Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
В.о. завідувача кафедрою
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО
«___» червня 2025 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньо-професійної програми)

на тему: «Засоби захисту обміну даними з використанням блокчейн-технологій»

Виконавець: студент IV курсу, групи КБ-42

_____ Ілля ЮССЕФ
(підпис) (ім’я, прізвище)

	Підпис	**Ім’я ПРІЗВИЩЕ**
Керівник		Леся БАРАНОВСЬКА
Нормоконтроль		Лариса МИРУТЕНКО

Київ 2025

Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.о. завідувача кафедрою
кібербезпеки та захисту інформації
_____Іван ПАРХОМЕНКО
«29» листопада 2024 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

Студенту **КБ-42** (група) **Юссеф Ілля Іссайович** (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Засоби захисту обміну даними з використанням блокчейн-технологій

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №6 від 28.11.2024 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Засоби шифрування повідомлень, повідомлення які потребують захисту, алгоритми асиметричного та гібридного шифрування, блокчейн алгоритми

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Потрібно проаналізувати методи захисту інформації, розробити захищену систему обміну повідомленнями з використанням криптографії та блокчейн-технологій, реалізувати шифрування, аутентифікацію користувачів, створити інтерфейс і провести тестування ефективності

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розроблено вебзастосунок для обміну зашифрованими повідомленнями між користувачами та верифікації їх за допомогою блокчейну. Даний вебзастосунок може бути використано у сферах, де критично важливі захист і довіра до обміну даними, таких як корпоративна комунікація, банківська справа,електронне урядування

5. ДАТА ВИДАЧІ ЗАВДАННЯ

Дата видачі завдання: «29» листопада 2024 року

Завдання видала		Леся БАРАНОВСЬКА
	(підпис)	(ініціали, прізвище)
Завдання прийняв до виконання		Ілля ЮССЕФ
	(підпис)	(ініціали, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів робіт	Строки виконання робіт (початок-кінець)	Відмітка про виконання
1	Уточнення постановки задачі	29.11.2024 – 03.12.2024	*виконано*
2	Аналіз літератури	04.12.2024 – 28.12.2024	*виконано*
3	Аналіз цифрових загроз і вразливостей текстових, графічних та відеоповідомлень	29.12.2024 – 16.01.2025	*виконано*
4	Розгляд криптографічних алгоритмів шифрування (AES, RSA, Fernet)	17.01.2025 – 03.02.2025	*виконано*
5	Побудова асиметричного та гібридного шифрування для повідомлень	04.02.2025 – 05.03.2025	*виконано*
6	Розробка структури блокчейну для зберігання зашифрованих повідомлень	06.03.2025 – 27.03.2025	*виконано*
7	Інтеграція криптографії та блокчейн-технології в єдину систему	28.03.2025 – 17.04.2025	*виконано*
8	Реалізація аутентифікації, авторизації та користувацького інтерфейсу	18.04.2025 – 05.05.2025	*виконано*
9	Тестування системи, аналіз ефективності, візуалізація метрик	06.05.2025 – 21.05.2025	*виконано*
10	Оформлення пояснювальної записки	22.05.2025 – 02.06.2025	*виконано*
11	Підготовка до захисту	03.06.2025 – 13.06.2025	*виконано*

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

Завдання прийняв до виконання _____ Ілля ЮССЕФ
(підпис) (ініціали, прізвище)

Термін подання кваліфікаційної роботи до ЕК 13 червня 2025 року

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків, має 68 сторінок основного тексту, 19 рисунків та 1 таблиця. Список використаних джерел містить 27 найменувань і займає 3 сторінки.

Метою роботи є розробка захищеної вебсистеми обміну повідомленнями з використанням криптографії та блокчейн-технологій для забезпечення конфіденційності, цілісності та автентичності.

Для досягнення зазначеної мети поставлено наступні завдання:

- провести аналіз існуючих підходів до захисту інформації під час її обміну в мережі;
- розробити механізми асиметричного шифрування для текстових повідомлень та гібридного шифрування для мультимедійних повідомлень;
- побудувати структуру блокчейну для зберігання зашифрованих повідомлень у вигляді транзакцій;
- створити інтерфейс користувача для взаємодії з системою, включно з авторизацією, надсиланням, переглядом повідомлень та візуалізацією блокчейн-ланцюга.

Об'єктом дослідження є процес передачі цифрових повідомлень різних форматів, що передаються у мережі та потребують захисту.

Предмет дослідження є методи шифрування (симетричне, асиметричне, гібридне) та блокчейн для захисту повідомлень.

Методи дослідження:

- аналіз сучасних криптографічних методів шифрування для захисту цифрових повідомлень;
- аналіз архітектури та принципів функціонування блокчейн-технологій у сфері інформаційної безпеки;

- синтез криптографічних та блокчейн-рішень для побудови системи захищеного обміну повідомленнями;
- розробка та тестування веб-застосунку з функціями шифрування, автентифікації, авторизації та верифікації повідомлень через блокчейн-реєстр.

Практичною цінністю отриманих результатів є розроблений вебзастосунок для обміну зашифрованими повідомленнями між користувачами та верифікації їх за допомогою блокчейну.

Даний вебзастосунок може бути використаний у сферах, де критично важливі захист і довіра до обміну даними, таких як корпоративна комунікація, банківська справа, електронне урядування.

Ключові слова: блокчейн, шифрування, RSA, обмін повідомленнями, інформаційна безпека, Flask, Fernet, гібридне шифрування, цифровий підпис.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES	–	Advanced Encryption Standard
DRM	–	Digital Rights Management
Exif	–	Exchangeable Image File Format
HLS	–	HTTP Live Streaming
IP	–	Internet Protocol
RSA	–	Rivest–Shamir–Adleman
SHA	–	Secure Hash Algorithm
TLS	–	Transport Layer Security

ВСТУП

У контексті глобальної цифровізації стрімко зростає обсяг чутливої інформації, яка передається через електронні комунікаційні канали. Паралельно з цим зростає кількість атак, спрямованих на перехоплення, підміну та підробку повідомлень, особливо у сфері державного управління, фінансових операцій, електронного документообігу та приватного спілкування. Традиційні централізовані моделі обміну даними, попри впроваджені засоби захисту, залишаються вразливими до низки загроз – від фішингу й MITM-атак до компрометації серверів і витоку ключів.

У цьому контексті особливу актуальність набуває поєднання криптографічних механізмів (шифрування, цифровий підпис, хешування) з децентралізованими технологіями, зокрема блокчейн. Блокчейн як технологія забезпечує незмінність, прозорість та достовірність даних, а при інтеграції з криптографічними алгоритмами дозволяє формувати безпечне середовище для передачі цифрових повідомлень. Такі рішення сприяють формуванню нової парадигми безпеки цифрового обміну – децентралізованого, захищеного та верифікованого.

Метою кваліфікаційної роботи є розробка захищеної вебсистеми обміну повідомленнями з використанням криптографії та блокчейн-технологій для забезпечення конфіденційності, цілісності та автентичності.

Для досягнення зазначеної мети поставлено наступні завдання:

- провести аналіз існуючих підходів до захисту інформації під час її обміну в мережі;
- розробити механізми асиметричного шифрування для текстових повідомлень та гібридного шифрування для мультимедійних повідомлень;
- побудувати структуру блокчейну для зберігання зашифрованих повідомлень у вигляді транзакцій;

- створити інтерфейс користувача для взаємодії з системою, включно з авторизацією, надсиланням, переглядом повідомлень та візуалізацією блокчейн-ланцюга.

Об'єкт дослідження: процес передачі цифрових повідомлень різних форматів, що передаються у мережі та потребують захисту.

Предмет дослідження: методи шифрування (симетричне, асиметричне, гібридне) та блокчейн для захисту повідомлень.

Оцінка сучасного стану проблеми на основі вітчизняної та зарубіжної літератури. Аналіз наукових і технічних джерел свідчить про зростання інтересу до застосування блокчейну у сфері кібербезпеки. У зарубіжних виданнях країн США, ЄС блокчейн активно інтегрується в системи документообігу, автентифікації та обміну конфіденційними даними. В Україні досвід практичного впровадження таких рішень є обмеженим, хоча потреба в децентралізованих, прозорих і захищених каналах обміну інформацією зростає, зокрема у військовому, банківському та державному секторах.

Галузь застосування. Результати дослідження можна використовувати у сфері державного управління, електронного документообігу, правовому та фінансовому середовищі, а також в корпоративному секторі – всюди, де вимагається гарантована цілісність і конфіденційність обміну даними.

В даній роботі інтегровано сучасні криптографічні схеми (гібридне шифрування з використанням RSA + Fernet) з приватним блокчейн-реєстром транзакцій, що дозволяє фіксувати кожне повідомлення або файл у вигляді незмінної записи з міткою часу. Така архітектура підвищує довіру до процесу передачі даних і унеможливлює їхнє фальшування.

Практична цінність дослідження полягає у розробленому вебзастосунку для обміну зашифрованими повідомленнями між користувачами та верифікації їх за допомогою блокчейну.

Даний вебзастосунок може бути використано у сферах, де критично важливі захист і довіра до обміну даними, таких як корпоративна комунікація, банківська справа, електронне урядування.

РОЗДІЛ 1
ПІДХОДИ ДО ЗАХИСТУ ІНФОРМАЦІЇ В ПОВІДОМЛЕННЯХ, ТА МУЛЬТИМЕДІА ДАНИХ

1.1 Основні типи цифрових повідомлень та їх вразливості

У сучасному цифровому середовищі обмін інформацією відбувається у різних форматах, і кожен з них має свої особливості та потенційні вразливості. З розвитком цифрових технологій значно збільшився обсяг даних, що передається через електронні засоби зв’язку, а також кількість каналів, якими ці повідомлення можуть бути доставлені до користувача. Зрозуміння типів цифрових повідомлень і потенційних загроз, які їм притаманні, є критично важливим етапом при розробці систем захисту даних, зокрема при використанні блокчейн-технологій [1, 2].

Усі цифрові повідомлення можна умовно поділити на кілька основних категорій:

- Текстові повідомлення – це найбільш поширений формат, що включає короткі повідомлення (SMS, чати), електронні листи, документи тощо.
- Графічні зображення – фотографії, скріншоти, скановані документи, графіки.
- Аудіо – голосові повідомлення, музичні треки, подкасти, аудіофайли у форматах MP3, WAV тощо.
- Відео – відеоповідомлення, записи конференцій, стріми, кліпи тощо.
- Мультимедійні комбінації – поєднання кількох типів даних, наприклад, відео з аудіосупроводом, презентації з вбудованими відеофрагментами, інтерактивний контент.

Кожен з цих форматів має різну структуру та ступінь складності з точки зору збереження цілісності та конфіденційності. Наприклад, текст легше піддати

криптографічному аналізу, ніж великі відеофайли, проте останні містять більше метаданих і є більш важкими для обробки та контролю.

Передача цифрових повідомлень здійснюється через різноманітні канали, серед яких варто виділити:

- Електронна пошта – класичний канал, що часто піддається фішинговим атакам.
- Миттєві месенджери – такі як WhatsApp, Telegram, Signal, Viber, які дозволяють обмінюватися текстом, зображеннями, відео та аудіо [3].
- Соціальні мережі – Facebook, Instagram, TikTok, які включають широкі можливості для мультимедійного обміну.
- VoIP-зв’язок – голосові та відеодзвінки через Skype, Zoom, Microsoft Teams, Discord.

Ці канали мають різні рівні захисту, і вразливості можуть виникати як на рівні клієнта, так і на рівні сервера або під час передавання даних у мережі.

Основні загрози для цифрових повідомлень пов’язані з порушенням конфіденційності, цілісності та автентичності даних [4, 5]. До найпоширеніших загроз належать:

- Перехоплення повідомлень – хакер отримує доступ до трафіку між відправником і отримувачем, що дозволяє йому зчитувати або змінювати повідомлення.
- Модифікація вмісту – зловмисник змінює зміст повідомлення під час його передавання, що призводить до втрати цілісності інформації.
- Підміна джерела повідомлення – атаки, під час яких зловмисник видає себе за легітимного відправника (спуфінг), вводячи одержувача в оману.
- Несанкціонований доступ або витік даних – включає як зовнішні атаки, так і внутрішні інциденти, коли працівники чи партнери компанії навмисно чи випадково розголошують конфіденційну інформацію.

