for (uint i = 0; i < K; i++) {
            a[i] = c1_raw[i];
            b[i] = c2_raw[i];
        }

        uint256[K] memory m;
```

```
        for (uint i = 0; i < K; i++) m[i] = nSquared[i];

        uint256[K] memory r = BigInt2048.mulMod(a, b, m);

        uint256[K2] memory out;
        for (uint i = 0; i < K; i++) out[i] = r[i];
        for (uint i = K; i < K2; i++) out[i] = 0;
        return out;
    }

    function paillierMultiply(uint256[K2] calldata c_raw, uint256 exponent) public
view returns (uint256[K2] memory) {
        uint256[K] memory base;
        for (uint i = 0; i < K; i++) base[i] = c_raw[i];

        uint256[K] memory result = BigInt2048.one();

        uint256[K] memory m;
        for (uint i = 0; i < K; i++) m[i] = nSquared[i];

        uint256 e = exponent;
        while (e > 0) {
            if ((e & 1) == 1) {
                result = BigInt2048.mulMod(result, base, m);
            }
            e = e >> 1;
            if (e > 0) base = BigInt2048.mulMod(base, base, m);
        }

        uint256[K2] memory out;
        for (uint i = 0; i < K; i++) out[i] = result[i];
        for (uint i = K; i < K2; i++) out[i] = 0;
        return out;
    }

    function updateChemotherapySessions(bytes32 patientID, uint256[K2] calldata
newCipher) external {
        require(patients[patientID].exists, "Patient not found");
        for (uint i = 0; i < K2; i++)
patients[patientID].chemotherapySessionsEncrypted[i] = newCipher[i];
        emit ChemotherapySessionUpdated(patientID);
    }

    function getChemotherapySessions(bytes32 patientID) external view returns
(uint256[K2] memory) {
        require(patients[patientID].exists, "Patient not found");
        uint256[K2] memory out;
        for (uint i = 0; i < K2; i++) out[i] =
patients[patientID].chemotherapySessionsEncrypted[i];
        return out;
    }

    function getPaillierParams() external view returns (uint256[K] memory,
uint256[K2] memory) {
        uint256[K] memory nCopy;
        for (uint i = 0; i < K; i++) nCopy[i] = n[i];
        return (nCopy, n2_raw);
    }
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД ТА ЗАСІБ ЗАХИСТУ МЕДИЧНИХ ДАНИХ НА ОСНОВІ ГОМОМОРФНОГО ШИФРУВАННЯ

Виконала: студентка 2 курсу групи 1БС-24 м спеціальності 125 Кібербезпека та захист інформації

_____Владислава ЛАНОВА

Керівник: к. т. н., доц., доцент каф. ЗІ

_____Юрій БАРИШЕВ

«16» грудня 2025 р.

ПОРІВНЯЛЬНИЙ АНАЛІЗ НОРМАТИВНО-ПРАВОВИХ АКТІВ

Країна/Регіон	Основний закон/акт	Рік ухвалення	Основний акцент	Санкції/штрафи
США	HIPAA	1996	Захист медичної інформації (PHI), права пацієнтів, конфіденційність	До $1,5 млн на рік за категорію порушення; кримінальна відповідальність
ЄС	GDPR	2016 (чинний з 2018)	Захист особистих даних, що стосуються громадян	До €20 млн або 4% глобального обороту
Велика Британія	UK GDPR, Data Protection Act	2018	Адаптація GDPR після брекзиту, регулювання NHS	До £17,5 млн або 4% обороту
Канада	PIPEDA (федеральний), PHIPA (Онтаріо)	2000 / 2004	Захист персональних і медичних даних у приватному секторі	До $100 тис. CAD
Австралія	Privacy Act, My Health Records Act	1988 / 2012	Конфіденційність, електронні медичні записи	До AUD $2,22 млн
Японія	APPI	2003	Захист персональних і медичних даних	До ¥100 млн (≈ $900 тис.)
Китай	PIPL, Cybersecurity Law	2021 / 2017	Аналог GDPR, особливий захист медичних даних	До ¥50 млн або 5% обороту
Індія	Digital Personal Data Protection Act	2023	Захист персональних даних	До 2,5 млрд INR (≈ $30 млн)
Бразилія	LGPD	2020	Аналог GDPR	До 2% обороту, макс. BRL 50 млн
Україна	Закон України «Про захист персональних даних»	2010	Захист персональних даних громадян України	До 34 тис. грн
Україна	Закон України «Про електронну систему охорони здоров’я» (eHealth)	2018	Регулювання електронних медичних записів	Адміністративна відповідальність