У багатьох випадках ці загрози реалізуються через вразливості програмного забезпечення, недосконаліть протоколів захисту або людський фактор . [6]

Такі чинники значно підвищують ризик несанкціонованого доступу, втрати або підміни інформації.

Цифрові повідомлення є об'єктами трьох основних видів атак:

- На конфіденційність – спрямовані на незаконне отримання інформації. Приклад – зчитування переписки в незахищеному месенджері через скомпрометовану мережу Wi-Fi.
- На цілісність – підміна або спотворення вмісту, наприклад, модифікація банківського рахунку в електронному листі.
- На автентичність – використання фальшивих акаунтів або підроблених цифрових підписів для того, щоб видатися надійним джерелом.

Для запобігання цим атакам використовують криптографічні алгоритми, цифрові сертифікати, захищені канали зв’язку (TLS/SSL) та багатоетапну автентифікацію.

Існує безліч задокументованих інцидентів, що демонструють масштаб проблеми:

- Фішингові атаки через SMS та електронну пошту – користувачам надсилають підроблені повідомлення від банків або сервісів із метою викрадення облікових даних. Наприклад, масові атаки на клієнтів українських банків через підроблені листи НБУ.
- Злом акаунтів у месенджерах – випадки, коли зловмисники отримують контроль над обліковими записами WhatsApp або Telegram через фішингові сайти або соціальну інженерію.
- Атаки на платформи відеоконференцій – наприклад, у 2020 році набули поширення інциденти "Zoombombing", коли сторонні особи підключалися до конференцій без дозволу через слабкий захист або відсутність пароля.
- Discord як платформа для розповсюдження шкідливих файлів – платформа, що не призначена для офіційного обміну документами, все частіше використовується для фішингу та інфікування пристроїв через вкладення.

Цифрові повідомлення, незалежно від формату, залишаються вразливими до широкого спектру атак. У зв’язку з цим постає питання забезпечення належного рівня захисту таких даних.

1.2 Особливості захисту текстової інформації в повідомленнях

Текстова інформація є одним із найпоширеніших та водночас найбільш вразливих типів даних, що передаються через електронні комунікації. Повідомлення, які передаються у вигляді тексту, охоплюють електронну пошту, чати в месенджерах, SMS, нотатки в хмарних сервісах, а також внутрішньо системні лог-файли або адміністративні протоколи. Забезпечення безпеки такої інформації вимагає врахування як технічних, так і організаційних аспектів, зокрема впровадження криптографічних алгоритмів, верифікації джерела повідомлення, захисту від спуфінгу та соціальної інженерії.

Симетричне шифрування – це криптографічний метод, при якому один і той самий ключ використовується як для шифрування, так і для розшифрування повідомлення. Прикладом широко застосовуваного симетричного алгоритму є AES (Advanced Encryption Standard) [7, 8]. Він підтримує довжину ключа 128, 192 і 256 біт і є дуже ефективним при обробці великих обсягів даних.

Основна перевага симетричного шифрування – це його швидкодія. Однак головна проблема полягає у безпечному розповсюдженні ключів: якщо ключ перехоплений під час передачі, зловмисник може легко розшифрувати будь-які повідомлення.

Для подолання проблеми розповсюдження ключів використовують асиметричне шифрування, де застосовуються два ключі – публічний (відкритий) і приватний (закритий). Відомим прикладом є алгоритм RSA [9, 10, 12].

Відправник шифрує текст повідомлення публічним ключем одержувача, а той розшифровує повідомлення своїм приватним ключем. Такий підхід дозволяє безпечно надсилати повідомлення навіть у незахищеному середовищі. Однак

асиметричне шифрування є більш ресурсоємним, тому часто комбінується з симетричними алгоритмами (наприклад, в гібридних криптосистемах) [11].

Хеш-функція – це алгоритм, який перетворює довільний вхідний текст у фіксований набір символів (хеш), що виконує роль цифрового «відбитка» повідомлення. Популярні алгоритми: SHA-256**,** SHA-3**,** BLAKE2 [5].

Хеш-функції використовуються для перевірки цілісності даних: якщо повідомлення змінено хоча б на один символ, його хеш зміниться кардинально. Це дозволяє оперативно виявити будь-які зміни.

Електронний підпис базується на асиметричному шифруванні і використовується для верифікації відправника та забезпечення незаперечності повідомлення [12, 13]**.**

Процес створення підпису:

1. Повідомлення хешується.
2. Хеш шифрується приватним ключем відправника.
3. Отриманий підпис надсилається разом із повідомленням.

Одержувач перевіряє справжність підпису, розшифрувавши його публічним ключем та порівнюючи з самостійно отриманим хешем повідомлення.

Протокол Signal – сучасний криптографічний стандарт, який лежить в основі роботи месенджерів Signal**,** WhatsApp**,** Messenger (у режимі «таємної розмови») [3, 16].

Його особливості:

- наскрізне шифрування (end-to-end),
- використання Double Ratchet алгоритму, що змінює ключ для кожного повідомлення,
- підтримка перенесення довіри та перевірки ідентичності співрозмовника.

Завдяки цим функціям Signal Protocol забезпечує один із найвищих рівнів безпеки у цифровій комунікації.

Для перевірки автентичності джерела застосовують [5]:

- Цифрові сертифікати X.509.

- SPF (Sender Policy Framework) – перевірка дозволених IP-адрес для відправки пошти з певного домену.
- DKIM (DomainKeys Identified Mail) – перевірка підпису в заголовках листа.
- DMARC (Domain-based Message Authentication, Reporting and Conformance) – комплексна політика перевірки автентичності.

Фішинг – поширена атака, при якій зловмисник видає себе за легітимного користувача або організацію для збору конфіденційних даних. Основні способи боротьби [14]:

- двофакторна аутентифікація (2FA),
- візуальна перевірка доменів,
- антифішингові плагіни в браузерах,
- навчання користувачів розпізнаванню фішингових повідомлень.

Спуфінг дозволяє зловмисникам маскувати справжнє джерело повідомлення. Для боротьби використовуються:

- DNSSEC (розширення системи DNS),
- використання цифрових підписів,
- блокування нестандартних заголовків у поштових серверах.

Навіть при використанні найсучасніших криптографічних алгоритмів, людський фактор**,** помилки конфігурації та централізація залишаються ключовими вразливими місцями. У разі зламу серверу електронної пошти або витоку ключів зловмисник може отримати доступ до великої кількості даних.

Крім того, централізовані сервіси зберігають повідомлення на стороні постачальника (наприклад, Google, Microsoft), що створює ризики несанкціонованого доступу з боку внутрішніх або зовнішніх загроз.

Технології, такі як блокчейн пропонують альтернативний підхід – зберігання цифрових підписів, ключів або навіть повідомлень у розподілених реєстрах, що унеможливлює централізоване маніпулювання або підміну даних [1, 2, 15]. У наступних розділах буде розглянуто, як блокчейн може підвищити захист обміну повідомленнями.

Захист текстової інформації в цифрових комунікаціях потребує комплексного підходу, який включає криптографію, верифікацію автентичності, захист від соціальної інженерії та впровадження сучасних протоколів. Попри високий рівень розвитку традиційних рішень, централізовані системи все ще мають критичні недоліки, що відкриває перспективу для впровадження децентралізованих технологій на основі блокчейну як наступного етапу еволюції безпеки даних.

1.3 Проблеми безпеки при передачі зображень

Зображення відіграють не менш важливу роль, ніж текстова інформація. Вони активно використовуються в комунікаціях, соціальних мережах, електронній комерції, медичних та юридичних системах, верифікаційних процесах (наприклад, у формі сканів документів) тощо. Проте передача, зберігання та обробка графічних файлів супроводжується рядом серйозних викликів у сфері інформаційної безпеки. На відміну від простого тексту, зображення мають низку специфічних особливостей – великий обсяг, наявність метаданих, можливість приховування інформації – які відкривають додаткові вектори атак.

Цифрові зображення зберігаються у форматах JPEG, PNG, BMP, TIFF, HEIC тощо. На відміну від тексту, ці файли мають значно більший обсяг, що створює виклики щодо ефективності шифрування та зберігання. Окрім самих піксельних даних, у зображеннях часто містяться метадані – додаткова службова інформація, яка може включати:

- дату і час створення файлу;
- модель камери;
- геолокацію (GPS-координати);
- назву автора або пристрою;
- історію редагування.

Такі дані зберігаються у структурі EXIF (Exchangeable Image File Format). Їхня наявність може призвести до витоку особистої або конфіденційної інформації навіть без потреби аналізу вмісту зображення.

Однією з найбільш поширених загроз є витік конфіденційної інформації через EXIF-дані. Наприклад, фотографія, зроблена на смартфон, може містити координати з точністю до кількох метрів [4, 5]. Якщо така фотографія викладається у відкритий доступ, її може використати зловмисник для ідентифікації місцезнаходження користувача.

Відомі випадки, коли журналісти або правозахисники випадково викладали фотографії з геотегами, які пізніше використовувалися для стеження чи атак [5]. У корпоративному середовищі такий витік може розкрити внутрішні локації, IP-адреси принтерів чи імена серверів.

Зображення можуть використовуватися для приховання інформації шляхом стеганографії – техніки, яка дозволяє вписувати текстові або бінарні повідомлення в непомітний для ока спосіб, наприклад, змінюючи незначні біти кольорових пікселів [2].

Також існують приклади ін’єкції шкідливих скриптів у графічні файли. Зловмисники можуть вставити JavaScript, shell-код або інші шкідливі інструкції, які будуть виконані вразливими системами перегляду або обробки зображень.

З появою інструментів на базі штучного інтелекту стало можливим створення фальшивих зображень, включаючи deepfake-фотографії [2]. Змінене зображення може бути використане для дезінформації, шантажу, підроблення доказів тощо. Також можлива заміна файлів під час передачі, коли справжнє зображення підміняється шкідливим або маніпульованим аналогом.

Цифровий водяний знак – це інформація, приховано вбудована у зображення, що використовується для верифікації автентичності або авторства [5]. На відміну від стеганографії, водяний знак зазвичай призначений для виявлення, а не для приховування. Існують видимі (наприклад, логотип) та невидимі (наприклад, зміна частоти пікселів) водяні знаки. Цифрові водяні знаки

можуть бути стійкими до атак, таких як стиснення, обрізання чи зміна розміру, що робить їх корисними для захисту авторських прав.

Такі технології широко застосовуються в:

- системах захисту авторських прав;
- відстеженні витоку медіаконтенту;
- захисті цифрових доказів у судових справах.

Для контролю цілісності зображення використовується хешування [5, 19]. Перед передачею зображення створюється його хеш-сума (наприклад, алгоритмами MD5**,** SHA-1**,** SHA-256), яка перевіряється після доставки. Якщо файл змінено – навіть на один байт – хеш змінюється повністю.

Цей метод не дозволяє запобігти атакам, але допомагає виявити підміну або модифікацію файлу на етапі передачі чи зберігання.

Щоб забезпечити автентичність зображення, можна використовувати цифровий підпис, який створюється за допомогою закритого ключа відправника [10, 12, 19]. Це дозволяє одержувачу переконатися, що:

- файл справді надісланий заявленим джерелом;
- файл не було змінено після створення.

У практиці цифрової криміналістики підписані зображення можуть бути використані як докази, якщо забезпечено ланцюг довіри (chain of custody).

Навіть найкращі методи шифрування не допоможуть, якщо передача здійснюється незахищеним каналом [5, 8]. Саме тому використовуються TLS (Transport Layer Security) та HTTPS для забезпечення шифрування на транспортному рівні. Це запобігає перехопленню зображень (sniffing) або втручанню посередників (man-in-the-middle attack).

Останніми роками з'явилися численні випадки використання deepfake-зображень – змінених за допомогою ШІ фото осіб у компрометуючих ситуаціях. Такі фото складно ідентифікувати як підробку без спеціальних алгоритмів. У деяких країнах deepfake-використання вже заборонене на законодавчому рівні.

У 2020-х роках дослідники з безпеки неодноразово повідомляли про атаки, у яких JavaScript-код вбудовувався у поля метаданих EXIF. Після завантаження

такого зображення на вразливий вебсайт скрипт виконувався в браузері, відкриваючи шлях до XSS-атак (Cross-site Scripting) [19].

Зловмисники можуть надсилати фальшиві зображення, які на вигляд є копіями офіційних документів, сканів паспортів, рахунків тощо. Якщо користувач не верифікує автентичність зображення (через підпис або інший механізм), його легко обманути.

Передача та обробка зображень у цифровому середовищі супроводжується великою кількістю ризиків, які значно перевищують загрози, притаманні звичайному тексту. Зображення можуть містити як явну, так і приховану інформацію – у вигляді метаданих, стеганографічних повідомлень або шкідливих скриптів – і при цьому залишатися візуально незмінними для користувача. Тому ефективний захист графічного контенту має включати не лише шифрування даних та використання захищених каналів передачі (TLS/HTTPS), а й механізми верифікації автентичності та цілісності, зокрема хешування, цифрові підписи та водяні знаки.

З поширенням фальсифікацій, особливо тих, що створюються за допомогою генеративного штучного інтелекту, зростає потреба у надійному зберіганні зображень з обмеженим доступом у захищеному цифровому середовищі. Генеративні моделі здатні створювати фейкові зображення, які важко відрізнити від справжніх, що створює серйозні ризики у сферах, де автентичність зображення має критичне значення. У цьому контексті блокчейн-технології набувають особливої актуальності, оскільки забезпечують незмінність записів, прозорість доступу до даних, а також дають змогу перевіряти джерело походження кожного зображення [1, 6, 15]. Завдяки фіксації хеш-суми графічного файлу та відповідної метаінформації у розподіленому реєстрі можна гарантувати, що зображення не було змінено або підмінено з моменту його завантаження до системи. Такий підхід дозволяє автоматизувати процес верифікації цілісності візуального контенту.

Крім того, децентралізована модель зберігання, яку пропонує блокчейн, дозволяє реалізувати гнучкий контроль доступу без залучення центральних

посередників. Це має особливе значення у випадках, коли мова йде про обробку конфіденційної або чутливої графічної інформації, зокрема у медичній, правовій чи ідентифікаційній сферах. Інтеграція блокчейну з сучасними криптографічними протоколами, зокрема цифровими підписами та системами керування правами доступу, створює умови для формування безпечного середовища, в якому кожна дія над зображенням може бути відстежена, перевірена та зафіксована. Таким чином, поєднання традиційних методів захисту з можливостями блокчейн-технологій забезпечує високий рівень довіри до цифрових зображень та сприяє побудові ефективної системи захисту візуальних даних.

1.4 Захист відеоконтенту: загрози та методи протидії

Відеоконтент став ключовим засобом передачі інформації, розваг і навчання [5]. Платформи потокового мовлення (наприклад, YouTube, Netflix, Twitch), системи відеоконференцій (Zoom, Microsoft Teams), а також відеонагляд (CCTV, IP-камери) генерують та обробляють величезні обсяги відеоданих. З огляду на це, забезпечення безпеки відеоконтенту є однією з найскладніших задач у сфері кіберзахисту, адже відеофайли мають великий розмір, передаються в режимі реального часу, та можуть бути легко змінені або скопійовані зловмисниками.

Відео має низку характеристик, які створюють специфічні виклики для кібербезпеки:

- Великий обсяг даних: відеофайли, особливо у високій роздільній здатності (Full HD, 4K, 8K), можуть важити десятки гігабайтів, що ускладнює їх шифрування, зберігання та передавання.
- Потоковість: більшість відео передаються у вигляді стрімінгових потоків, що потребує захисту в режимі реального часу.
- Комбіновані дані: відео поєднує зображення, звук, субтитри, інтерактивні елементи – кожен з цих компонентів може мати власні вразливості.

- Труднощі автентифікації: у разі підміни відео або модифікації окремих кадрів звичайний користувач не завжди здатен виявити фальсифікацію.

Найбільш поширеною загрозою є неліцензійне копіювання та розповсюдження відеоконтенту, особливо в сфері розваг [5]. Фільми, серіали, спортивні трансляції та концерти часто копіюються із легальних платформ і публікуються на піратських сайтах. Це призводить до:

- значних фінансових збитків правовласникам;
- порушення авторського права;
- розповсюдження відео з вбудованими шкідливими елементами (вбудований малваре).

З появою генеративного ШІ виникла нова загроза – deepfake-відео [2]. Такі ролики дозволяють змінювати обличчя, голос або дії реальної особи. Це може використовуватись у:

- політичній пропаганді;
- шахрайстві (наприклад, підробка відеозвернення керівника компанії з проханням переказати кошти);
- репутаційних атаках.

Під час прямої трансляції через стрімінгові платформи (IPTV, OTT, VoD) існує ризик перехоплення відео (packet sniffing, MITM) [5]. Якщо відеопотік передається без шифрування, зловмисник може:

- копіювати трансляцію у реальному часі;
- інжектувати шкідливий контент;
- підмінити окремі фрагменти відео.

Системи керування цифровими правами (DRM) – це набір технологій, які регулюють доступ до відеоконтенту, обмежують копіювання, перегляд та поширення [5].

Основні функції DRM:

- Шифрування відео на стороні сервера.
- Видача ключів розшифрування лише автентифікованим користувачам.

- Обмеження дій користувача: заборона збереження, обмеження часу перегляду, кількість переглядів тощо.

Популярні DRM-платформи: Google Widevine, Apple FairPlay, Microsoft PlayReady.

Для захисту відео, що передається потоково, використовуються спеціальні протоколи:

- RTMP (Real-Time Messaging Protocol) – старіший протокол, який часто використовувався для стрімінгу, але не має вбудованої підтримки сучасного шифрування.
- HLS (HTTP Live Streaming) – сучасніший протокол від Apple, що дозволяє шифрувати відео та передавати його частинами (chunked encoding). HLS підтримує адаптивну якість і роботу з DRM.

Для захисту вмісту відео найчастіше використовується алгоритм симетричного шифрування AES (Advanced Encryption Standard) [7, 8]. Наприклад, у HLS можна застосовувати AES-128 для кожного фрагмента відео. У браузерах використовується Encrypted Media Extensions (EME) – інтерфейс, який дозволяє браузеру працювати з DRM і декодувати зашифроване відео без втрати безпеки [5, 7].

Перевага AES – висока швидкість обробки, що критично для потокового контенту. Проте важливо правильно організувати керування ключами шифрування, інакше захист буде марним. Оскільки асиметричне шифрування неефективне для великих обсягів відео, його застосовують лише для передачі ключів, тоді як саме відео обробляється виключно симетричними методами. У більшості практичних реалізацій для захисту відео застосовуються гібридні криптосистеми, де сам відеопотік шифрується швидким симетричним алгоритмом AES, а ключ до нього передається за допомогою RSA або іншого асиметричного методу [18].

Щоб підтвердити автентичність відео (наприклад, у системах відеоспостереження чи судових доказах), використовується [5. 10, 12, 19]:

- Цифровий підпис відеофайлу або кожного кадру.
- Хешування відео або окремих блоків (SHA-256, SHA-3).

Це дає змогу перевірити, чи не було змінено відео в процесі зберігання чи передавання. Для автоматизованої перевірки автентичності та цілісності відеофайлів широко використовуються бібліотеки криптографії на кшталт cryptography у Python, які підтримують створення та валідацію цифрових підписів [19].

Водяні знаки вбудовуються у відео для визначення джерела витоку [5]. Існує:

- Статичний watermark – логотип, який видно під час перегляду.
- Динамічний watermark – приховані маркери, які ідентифікують кожного користувача (наприклад, ID, IP, сесію).

Це унікальна мітка, яка не відображається користувачу, але дозволяє ідентифікувати джерело витоку. Часто використовується у медіасервісах, щоб виявити, хто з користувачів записав або викрав відео.

Це нова концепція безпеки, яка базується на принципі “довіра відсутня за замовчуванням” [5]. Кожна дія з відео – завантаження, перегляд, шифрування, кешування – вимагає перевірки автентичності та авторизації. Це дозволяє:

- мінімізувати ризики внутрішніх загроз;
- відстежувати будь-який доступ до відео;
- адаптувати політики безпеки до конкретного пристрою або локації.

Захист відеоконтенту є складним багаторівневим завданням, яке охоплює як технічні засоби (шифрування, DRM, watermarking, цифровий підпис), так і організаційні заходи (контроль доступу, аудит, політики управління контентом). В умовах стрімкого зростання обсягів відео та поширення технологій цифрової фальсифікації, зокрема deepfake, ключовими стають не лише запобігання несанкціонованому копіюванню, а й забезпечення достовірності джерела, автентичності та цілісності відео. У цьому контексті доцільним є впровадження блокчейн-технологій, які завдяки незмінності записів та децентралізованому зберіганню дозволяють надійно фіксувати інформацію про кожну транзакцію з відеофайлом – завантаження, доступ, зміну або поширення [1, 6, 15]. Це особливо важливо для контенту з обмеженим доступом, оскільки блокчейн

забезпечує прозорий журнал дій, виключає можливість прихованого редагування або підробки, а також дозволяє реалізувати механізми контролю прав доступу без посередників. Додатково, у блокчейн можна записувати хеші ключових подій, наприклад, момент створення або зміни відеофайлу, що забезпечує доказовість та юридичну значущість запису. Таким чином, комбінація DRM, криптографічного захисту та блокчейну створює стійку до атак інфраструктуру для безпечного розповсюдження відео, яка адаптована до сучасних загроз і може масштабуватися відповідно до потреб різних галузей.

Висновки до розділу 1

У першому розділі було здійснено комплексний аналіз основних форматів цифрових повідомлень – текстових, графічних, аудіо- та відеоданих – з точки зору їхніх вразливостей у процесі передавання через мережу. З’ясовано, що кожен тип повідомлень має специфічні загрози, пов’язані з порушенням конфіденційності, цілісності та автентичності даних. Особливу увагу приділено фішинговим атакам, витоку метаданих, маніпулюванню мультимедійним контентом та використанню централізованих сервісів, які створюють додаткові ризики через можливу компрометацію серверів.

Крім того, було акцентовано на обмеженнях традиційних підходів до захисту інформації, таких як шифрування та цифрові підписи, які, хоча й залишаються ефективними, не усувають вразливості, притаманні централізованій архітектурі систем. У зв’язку з цим обґрунтовано актуальність використання блокчейн-технологій, що завдяки децентралізованій природі, криптографічному захисту та незмінності записів, створюють нові можливості для безпечного обміну повідомленнями без потреби в довірених посередниках. Отже, зроблено висновок, що ефективна система захисту має ґрунтуватися на поєднанні сучасних криптографічних методів і блокчейн-підходів, які здатні забезпечити високий рівень довіри та інформаційної стійкості.

РОЗДІЛ 2
ПРОГРАМНЕ РІШЕННЯ ЗАХИСТУ ОБМІНУ ДАНИМИ З ВИКОРИСТАННЯМ БЛОКЧЕЙН-ТЕХНОЛОГІЙ

2.1 Опис алгоритмів симетричного та асиметричного шифрування

У сучасній криптографії ключову роль відіграють два базових підходи до захисту інформації – симетричне та асиметричне шифрування. Незважаючи на спільну мету – забезпечити конфіденційність, цілісність і автентичність даних, – ці методи відрізняються як за математичною природою, так і за сценаріями використання [5, 11].

Симетричні алгоритми базуються на використанні одного спільного секретного ключа для процесів шифрування і дешифрування. Це означає, що як відправник, так і одержувач повинні мати доступ до однакового ключа, причому передача цього ключа має здійснюватися захищеним способом.

Серед класичних симетричних алгоритмів можна виділити:

- AES (Advanced Encryption Standard) – сучасний стандарт шифрування з довжиною ключа 128, 192 або 256 біт [7];
- DES (Data Encryption Standard) – застарілий, але історично важливий алгоритм [5];
- Fernet – високорівневий симетричний алгоритм, який забезпечує як шифрування, так і автентифікацію повідомлень [24, 26].

Процес шифрування симетричним методом можна описати формулою:

$$C = E_K(M), \tag{2.1}$$

де:

- M – відкритий текст (оригінальне повідомлення),
- K – секретний ключ,
- E_K – функція шифрування з використанням ключа K,
- C – зашифроване повідомлення (шифротекст).

Процес дешифрування відповідно виконується так:

$$M = D_K(C)\,, \tag{2.2}$$

де D_K – функція дешифрування з тим самим ключем K.

Симетричне шифрування має ряд суттєвих переваг [5, 7]:

- висока швидкість обробки даних;
- ефективність для великих обсягів інформації (наприклад, медіа);
- мінімальні ресурси для реалізації.

Однак основним недоліком є проблема розповсюдження ключа: якщо третя сторона перехопить ключ під час передачі, безпека системи буде повністю скомпрометована.