МАТЕМАТИЧНИЙ ОПИС ЗАВДАННЯ

Нехай ***D*** – це дані пацієнтів, які зберігаються в їх медичній картці. Нехай ***M*** – це множина носіїв даних, які використовуються для зберігання ***D***, а ***S*** – множина даних, вже збережених на носіях з множини ***M***;

store(), update(), retrieve() – процеси зберігання, оновлення та отримання даних відповідно;

d, m – дані, пацієнтів, які належать множині D та носії даних відповідно;

s_m – результат зберігання даних на носіях.

$$MathematicalDescription =$$
$$= \{D, S, M, \{store(d, m), update(s_m), retrieve(s_m, m)\}\}$$

МЕТОД ЗАХИСТУ МЕДИЧНИХ ДАНИХ

Крок 1. Збір ідентифікаційних даних пацієнта, а саме – номер медичної картки та номер паспорта.

Крок 2. Збір параметрів пацієнта – зріст, вага, кількість проведених сесій хіміотерапії.

Крок 3. Обчислення геш-значення ідентифікатора пацієнта для формування шляху доступу (path) до його даних у блокчейні за допомогою функції Keccak-256.

Крок 4. Гомоморфне зашифрування параметрів зросту, ваги та кількості сесій хіміотерапії за публічним ключем.

Крок 5. Виклик методу смарт-контракту для створення або отримання профілю пацієнта в блокчейні за його унікальним ідентифікатором.

Крок 6. Зберігання зашифрованих даних в блокчейні.

Крок 7. Виклик методу смарт-контракту для отримання зашифрованих значень параметрів.

Крок 8. Використання гомоморфних перетворень для обчислення дози хіміотерапевтичного препарату.

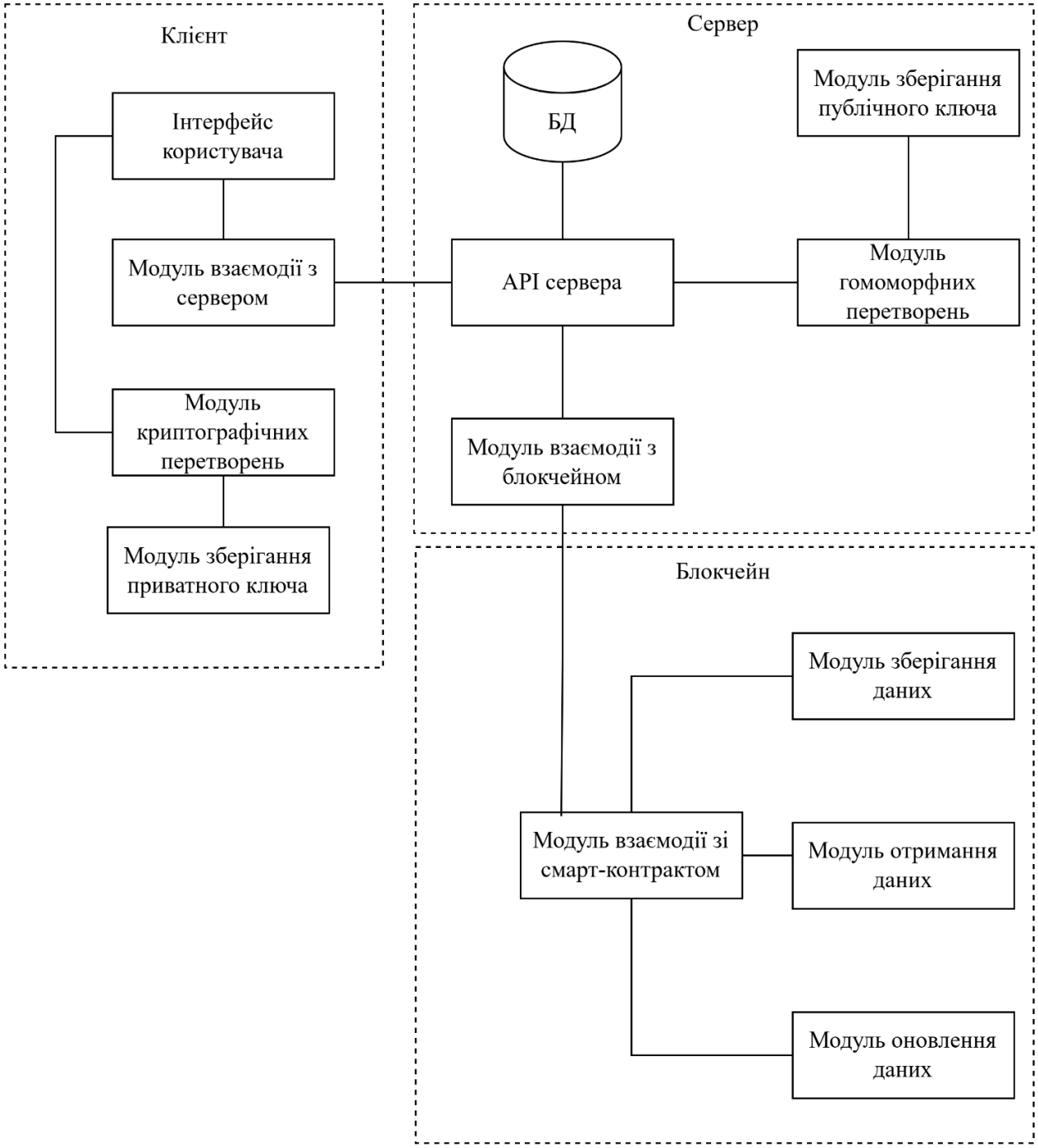
Крок 9. Розшифрування значення дози за приватним ключем.

Крок 10. Для оновлення кількості сесій хіміотерапії зашифрувати значення «1» за публічним ключем та отримати попередньо збережене значення кількості сесій.

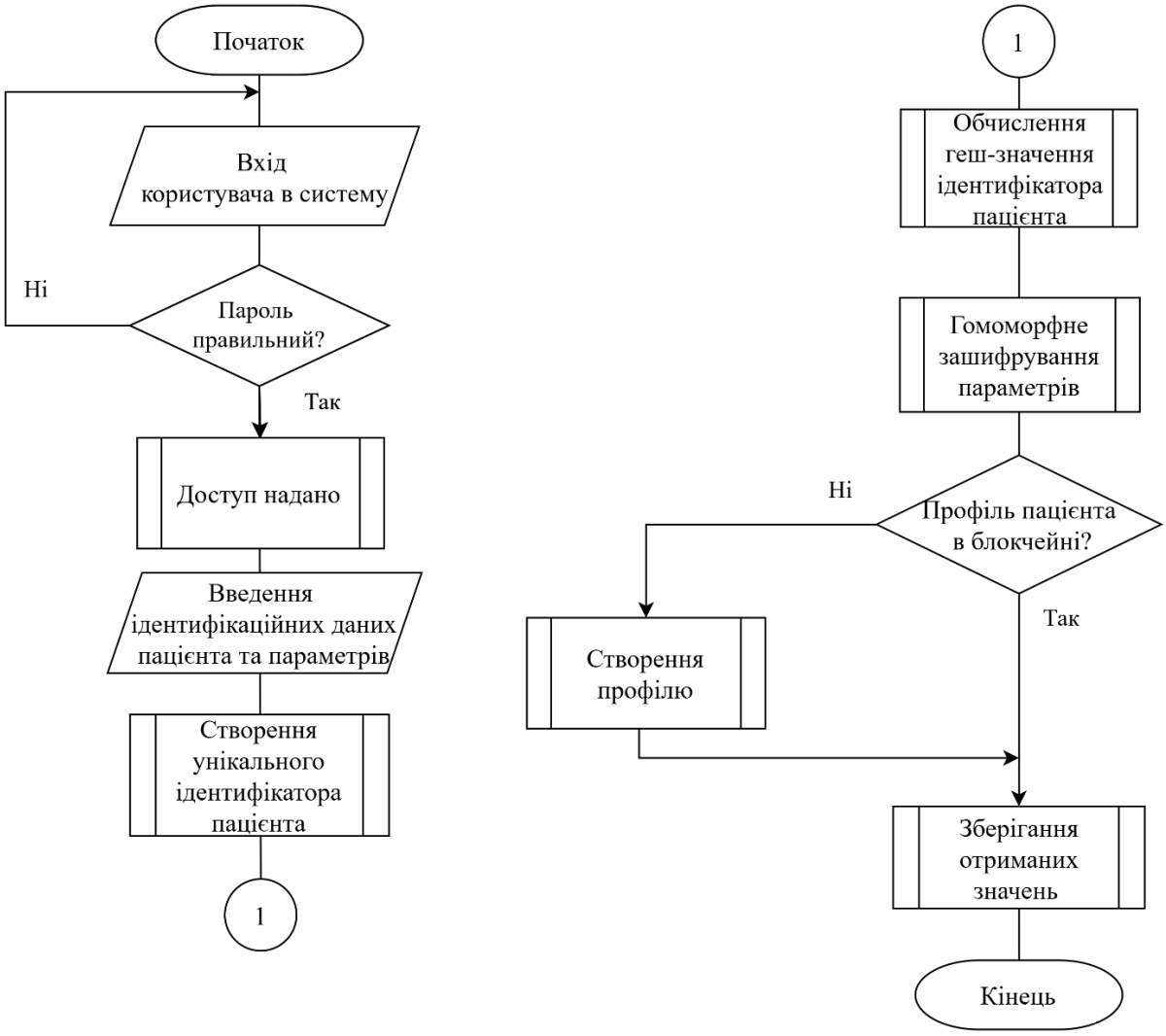
Крок 11. Виконання гомоморних перетворень для отримання оновленого значення кількості сесій.

Крок 12. Зберігання оновленого значення кількості сесій в блокчейні.