На противагу симетричному, асиметричне шифрування використовує пару ключів – публічний (який можна вільно розповсюджувати) і приватний (який зберігається в таємниці) [3, 9]. Цей підхід дозволяє розв’язати проблему обміну ключами без необхідності їх передачі по захищених каналах.

До найвідоміших алгоритмів асиметричного шифрування належать:

- RSA (Rivest–Shamir–Adleman) – алгоритм на основі задачі факторизації великих чисел [13];
- ElGamal – алгоритм, що базується на складності логарифмування в скінчених полях [11].

Асиметричне шифрування базується на математичних функціях, які легко обчислюються у прямому напрямку, але складно обертаються без знання секретного ключа.

Шифрування: $C = E_{K_{pub}}(M)$

Дешифрування: $M = D_{K_{priv}}(C)$

де:

- K_{pub} – публічний ключ,
- K_{priv}– приватний ключ,
- E – функція шифрування,
- D – функція дешифрування.

У випадку RSA ці процеси реалізуються за допомогою наступних формул [9, 10, 12, 18]:

$$C = M^e \ mod \ n \ , \tag{2.3}$$

$$M = C^d \ mod \ n, \tag{2.4}$$

де

- e, n – публічний ключ,
- d – приватний ключ,
- M – початкове повідомлення,
- C – шифротекст.

Асиметричне шифрування має такі переваги:

- захищений обмін ключами без попередньої домовленості;
- можливість створення електронного підпису;
- підвищена стійкість до перехоплення.

Проте існують і суттєві обмеження:

- низька швидкість обробки даних порівняно з симетричними алгоритмами;
- обмеження розміру повідомлення, яке можна безпосередньо зашифрувати;
- високе споживання ресурсів, особливо для великих ключів.

Порівняльний аналіз зображено в таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз симетричного та асиметричного шифрування

Критерій	**Симетричне шифрування**	**Асиметричне шифрування**
Ключі	Один спільний ключ	Пара публічного та приватного
Швидкість	Висока	Низька
Захист передавання	Необхідний захищений канал	Захист вбудований у метод
Застосування	Захист файлів, потоків	Обмін ключами, підпис

2.2 Використання RSA-шифрування для захисту текстових повідомлень

Одним із найпоширеніших і водночас найнадійніших методів забезпечення конфіденційності текстових повідомлень є використання криптосистеми з відкритим ключем, зокрема алгоритму RSA. Цей метод широко застосовується в електронному листуванні, захисті особистих даних, електронному підписі та цифровій ідентифікації. Його ефективність базується на складності розв'язання задачі факторизації великих простих чисел, що є фундаментально важливою властивістю для забезпечення криптографічної стійкості [13].

RSA (названий за ініціалами авторів – Rivest, Shamir і Adleman) є прикладом асиметричного шифрування [5, 12], тобто такого, де для шифрування та розшифрування використовуються різні ключі: відкритий (public key) і закритий (private key). Оскільки відкритий ключ можна вільно поширювати, а приватний – зберігається в таємниці, такий підхід дозволяє безпечно передавати зашифровані повідомлення, навіть у відкритих або ненадійних каналах зв'язку. Стандарти управління ключами в таких системах регламентуються рекомендаціями NIST (*NIST Special Publication 800-57*).

Алгоритм RSA складається з кількох етапів: генерації ключів, шифрування повідомлення і його подальшого розшифрування. Розглянемо кожен з них детальніше.

Генерація ключів

Для початку необхідно згенерувати пару ключів. Цей процес включає в себе:

- Вибір двох великих простих чисел p та q;
- Обчислення модуля n=p·q, який буде використовуватись у подальших обчисленнях;
- Визначення функції Ейлера $\varphi(n) = (p-1)(q-1)$;
- Вибір числа e, що є взаємно простим з φ(n) (зазвичай e=65537);

- Обчислення секретного експонента d, який є мультиплікативно оберненим до e за модулем φ(n), тобто $d \cdot e \equiv 1 \, mod \, \varphi(n)$.

Таким чином, відкритий ключ утворюється як пара e, n, а закритий ключ – це d, n.

Шифрування

Нехай m – це повідомлення, яке слід зашифрувати. Спочатку його потрібно перетворити у числове представлення (наприклад, за допомогою кодування UTF-8 або ASCII). Потім повідомлення шифрується за наступною формулою [9, 10, 18]:

$$c = m^e \bmod n, \tag{2.5}$$

де:

- m – вихідне повідомлення у вигляді числа;
- e – експонента відкритого ключа;
- n – модуль.

Результатом обчислення є зашифроване повідомлення ссс, яке можна передавати по незахищеному каналу.

Розшифрування

Для розшифрування повідомлення отримувач використовує свій закритий ключ. Формула розшифрування виглядає так [12, 18]:

$$m = c^d \bmod n, \tag{2.6}$$

де:

- c – зашифроване повідомлення;
- d – приватна експонента;
- n – модуль.

Таким чином, початкове повідомлення m відновлюється у числовій формі, після чого декодується у текстовий формат.

Асиметрична криптосистема RSA має низку переваг, серед яких [5, 11]:

- Висока безпека: складність зламу ґрунтується на проблемі факторизації, що не має ефективного вирішення на класичних комп’ютерах.

- Безпечний обмін ключами: оскільки лише відкритий ключ передається мережами, приватний залишається недоступним третім особам.
- Можливість цифрового підпису: використовується не лише для шифрування, але й для автентифікації [13].

Проте, варто зазначити і недоліки [4, 11], головним з яких є невисока швидкість обчислень, особливо при роботі з великими обсягами даних. Саме тому RSA рідко використовується для шифрування файлів, зображень або відео – для цього застосовуються гібридні підходи, які поєднують симетричне та асиметричне шифрування.

2.3 Гібридне шифрування для захисту зображень та відео

У сучасних умовах, коли інформаційні технології стрімко розвиваються, зростає й потреба у захисті великих обсягів медіаданих, зокрема, зображень та відеофайлів. Це підтверджують також новітні дослідження з тематики захисту мультимедіа. Такі дані мають значно більший обсяг порівняно з текстовою інформацією, внаслідок чого застосування класичних асиметричних алгоритмів шифрування, таких як RSA, стає технічно недоцільним. Основною причиною є висока обчислювальна складність та повільна швидкодія RSA при роботі з великими об’ємами бітів.

У зв’язку з цим широкого застосування набули гібридні криптографічні системи, які поєднують переваги як симетричних, так і асиметричних методів. Такий підхід дозволяє ефективно забезпечити як високу швидкість шифрування, так і надійність розподілу ключів між сторонами комунікації [18, 24, 26, 27].

Гібридне шифрування – це стратегія, при якій для безпосереднього шифрування великого обсягу даних (наприклад, зображень чи відео) використовується симетричний алгоритм (наприклад, AES або Fernet), а для захисту самого симетричного ключа застосовується асиметричний алгоритм, найчастіше RSA [18, 24, 26]. Такий підхід забезпечує баланс між швидкістю обробки та надійністю зберігання секретного ключа. Такий підхід активно

застосовується на практиці, зокрема в системах із підвищеними вимогами до безпеки, де необхідна гарантія цілісності на рівні кожного повідомлення.

Іншими словами, користувач формує випадковий симетричний ключ, шифрує ним зображення або відеофайл, після чого цей симетричний ключ шифрується за допомогою відкритого RSA-ключа одержувача. У результаті:

- зловмисник не може відновити вміст медіафайлу без знання симетричного ключа;
- навіть якщо шифротекст стає відомим, отримати ключ без знання приватного RSA-ключа неможливо;
- для обох сторін зберігається гнучкість і сумісність з уже наявною криптографічною інфраструктурою.

Алгоритм гібридного шифрування

Загальна схема процесу гібридного шифрування складається з таких етапів:

1) Генерація симетричного ключа (K):

Сторона-відправник формує одноразовий симетричний ключ для шифрування медіаданих. Цей ключ є випадковим рядком байтів, придатним для використання в алгоритмах, а саме Fernet.

2) Шифрування медіаданих:

Відповідно до алгоритму симетричного шифрування, вихідні дані MMM (наприклад, зображення) шифруються за допомогою ключа KKK, внаслідок чого утворюється зашифрований файл $C_M = Enc_K(M)$.

3) Шифрування ключа K за допомогою RSA:

Симетричний ключ шифрується відкритим ключем одержувача (e,n) за формулою:

$$C_K = K^e \bmod n, \tag{2.7}$$

де C_K – зашифрований ключ, що може бути надісланий через відкритий канал.

4) Передача:

Одержувачу надсилаються два компоненти:

- зашифровані медіадані C_M,

- зашифрований симетричний ключ C_K

5) Розшифрування ключа:

Одержувач, маючи свій приватний ключ d, розшифровує симетричний ключ:

$$K = C_K^d \bmod n \tag{2.8}$$

6) Розшифрування повідомлення:

За допомогою відновленого ключа K відбувається дешифрування медіаданих:

$$M = Dec_K(C_M) \tag{2.9}$$

Таким чином, відновлюється початкове зображення або відеофайл.

Ілюстративна схема гібридного шифрування зображена на рис. 2.1.

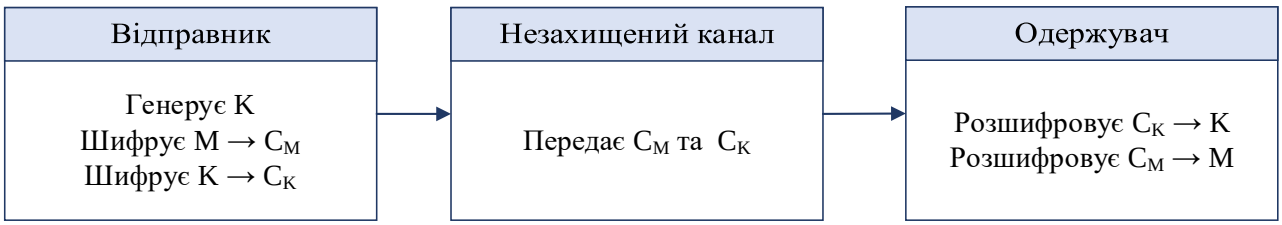

Рисунок – 2.1. Схема гібридного шифрування

Використання гібридного шифрування дозволяє досягти ряду важливих цілей:

Продуктивність (симетричне шифрування значно швидше за асиметричне, що робить його придатним для шифрування медіаданих великого розміру).

Безпека обміну ключами (передача симетричного ключа в зашифрованому вигляді гарантує, що тільки власник приватного ключа зможе його розшифрувати).

Масштабованість (система легко адаптується до будь-якого обсягу даних без суттєвого впливу на час обробки, оскільки основне шифрування виконується за допомогою високопродуктивних симетричних алгоритмів).

Це дозволяє ефективно працювати з великими файлами чи потоками даних.

2.4 Процес створення блокчейну для забезпечення цілісності даних

У контексті сучасних інформаційних систем надзвичайно актуальним є питання гарантування цілісності, достовірності та незмінності даних. Одним із найбільш ефективних підходів до вирішення цього завдання є впровадження блокчейн-технології, яка, завдяки своїй децентралізованій природі та криптографічній основі, забезпечує високий рівень захисту інформації від фальсифікацій і несанкціонованих змін.

Блокчейн – це послідовний ланцюг блоків, кожен з яких містить певний обсяг інформації та має посилання на попередній блок. Така структура формує незмінну хронологію подій, що дозволяє відслідковувати походження й історію змін даних.

Кожен блок, як правило, містить наступні компоненти:

- унікальний ідентифікатор (індекс);
- мітку часу створення;
- список транзакцій або повідомлень;
- криптографічний хеш попереднього блоку;
- доказ роботи (proof-of-work) – число, що підтверджує виконання певного обчислювального завдання.

Завдяки наявності хешу попереднього блоку, змінити один блок без порушення цілісності всього ланцюга практично неможливо, оскільки така зміна потребуватиме перерахунку хешів усіх наступних блоків.

Процес формування нового блоку в блокчейні включає кілька ключових етапів:

1. Накопичення транзакцій або повідомлень – користувачі надсилають повідомлення, які зберігаються у буфері до моменту створення нового блоку.

2. Обчислення доказу роботи (Proof-of-Work) – виконується пошук такого значення, яке, у поєднанні з попереднім доказом, дає хеш з певною кількістю початкових нулів. Це забезпечує складність і унеможливлює масове підроблення блоків.

3. Формування блоку – створюється новий блок з обчисленим доказом, поточним часом, хешем попереднього блоку та списком транзакцій.