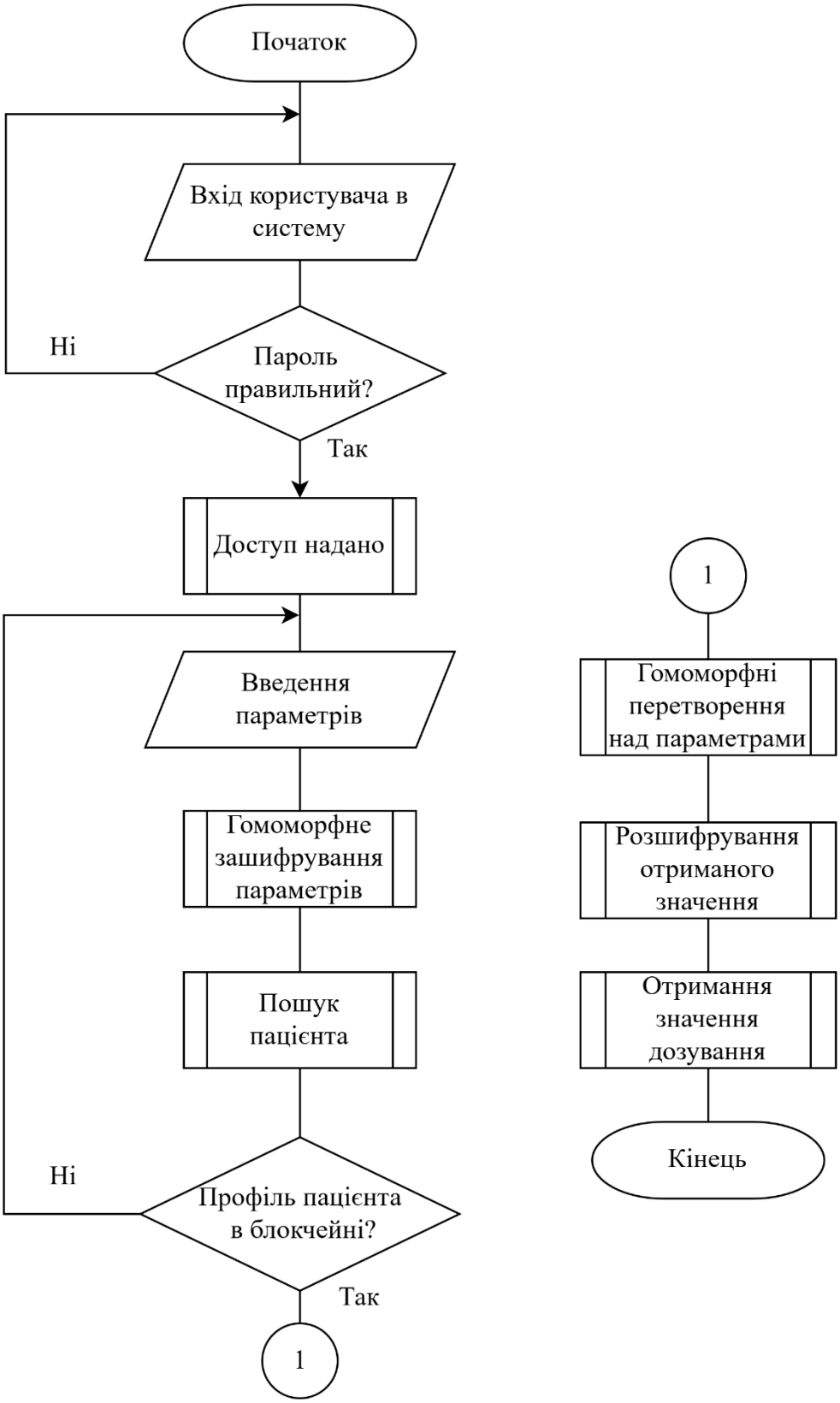
АРХІТЕКТУРА ЗАСОБУ



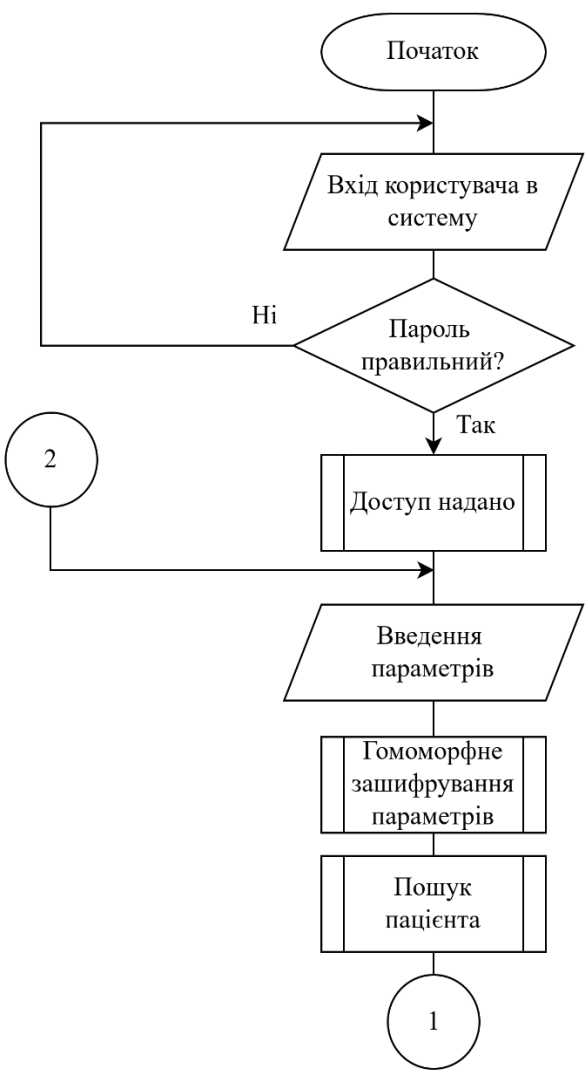
АЛГОРИТМ ЗБЕРІГАННЯ ДАНИХ

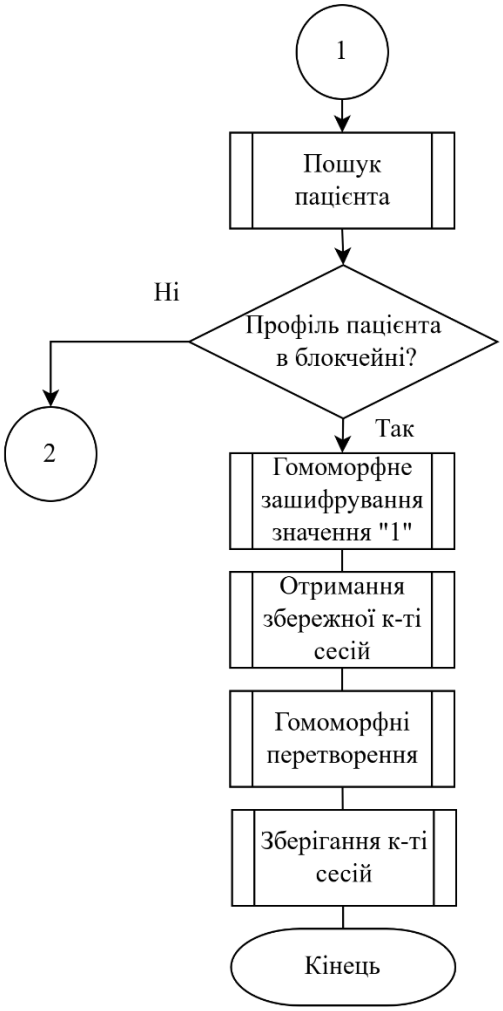


АЛГОРИТМ ОТРИМАННЯ ДАНИХ

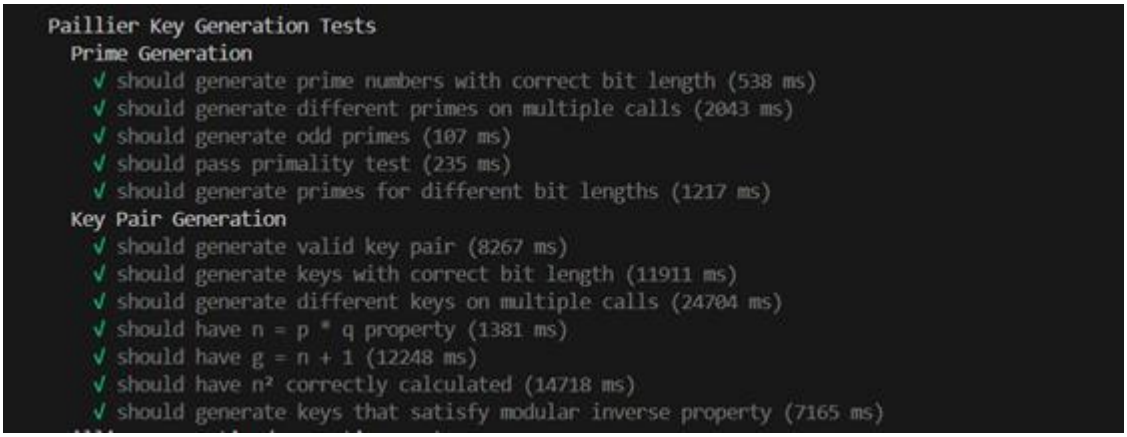


АЛГОРИТМ ОНОВЛЕННЯ ДАНИХ

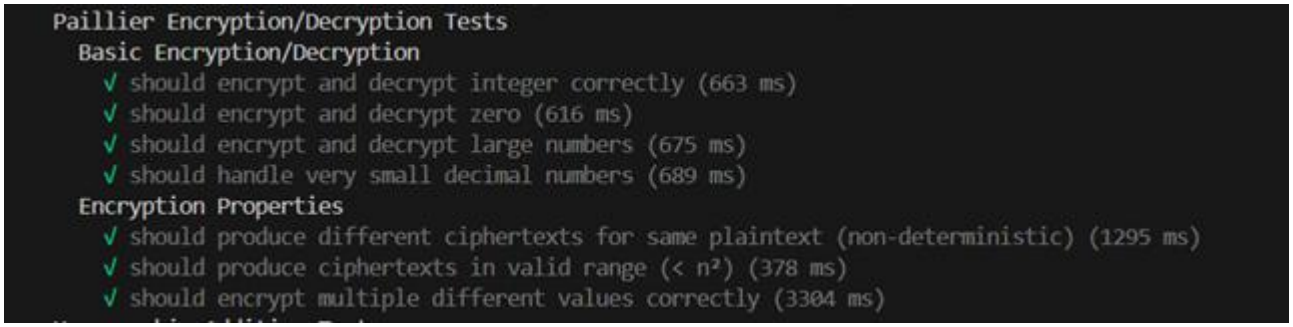




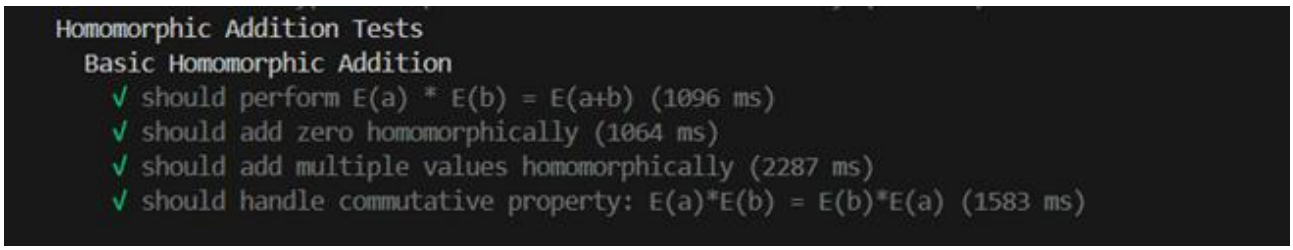
РЕЗУЛЬТАТИ МОДУЛЬНОГО ТЕСТУВАННЯ



Результати виконання тестів генерування ключів



Результати виконання тестів зашифрування/розшифрування

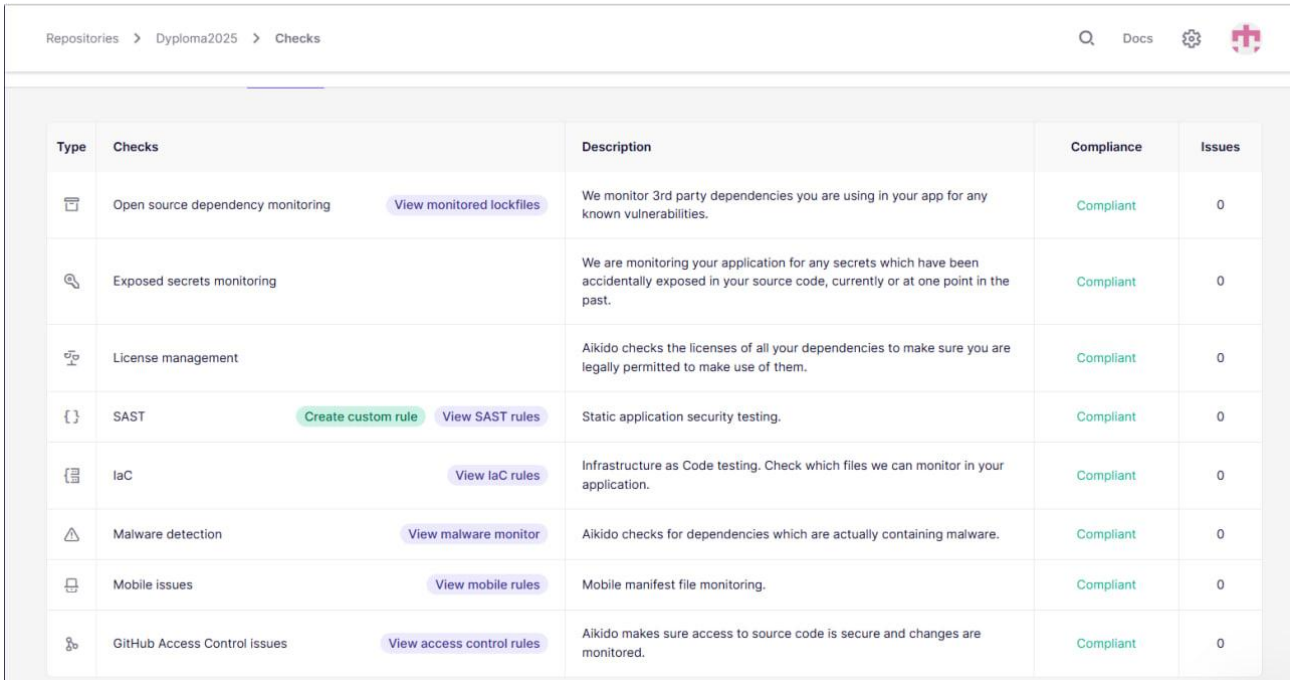


Результати виконання тестів гомоморфних перетворень

РЕЗУЛЬТАТИ ТЕСТУВАННЯ БЕЗПЕКИ

```
INFO:Detectors:
Parameter PatientChemotherapy.updatePatient(uint256,uint256,uint256,uint256)._id (PatientData.sol#18) is n