4. Додавання блоку до ланцюга – новий блок приєднується до вже існуючого ланцюга, після чого транзакції вважаються підтвердженими.

5. Очищення буфера – список поточних транзакцій оновлюється, і система готується до прийому нових повідомлень.

Цей підхід забезпечує незворотність кожного запису: зміна будь-якого блоку автоматично порушує хеш усієї послідовності, внаслідок чого така маніпуляція легко виявляється.

Однією з ключових концепцій, що забезпечують захист блокчейну, є доказ роботи. Його метою є ускладнення процесу створення блоку, що запобігає атакам типу "переписування історії".

Суть методу полягає в знаходженні такого числа (proof), яке, разом із доказом з попереднього блоку (last_proof), формує хеш-функцію з певною кількістю початкових нулів. Формально цей процес можна описати як:

$$SHA - 256(last\_proof \parallel proof) = 0000\ldots xyz\ , \qquad (2.10)$$

де знак "||" позначає конкатенацію. Значення хешу повинно задовольняти умову складності (наприклад, починатися з двох чи більше нулів), встановлену мережею.

Найважливішою властивістю блокчейну є цілісність даних – жоден блок не може бути змінений без зміни всього ланцюга, що потребує переобчислення хешів усіх наступних блоків і повторного виконання ресурсоємної операції proof-of-work для кожного з них.

Цей механізм гарантує незмінність інформації та забезпечує стійкість до фальсифікацій навіть у середовищі з недовірою між учасниками, де не існує централізованого контролю.

Завдяки розподіленій природі блокчейну (у більш складних реалізаціях) кожен вузол зберігає копію всієї бази, і будь-яка невідповідність у ланцюзі одразу виявляється більшістю вузлів.

Це ускладнює здійснення атак, оскільки зловмиснику необхідно одночасно контролювати більшість учасників мережі, що на практиці є надзвичайно складним і малоймовірним.

2.5 Інтеграція блокчейн-технологій із криптографічною системою

Інтеграція блокчейну із криптографічними методами захисту набуває більшої популярності задля гарантування достовірності, незмінності та автентичності даних. Це дозволяє створити надійні системи безпеки для захисту цифрових повідомлень, документів, медіаданих та іншої важливої інформації.

Попри високу ефективність традиційних криптографічних алгоритмів, таких як RSA, AES або Fernet, вони не передбачають автоматичного підтвердження того, що дані не було змінено або підроблено після їх збереження чи передачі. Наприклад, навіть за умови використання шифрування, неможливо з абсолютною впевненістю довести, що повідомлення не було змінено зловмисником після розшифрування.

Натомість блокчейн створює незмінний цифровий реєстр транзакцій або повідомлень, який фіксує кожну зміну та дозволяє верифікувати її за допомогою криптографічних хешів. У поєднанні ці дві технології – криптографія та блокчейн – формують комплексну систему захисту даних, яка охоплює як їхню конфіденційність, так і цілісність.

Інтеграція передбачає взаємодію двох рівнів захисту:

1. Криптографічний рівень – відповідає за шифрування/дешифрування повідомлень або медіаданих. Для цього можуть використовуватись:
   - асиметричні алгоритми (RSA) – для захисту ключів або текстових повідомлень;
   - симетричні алгоритми (AES, Fernet) – для шифрування великих обсягів даних (наприклад, відео чи зображень).
2. Блокчейн-рівень – зберігає метаінформацію про зашифровані повідомлення: відправника, одержувача, тип даних, хеш-значення вмісту, мітку

часу тощо. Кожне повідомлення фіксується у новому блоці, який, завдяки механізму proof-of-work та хеш-зв’язку між блоками, гарантує незмінність записів.

Таким чином, навіть якщо зловмисник отримає доступ до зашифрованих даних, він не зможе змінити або замінити повідомлення без того, щоб порушити структуру блокчейну – що одразу буде виявлено мережею.

Інтеграція блокчейну з криптографією забезпечує кілька ключових переваг:

- Незмінність: після фіксації даних у блокчейні їх неможливо змінити без порушення цілісності всього ланцюга.
- Прозорість і аудит: кожен блок містить історію транзакцій, що дозволяє легко відстежити походження даних.
- Децентралізованість (у розширених моделях): дані не зберігаються централізовано, що знижує ризик атаки на один вузол.
- Довготривале збереження цілісності: навіть через роки можливо перевірити, що певне повідомлення залишилося незмінним з моменту його створення.
- Сумісність із сучасними криптопротоколами: інтеграція може бути реалізована з використанням існуючих стандартів криптографії (PKCS, JWT, тощо).

У типовій системі захисту даних із використанням блокчейну та криптографії процес виглядає наступним чином:

1) Користувач формує повідомлення або медіадані.

2) Дані шифруються за допомогою криптографічного алгоритму (наприклад, RSA або AES).

3) Результат шифрування (або його хеш) передається в блокчейн як транзакція.

4) Блокчейн зберігає всі необхідні метадані: хеш, відправника, одержувача, тип повідомлення та мітку часу.

5) Надалі будь-який користувач може перевірити справжність та незмінність зашифрованого повідомлення, порівнявши хеш із тим, що збережено в блокчейні.

Такі комбіновані системи вже знаходять застосування у різних сферах:

- у захисті електронної пошти та миттєвих повідомлень;
- у цифровій ідентифікації та перевірці особистості;
- у медицині – для захисту медичних зображень та історії хвороб;
- у юридичній сфері – для фіксації правочинів та договорів;
- у засобах масової інформації – для підтвердження автентичності новин, зображень і відео.

Висновки до розділу 2

У другому розділі було всебічно досліджено криптографічні підходи до захисту цифрових повідомлень, зокрема текстових та мультимедійних даних. Було акцентовано увагу на можливостях та обмеженнях алгоритму RSA, який, незважаючи на високу надійність, є малопридатним для обробки великих обсягів інформації. У зв’язку з цим обґрунтовано доцільність використання гібридного шифрування, що поєднує переваги симетричних алгоритмів (як-от AES та Fernet) із безпечним обміном ключами через RSA. Окремо розглянуто принципи функціонування симетричних та асиметричних алгоритмів, їхній вплив на ефективність і безпеку передачі даних.

Проаналізовано фундаментальні аспекти технології блокчейн як засобу забезпечення цілісності та незмінності інформації. Розглянуто механізми створення блоків, хешування та роль алгоритму proof-of-work, що забезпечують прозорість і стійкість блокчейн-структури.

На основі аналізу зроблено висновок про ефективність інтеграції блокчейн-компонентів із криптографічною системою для захисту даних і верифікації їхньої автентичності. Отримані результати стали підґрунтям для реалізації програмного рішення, описаного в наступному розділі.

РОЗДІЛ 3
ПРОГРАМНЕ РІШЕННЯ ЗАХИСТУ ОБМІНУ ДАНИМИ З ВИКОРИСТАННЯМ БЛОКЧЕЙН ТЕХНОЛОГІЙ

3.1 Архітектура програмного рішення

У ході розробки програмного забезпечення для захищеного обміну повідомленнями було створено повноцінну вебсистему, яка поєднує механізми автентифікації, асиметричного та симетричного шифрування, а також зберігання транзакцій у приватному блокчейні. Такий підхід дозволив не лише гарантувати конфіденційність даних, а й забезпечити цілісність та незмінність переданої інформації.

Загальна структура програмного рішення охоплює декілька ключових модулів, кожен із яких реалізовано у вигляді окремого Python-файлу. Всі модулі взаємодіють між собою через імпорти функцій і класів, створюючи цілісну програмну екосистему. Далі розглянемо архітектуру цього рішення та призначення основних складових.

Файл app.py: головний контролер вебзастосунку:

Цей файл відіграє центральну роль у проєкті, оскільки саме в ньому розгортається вебсервер на основі Flask. Файл app.py об’єднує логіку взаємодії з базою даних, обробку запитів користувача, механізми автентифікації та реєстрації, передачу та шифрування повідомлень, а також взаємодію з блокчейн-структурою.

На початку файлу відбувається імпорт необхідних бібліотек та модулів, зокрема: Flask, SQLAlchemy, LoginManager, а також власних модулів models, encryption, blockchain. Одразу ж ініціалізується об'єкт app, встановлюється секретний ключ, конфігурується база даних (SQLite) та запускається менеджер сесій авторизації.

Особливої уваги заслуговує інтеграція з класом Blockchain, який ініціалізується один раз і зберігається в оперативній пам’яті застосунку. Його методи використовуються під час кожної відправки повідомлення, коли потрібно додати нову транзакцію в ланцюг.

Ключові маршрути, визначені в app.py, включають:

- /register – обробляє створення нового облікового запису, включаючи генерацію RSA-ключів;
- /login та /logout – реалізують механізми автентифікації;
- /send_message – дозволяє користувачу створити повідомлення та передати його іншим зареєстрованим користувачам;
- /blockchain_view – надає інтерфейс перегляду стану блокчейну;
- /metrics – виводить статистичні графіки активності;
- /message_details – дозволяє переглянути зашифроване й розшифроване повідомлення.

Таким чином, app.py виступає центральним вузлом, який координує роботу інших модулів.

Файл models.py: моделі бази даних:

Усі сутності, які зберігаються в базі даних, описані в models.py. Зокрема, визначено дві основні моделі:

- User – зберігає інформацію про користувачів, включаючи ідентифікатор, логін, email, хеш пароля, а також публічний і зашифрований приватний RSA-ключ.
- Message – представляє повідомлення, що пересилаються між користувачами. Кожне повідомлення містить відправника, список отримувачів, тип (текст, зображення чи відео), оригінальний та зашифрований вміст, мітку часу й індекс блоку, до якого воно належить.

Таким чином, саме цей файл забезпечує сталість і структурованість зберігання даних, необхідних для роботи додатку.

Файл blockchain.py: реалізація приватного блокчейну:

Файл blockchain.py містить клас Blockchain, який реалізує приватний блокчейн, адаптований під локальне використання. Цей блокчейн не потребує децентралізованого консенсусу, однак підтримує основні принципи – незмінність, хронологічність та перевірку доказу роботи (proof-of-work).

Основні методи класу включають:

- new_transaction() – додає нове повідомлення як транзакцію;
- new_block() – створює новий блок із певним proof-of-work;
- proof_of_work() – реалізує класичний алгоритм добору доказу роботи;
- is_chain_valid() – перевіряє цілісність ланцюга блоків;
- save_chain() та load_chain() – дозволяють зберігати та відновлювати блокчейн з JSON-файлу.

Важливо, що кожне повідомлення, яке надсилається користувачем, фіксується у вигляді транзакції та записується в новий блок після проходження proof-of-work. Таким чином, досягається гарантія незмінності повідомлення навіть після його доставки.

Файл encryption.py: шифрування та дешифрування:

Цей модуль відповідає за реалізацію криптографічної складової системи. У ньому використовуються як асиметричні алгоритми (RSA), так і симетричне шифрування (Fernet). Це дає змогу реалізувати як класичне шифрування текстових повідомлень, так і гібридну схему для медіа-контенту.

Основні функції модуля:

generate_rsa_keys() – генерує пару відкритого та закритого ключа;

encrypt_message() / decrypt_message() – реалізують шифрування та розшифрування повідомлення з використанням RSA;

encrypt_private_key() / decrypt_private_key() – використовуються для шифрування приватного ключа користувача з допомогою головного ключа MASTER_KEY.

Особливої уваги заслуговує той факт, що приватний ключ кожного користувача не зберігається у відкритому вигляді. Завдяки шифруванню за

допомогою симетричного ключа, навіть при витоку бази даних зловмисник не зможе легко розшифрувати повідомлення.

Файл master_key.py: генерація головного ключа:

Цей невеликий скрипт використовується одноразово – для генерації ключа, який виступає у ролі "головного" (master key) при шифруванні приватних RSA-ключів користувачів. Застосовується бібліотека cryptography.fernet, яка створює 256-бітний ключ, придатний для надійного симетричного шифрування.

Надалі згенерований ключ копіюється в app.py у вигляді змінної MASTER_KEY.

Взаємозв’язок між модулями є чітким і структурованим: app.py взаємодіє з models.py для доступу до бази даних, з encryption.py – для шифрування/дешифрування даних, та з blockchain.py – для запису повідомлень у блокчейн. Кожен компонент виконує свою логічну функцію, що відповідає принципам модульного програмування. Завдяки цьому система залишається розширюваною, зручною для обслуговування та придатною до подальшої інтеграції з іншими платформами, переглянути код можна в додатку А.

3.2 Використані технології та інструменти

У процесі створення програмного рішення було використано низку сучасних технологій і програмних засобів, кожен з яких відіграє специфічну роль у реалізації функціональних можливостей системи. Вибір конкретних інструментів зумовлювався, з одного боку, їхньою надійністю, зручністю використання та широкою підтримкою в спільноті, а з іншого – необхідністю забезпечення високого рівня безпеки, продуктивності та масштабованості системи. У цьому підпункті детально розглянемо всі ключові технології, які було залучено в рамках проєкту.

Flask – серверна основа застосунку:

Основним фреймворком для побудови серверної частини є Flask – легкий мікрофреймворк для Python, що забезпечує гнучке і водночас потужне

середовище для створення вебдодатків [22, 23]. Завдяки мінімалістичній структурі, Flask дозволяє сконцентруватися на логіці програми, уникаючи надмірної складності.

Фреймворк використовується для створення маршрутів, обробки HTTP-запитів, а також для інтеграції HTML-шаблонів (через Jinja2) та обробки сесій користувача. Оскільки застосунок вимагає обробки форм (наприклад, при реєстрації або надсиланні повідомлень), Flask забезпечує зручні засоби для цього.

SQLAlchemy – ORM для взаємодії з базою даних:

З метою взаємодії з базою даних було використано SQLAlchemy – потужну ORM-бібліотеку для Python, яка дозволяє працювати з базою не напряму через SQL-запити, а через об'єктно-орієнтовані моделі [25]. Завдяки цьому підхід до роботи з даними стає більш безпечним, масштабованим і зручним у підтримці.

У нашому застосунку SQLAlchemy використовується для створення таблиць користувачів та повідомлень, організації зв'язків між ними, зберігання шифрованих даних, а також для виконання запитів при виведенні історії повідомлень, обробці логінів та фільтрації користувачів.

Flask-Login – керування автентифікацією:

Для реалізації механізмів автентифікації та керування сесіями було інтегровано розширення Flask-Login [27]. Це розширення дозволяє легко організувати входи/виходи користувачів, визначати рівні доступу, а також зберігати інформацію про активного користувача в контексті поточного запиту.

Завдяки Flask-Login ми можемо, наприклад, обмежити доступ до сторінки з повідомленнями лише для автентифікованих користувачів, а також автоматично здійснювати перенаправлення на сторінку входу при спробі доступу до захищеного ресурсу.

Cryptography – бібліотека для шифрування:

Безпека повідомлень є ключовою вимогою нашого застосунку. Саме тому для реалізації криптографічних функцій було обрано бібліотеку cryptography, яка

є однією з найпопулярніших у Python для симетричного та асиметричного шифрування [19, 24].

Використання цієї бібліотеки дозволило реалізувати такі механізми:

- Генерацію RSA-ключів для кожного користувача;
- Шифрування/дешифрування повідомлень з використанням RSA (текст) або гібридного підходу з RSA + Fernet (медіа-файли);
- Шифрування приватного ключа користувача для безпечного зберігання в базі даних.

Варто зазначити, що бібліотека підтримує сучасні криптографічні стандарти, включаючи SHA-256 та OAEP-падінг, що гарантує відповідність вимогам інформаційної безпеки.

Matplotlib – побудова графіків та метрик:

Для візуального аналізу активності користувачів у системі було інтегровано бібліотеку Matplotlib, яка використовується для побудови графіків [17]. Зокрема, реалізовано модуль /metrics, який відображає активність надсилання та отримання повідомлень у вигляді точкових діаграм.

Незважаючи на те, що Matplotlib частіше використовується у наукових проєктах, у нашому випадку вона дозволила швидко реалізувати вивід графічних метрик без необхідності використання сторонніх JS-бібліотек. Для побудови графіків було обрано темну тему з покращеним контрастом, що відповідає загальному стилю інтерфейсу.

Власна реалізація блокчейну:

Особливістю проєкту є наявність власного реалізованого з нуля приватного блокчейну, який функціонує локально та зберігається в оперативній пам’яті сервера [20, 21]. Блокчейн забезпечує незмінність та підтвердження кожного повідомлення, яке передається через систему.

Кожне повідомлення створює транзакцію, яка записується в блок. Блок формується лише після проходження процедури доказу роботи (proof-of-work), а його хеш обчислюється для підтвердження зв’язку з попереднім блоком. Таким чином, забезпечується логічна цілісність усієї історії обміну.

Програмна реалізація включає алгоритм SHA-256, а також базові методи для створення, перевірки та збереження блоків, які задовольняють мінімальні вимоги до безпечного ланцюга транзакцій.

3.3 Опис основних функцій програмного коду

Після розгляду архітектури застосунку та технологій, що були використані, доцільно зосередитися на детальному аналізі ключових функцій, реалізованих у програмному коді. Розуміння цих функцій дозволяє краще усвідомити логіку роботи системи, а також оцінити ефективність реалізації поставлених задач – зокрема, забезпечення конфіденційності, цілісності та доступності переданих повідомлень.

Реєстрація та вхід користувачів:

Процес реєстрації користувача реалізовано у маршруті /register. При надсиланні POST-запиту відбувається перевірка на унікальність логіна та email-адреси. Якщо користувач проходить перевірку, система генерує для нього пару ключів RSA: відкритий і приватний. Приватний ключ одразу ж шифрується симетричним алгоритмом (Fernet) з використанням заздалегідь збереженого MASTER_KEY. Після цього всі дані зберігаються в базу даних [19, 24, 26].

Процес входу реалізовано у маршруті /login, де використовується бібліотека Flask-Login для обробки сесії [27]. Якщо наданий логін (або email) і пароль відповідають збереженим у базі, користувач авторизується і перенаправляється до панелі керування.

Таким чином, уже на цьому етапі застосунок забезпечує базову автентифікацію та захист приватного ключа.

Генерація ключів та механізми шифрування:

У модулі encryption.py реалізовано функції, які відповідають за генерацію криптографічних ключів, а також за шифрування та дешифрування повідомлень.

- generate_rsa_keys() – генерує пару RSA-ключів розміром 2048 біт у форматі PEM.

- encrypt_message() та decrypt_message() – виконують шифрування й розшифрування тексту з використанням відкритого/закритого ключа.
- encrypt_private_key() та decrypt_private_key() – реалізують шифрування приватного ключа користувача симетричним способом для зберігання в базі даних.

Ці функції використовуються під час реєстрації, надсилання повідомлень та перегляду отриманих повідомлень.

Відправлення повідомлень і їх шифрування:

Найбільш складна й водночас ключова логіка реалізована у функції send_message() у файлі app.py. Вона забезпечує як створення повідомлення, так і його зашифровану передачу через блокчейн.

Залежно від типу повідомлення (текст, зображення або відео), застосовуються різні алгоритми:

- Текстові повідомлення шифруються одразу відкритим ключем кожного отримувача за допомогою RSA [9, 10, 18]. Для ілюстрації процесу шифрування текстового повідомлення, на рисунку 3.1 показано покрокову логіку перетворення повідомлення у зашифрований вигляд з використанням RSA-алгоритму. Схема охоплює етапи отримання публічного ключа, шифрування повідомлення, перетворення у шістнадцятковий формат і передачу зашифрованого результату.

Рисунок 3.1 – Алгоритм шифрування текстового повідомлення з використанням публічного ключа RSA

- Медіа-файли (зображення, відео) перед передачею кодуються у формат base64, шифруються симетричним ключем (Fernet), а сам симетричний ключ – відкритим ключем одержувача. Такий гібридний підхід забезпечує як конфіденційність, так і ефективність [18, 24, 26]. На рисунку 3.2 представлено блок-схему, яка ілюструє повний процес шифрування мультимедійного повідомлення перед його збереженням у блокчейні.

Рисунок 3.2 – Алгоритм шифрування мультимедійного повідомлення з використанням гібридного підходу (Fernet + RSA)

У результаті створюється словник, де для кожного отримувача зберігається унікальний зашифрований блок даних, що потім серіалізується у формат JSON і зберігається як транзакція.

Запис повідомлень у блокчейн:

Після того як повідомлення зашифровано, воно передається до блоку за допомогою методу new_transaction() з модуля blockchain.py [20, 21]. Одна транзакція містить такі поля, як ID відправника, список отримувачів, тип повідомлення, його вміст та мітку часу.

На рисунку 3.3 представлено блок-схему, яка демонструє повний процес формування та додавання нового блоку до блокчейну. Схема охоплює етапи створення транзакції, обчислення доказу роботи (Proof-of-Work), формування нового блоку з відповідним хешем попереднього та його включення до ланцюга.

Рисунок 3.3 – Алгоритм формування транзакції та додавання нового блоку до блокчейн-ланцюга

Щойно транзакція створена, застосовується процедура proof-of-work, реалізована в методі proof_of_work(), що знаходить числовий доказ, який відповідає складності (difficulty = 2). Лише після цього створюється новий блок (new_block()), і він приєднується до ланцюга [20].

Цей механізм забезпечує логічну послідовність усіх дій та незмінність кожного повідомлення.

Перегляд і розшифрування повідомлень:

Функція message_details() дозволяє користувачеві переглянути конкретне повідомлення – як відправлене, так і отримане. Якщо користувач є відправником, система просто виводить оригінальний вміст. Якщо ж він є отримувачем, повідомлення розшифровується за допомогою:

- Дешифрування приватного ключа користувача (decrypt_private_key()),
- Отримання потрібного сегменту з JSON-структури повідомлення,
- Розшифрування повідомлення (текстового чи медіа) із застосуванням відповідного алгоритму.

Таким чином, лише власник приватного ключа має змогу прочитати вміст повідомлення, що повністю відповідає принципам конфіденційності.

Побудова графіків користувацької активності:

Маршрут /metrics відповідає за статистичну візуалізацію активності користувача. За допомогою бібліотеки matplotlib будується два графіки – для надісланих та отриманих повідомлень [17]. Дані можна переглядати за добу або за тиждень.

Це не лише дозволяє оцінити продуктивність роботи користувача, а й може слугувати індикатором аномальної активності або просто допоміжною функцією з точки зору зручності.

Відображення стану блокчейну:

Нарешті, маршрут /blockchain_view надає можливість переглянути весь ланцюг блоків, включаючи кількість транзакцій, середній час формування блоку, а також часову прив’язку до кожної транзакції.

Для обробки часових даних використовуються стандартні функції Python та бібліотека datetime. Таким чином, користувач або адміністратор може перевірити, чи ланцюг є логічно послідовним і чи відповідає очікуваній структурі.

3.4 Огляд можливостей програмного інтерфейсу

Розроблене програмне рішення призначене для кінцевого користувача, який не обов’язково має технічну освіту. Саме тому однією з важливих цілей стало створення інтуїтивно зрозумілого, зручного та водночас функціонального користувацького інтерфейсу. У цьому підпункті буде детально проаналізовано

основні можливості графічного інтерфейсу, доступні через веббраузер, а також їх зв’язок із внутрішньою логікою додатку.

Структура та навігація:

Інтерфейс побудований за класичним принципом багатосторінкового вебзастосунку. Головна сторінка (/) пропонує користувачу можливість зареєструватися або увійти в систему. Після автентифікації користувач потрапляє на головну панель (/dashboard), де доступний список інших зареєстрованих користувачів, яким можна надіслати повідомлення.

Меню системи логічно структуроване та містить навігаційні пункти:

- Надіслати повідомлення (/send_message)
- Історія повідомлень (/message_history)
- Перегляд блокчейну (/blockchain_view)
- Метрики (/metrics)
- Вихід із системи

Таке розміщення дозволяє швидко орієнтуватися в застосунку, оскільки кожна функціональна можливість має окрему сторінку.

Реєстрація та автентифікація:

Інтерфейс сторінки реєстрації (/register) є простим і зрозумілим(Рис. 3.4). Користувачу пропонується ввести логін, адресу електронної пошти та пароль. У разі введення вже існуючого логіна або email, система негайно повідомляє про це за допомогою повідомлень (flash), що виводяться на екран.