ot in mixedCase
Parameter PatientChemotherapy.updatePatient(uint256,uint256,uint256,uint256)._height (PatientData.sol#19)
is not in mixedCase
Parameter PatientChemotherapy.updatePatient(uint256,uint256,uint256,uint256)._weight (PatientData.sol#20)
is not in mixedCase
Parameter PatientChemotherapy.updatePatient(uint256,uint256,uint256,uint256)._chemotherapySessions (Patien
tData.sol#21) is not in mixedCase
Parameter PatientChemotherapy.getPatient(uint256)._id (PatientData.sol#28) is not in mixedCase
Parameter PatientChemotherapy.getChemotherapySessions(uint256)._id (PatientData.sol#34) is not in mixedCas
e
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-co
nventions
```

Результати статичного тестування безпеки смарт-контрактів



Type	Checks	Description	Compliance	Issues
	Open source dependency monitoring — View monitored lockfiles	We monitor 3rd party dependencies you are using in your app for any known vulnerabilities.	Compliant	0
	Exposed secrets monitoring	We are monitoring your application for any secrets which have been accidentally exposed in your source code, currently or at one point in the past.	Compliant	0
	License management	Aikido checks the licenses of all your dependencies to make sure you are legally permitted to make use of them.	Compliant	0
{ }	SAST — Create custom rule — View SAST rules	Static application security testing.	Compliant	0