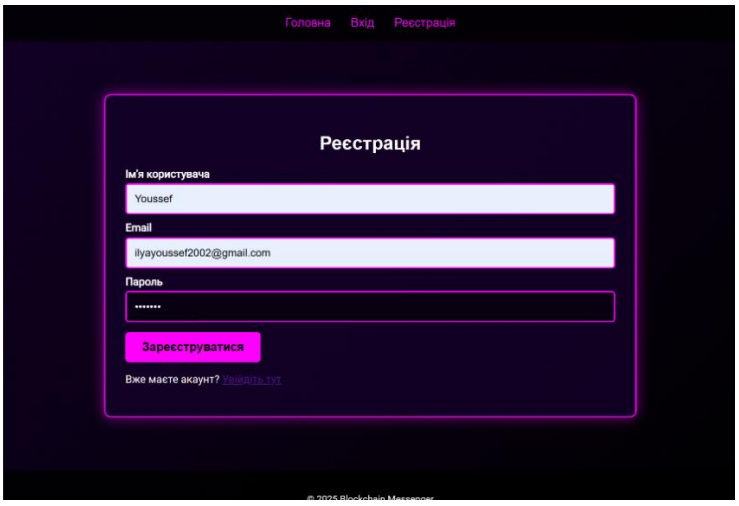

Рисунок 3.4 – Вікно реєстрації

Після реєстрації користувач автоматично перенаправляється на сторінку входу (/login), де він може використати логін або email для входу(Рис. 3.5). У разі неправильно введених облікових даних також виводиться відповідне повідомлення.

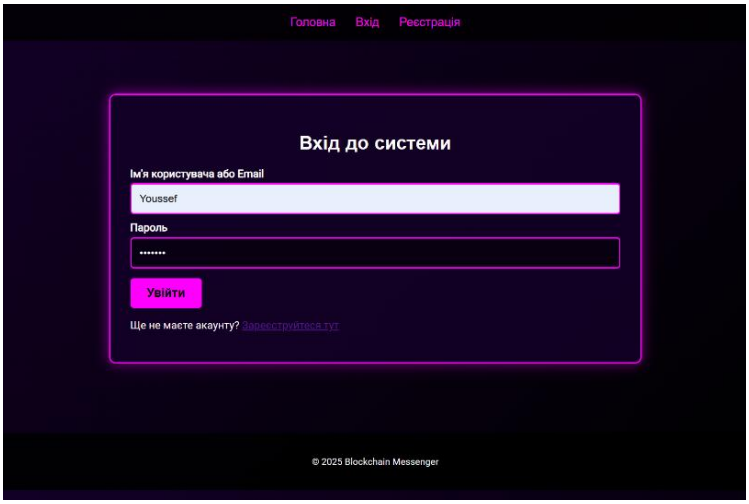

Рисунок 3.5 – Вікно входу в систему

Незважаючи на зовнішню простоту процесу обміну захищеними повідомленнями, за цим стоїть складна та багаторівнева криптографічна логіка. В основі лежать сучасні методи шифрування, такі як генерація пар ключів RSA, захищене зберігання приватного ключа, шифрування даних за допомогою симетричного алгоритму та передача ключів у зашифрованому вигляді. Усе це супроводжується внутрішніми перевірками автентичності, відповідності структури повідомлення та коректності дешифрування. Ці складні операції відбуваються у фоновому режимі, автоматично, що дозволяє мінімізувати вплив людського фактора та зменшує ймовірність помилок під час обміну інформацією [19, 24, 26].

Для кінцевого користувача взаємодія із системою виглядає максимально просто: достатньо лише надіслати або прийняти повідомлення, не замислюючись про складність внутрішніх процесів. Усі технічні деталі — від шифрування повідомлення до перевірки відповідності ключів — повністю приховані за

зручним інтерфейсом. Такий підхід не лише покращує загальне користувацьке враження, а й значно підвищує рівень безпеки, оскільки зменшує можливість помилкового втручання в критичні криптографічні операції.

Відправка повідомлень:

Одна з центральних сторінок інтерфейсу – /send_message – дозволяє надсилати повідомлення іншим користувачам. Форма складається з наступних елементів:

- поле для вибору одного або кількох одержувачів;
- вибір типу повідомлення (текст (Рис. 3.6), зображення (Рис. 3.7) або відео (Рис. 3.8));
- текстове поле для введення повідомлення або можливість прикріпити файл.

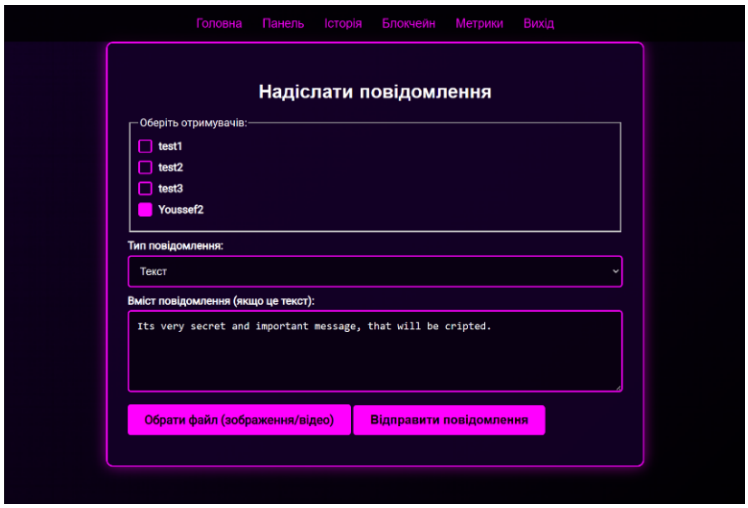

Рисунок 3.6 – Відправка текстового повідомлення

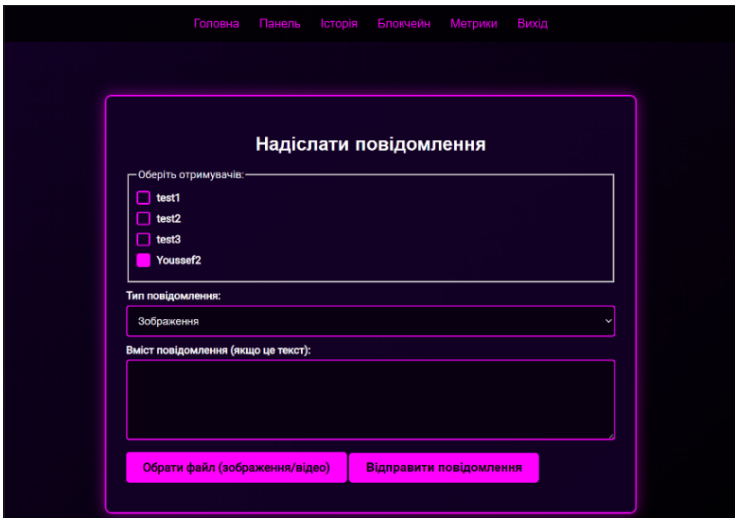

Рисунок 3.7 – Відправка зображення повідомлення

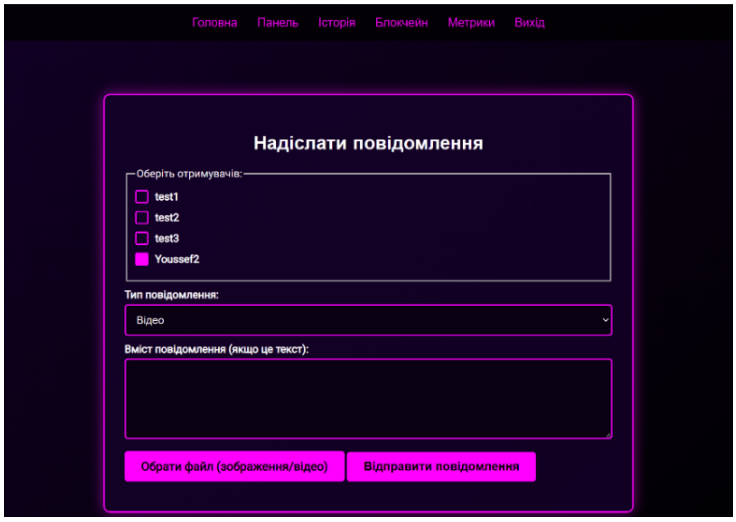

Рисунок 3.8 – Відправка відеофайлу повідомлення

Після натискання кнопки «Надіслати» виконується обробка введених даних, шифрування повідомлення (вибір типу шифрування залежить від типу вмісту), створення транзакції та запис у блокчейн [9, 10, 18]. Результат повідомляється користувачу у вигляді сповіщення: «Повідомлення відправлено і записано в блокчейн».

Важливо зазначити, що навіть при надсиланні файлів (зображень чи відео) весь процес відбувається у фоновому режимі – користувач бачить лише результат, а не складні криптографічні дії, які відбуваються у фоновому режимі.

Перегляд історії повідомлень:

Функціонал історії повідомлень реалізовано у вигляді таблиці на сторінці /message_history, де відображаються як надіслані, так і отримані повідомлення (Рис. 3.9). Користувач має змогу натиснути на будь-яке повідомлення, щоб переглянути його деталі на окремій сторінці.

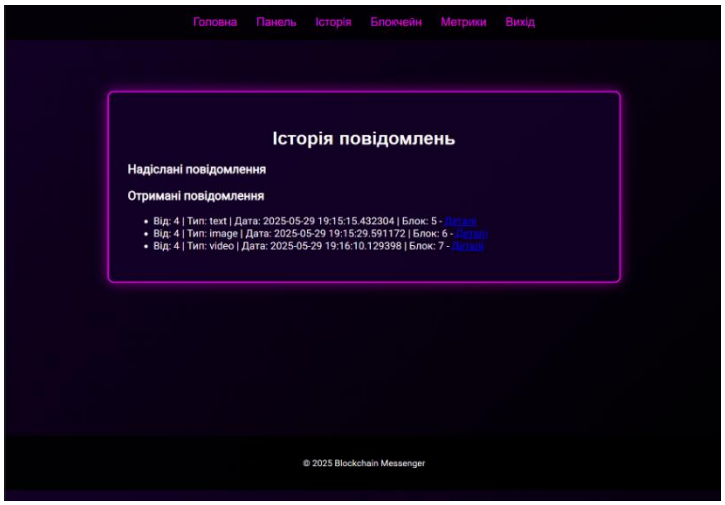

Рисунок 3.9 – Отримані повідомлення

У деталях повідомлення відображається наступна інформація:

- тип повідомлення (текст запис, графічне зображення, відеофайл);
- зашифроване повідомлення у вигляді JSON-структури;
- розшифроване повідомлення (Рис. 3.10 - 3.12) (лише для отримувача, за наявності правильного ключа);
- оригінальний вміст (для відправника).