	IaC — View IaC rules	Infrastructure as Code testing. Check which files we can monitor in your application.	Compliant	0
	Malware detection — View malware monitor	Aikido checks for dependencies which are actually containing malware.	Compliant	0
	Mobile issues — View mobile rules	Mobile manifest file monitoring.	Compliant	0
	GitHub Access Control issues — View access control rules	Aikido makes sure access to source code is secure and changes are monitored.	Compliant	0

Результат статичного тестування безпеки всього проєкту

ПОКАЗНИКИ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Критерії	Бали		
	Войтович О. П.	Лукічов В. В.	Гарнага В. А.
Технічна здійсненність концепції	3	3	3
Ринкові переваги (наявність аналогів)	2	3	2
Ринкові переваги (ціна продукту)	4	4	4
Ринкові переваги (технічні властивості)	4	4	4
Ринкові переваги (експлуатаційні витрати)	3	4	4
Ринкові переваги (розмір ринку)	2	3	3
Ринкові переваги (конкуренція)	3	3	3
Практична здійсненність (наявність фахівців)	4	3	3
Практична здійсненність (наявність фінансів)	4	4	4
Практична здійсненність (необхідність нових матеріалів)	4	4	4
Практична здійсненність (термін реалізації)	4	4	4
Практична здійсненність (розробка документів)	4	4	4
Сума балів	41	43	42
Середньоарифметична сума балів, $СБ_c$	42		

$$K_{\text{ІНТ}} = 1 \cdot \frac{1{,}16}{0{,}785} = 1{,}47$$

$$\text{ЗВ} = \frac{205237{,}41}{0{,}7} = 293196{,}3$$

$$T_{\text{ок}} = \frac{1}{0{,}78} = 1{,}28$$