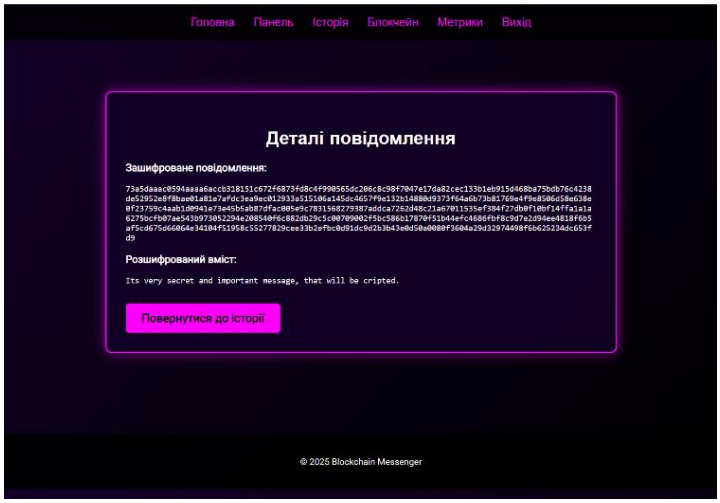

Рисунок 3.10 – Отримане текстове повідомлення

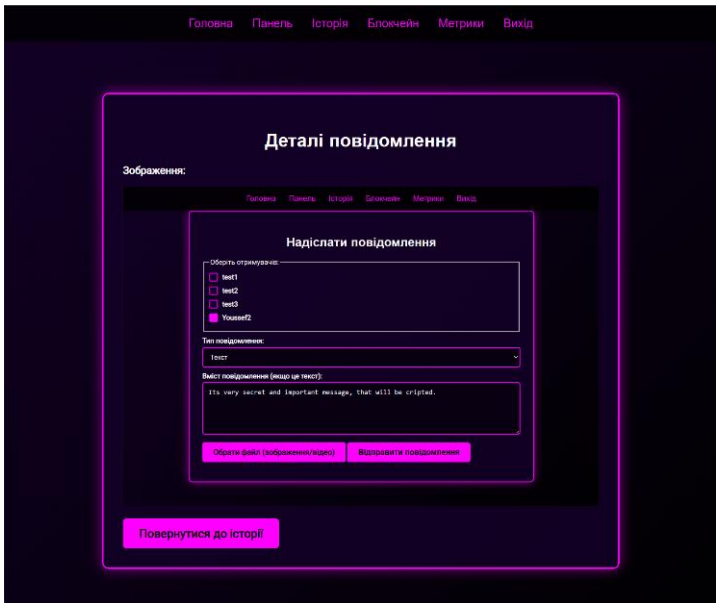

Рисунок 3.11 – Отримане зображення

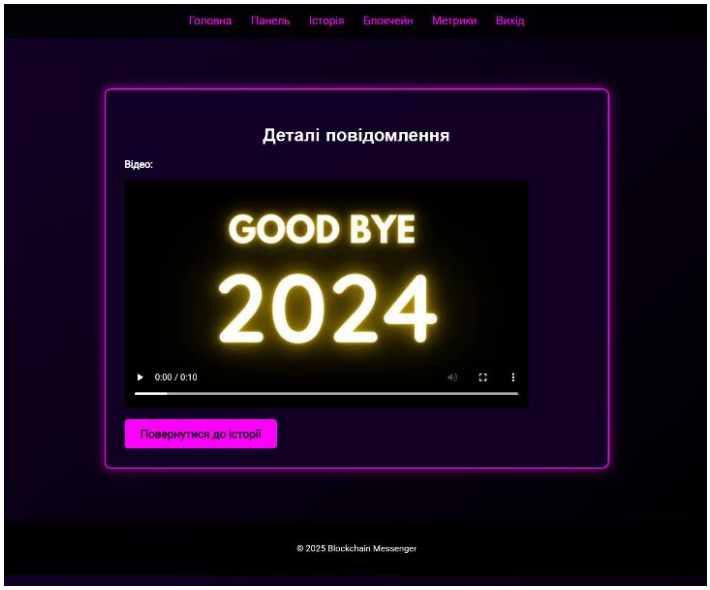

Рисунок 3.12 – Отриманий відеофайл

Таким чином, реалізується прозорий, контрольований і безпечний процес передачі інформації між учасниками системи, що дозволяє точно відстежувати кожен етап обробки даних — від шифрування до розшифрування.

У разі виникнення помилки під час дешифрування, наприклад, через використання неправильного ключа або пошкодження переданих даних, система автоматично сповістить користувача повідомленням «Не вдалося розшифрувати».

Цей механізм є важливим з погляду інформаційної безпеки, оскільки дозволяє своєчасно виявити порушення цілісності або спробу несанкціонованого доступу до захищеного вмісту. Більше про інтерфейс повідомлень в додатку Б.

Візуалізація блокчейну:

Сторінка /blockchain_view дозволяє користувачу переглянути всю структуру ланцюга блоків(Рис. 3.13) [20, 21]. У зручній таблиці виводиться інформація про кожен блок:

- індекс;
- кількість транзакцій;
- хеш попереднього блоку;
- доказ роботи (proof);
- часові мітки.

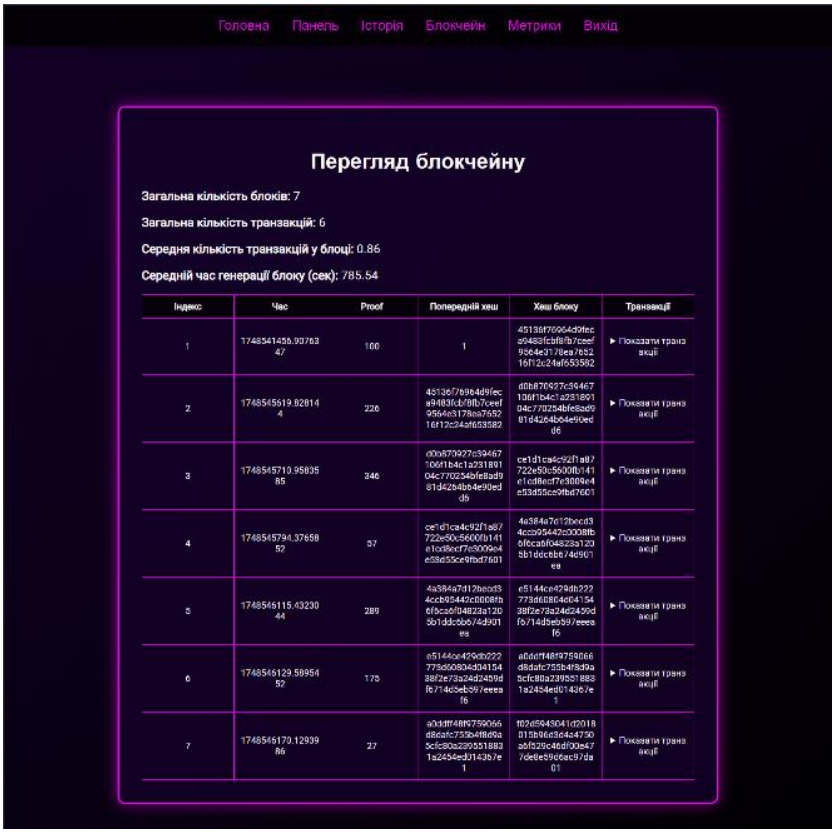

Рисунок 3.13 – Перегляд блокчейну

Крім того, внизу сторінки відображається статистична інформація: загальна кількість блоків, кількість транзакцій у ланцюгу, середня кількість

транзакцій на блок, а також середній час між створенням блоків. Це дозволяє не лише переглянути інформацію, але й зробити певні висновки щодо продуктивності та цілісності системи.

Графіки та метрики:

Останній елемент інтерфейсу – сторінка /metrics, яка реалізує вивід активності користувача у вигляді графіків [17]. Тут відображаються два графіки:

- надсилання повідомлень;
- отримання повідомлень.

Користувач може обрати період – за добу або за тиждень – після чого дані автоматично оновлюються. Графіки мають темне оформлення, що відповідає загальному стилю вебзастосунку, а також покращує контрастність та зменшує візуальне навантаження.

Цей елемент не є критично важливим для безпеки, однак сприяє підвищенню зручності користування та може допомогти в аналізі особистої активності.

Інтерфейс користувача розроблено з урахуванням принципів простоти, безпеки та функціональності. Кожна дія, яку може виконати користувач, чітко відображена в інтерфейсі, при цьому складна криптографічна логіка залишається прихованою. Це дозволяє зробити застосунок зручним для кінцевого користувача, не знижуючи рівень захисту даних. У підсумку, інтерфейс не лише виконує свою роль, а й підсилює довіру до системи, що особливо важливо в контексті захищеного обміну інформацією.

Для кращого розуміння логіки взаємодії користувача з системою, на рисунку 3.14 зображено алгоритм дій користувача – від реєстрації та автентифікації до надсилання повідомлення, вибору типу вмісту, застосування відповідного методу шифрування та фіксації в блокчейні.

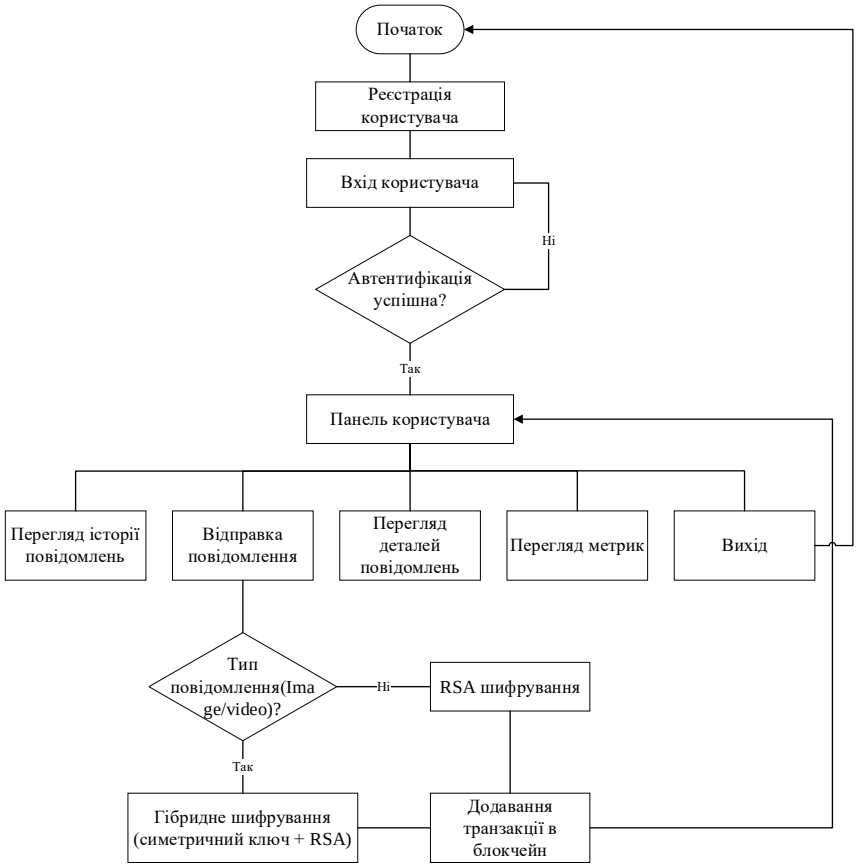

Рисунок 3.14 – Блок-схема взаємодії користувача з системою захищеного обміну повідомленнями на основі блокчейн-технологій

3.5 Дослідження впливу розміру повідомлень на продуктивність

У межах дослідження ефективності реалізованої системи було проведено експериментальну оцінку впливу розміру текстових повідомлень на її здатність успішно шифрувати, передавати та зберігати дані в блокчейні. Зокрема, було протестовано відправку повідомлень різної довжини за фіксованою кількістю повторень – кожне повідомлення надсилалося 20 разів для забезпечення репрезентативності результатів.

Дослідження охоплювало такі варіанти довжини текстових повідомлень: 100, 150, 170, 171, 172, 173, 174, 175, 180 та 200 символів. Варто зазначити, що повідомлення перед надсиланням проходили процес шифрування за допомогою алгоритму RSA, після чого у зашифрованому вигляді включалися до транзакцій у блокчейн.

У результаті експерименту було виявлено, що максимальна довжина повідомлення, яке вдалося успішно зашифрувати, зберегти у блокчейні та відправити без помилок, становить 173 символи. Повідомлення довжиною понад цей поріг – зокрема 174, 175, 180 та 200 символів – викликали помилки шифрування, через що їх обробка системою була неможливою.

Причини такої поведінки полягають у властивостях алгоритму RSA, що застосовується для шифрування повідомлень [5, 12]. Оскільки у даній реалізації використовується 2048-бітний ключ, максимальний розмір відкритого тексту, який можна зашифрувати за допомогою RSA з використанням схеми OAEP (Optimal Asymmetric Encryption Padding) та хеш-функції SHA-256, є обмеженим. Теоретично, цей розмір не перевищує приблизно 190–214 байтів, однак фактична межа залежить від конкретної реалізації бібліотеки та структури використовуваних символів. У випадку реалізованої системи, межа припадає саме на 173 символи, враховуючи додаткові байти, які використовуються для паддінгу та кодування.

Таким чином, дослідження показало критичну залежність функціональності системи від розміру вхідних повідомлень. У перспективі можливо розглянути оптимізацію схеми шифрування, наприклад, шляхом переходу до гібридного підходу, при якому повідомлення шифрується симетричним ключем, а сам ключ – за допомогою RSA. Це дозволить зняти обмеження на довжину повідомлень і покращити масштабованість системи.

3.6 Дослідження роботи метриків повідомлень

У рамках розробленої системи захищеного обміну повідомленнями передбачено механізм збору та візуалізації метрик активності користувачів. Такий підхід є особливо актуальним з огляду на потребу в моніторингу подій, виявленні аномальної поведінки та загальному аналізі безпечності функціонування системи. З цією метою було реалізовано функціонал, що

дозволяє будувати графіки активності на основі даних про надсилання та отримання повідомлень [17].

Метрики представлені у вигляді двох окремих графіків: один відображає динаміку надісланих повідомлень, а інший – отриманих повідомлень. Для зручності користувача передбачено перемикання між двома режимами перегляду: денним (щоденна активність) та тижневим (розподіл активності по днях тижня). На скріншотах нижче (Рис. 3.15 та 3.16) зображено приклади таких графіків у денному режимі. На скріншотах (Рис. 3.17 та 3.18) зображено приклади таких графіків у тижневому режимі.

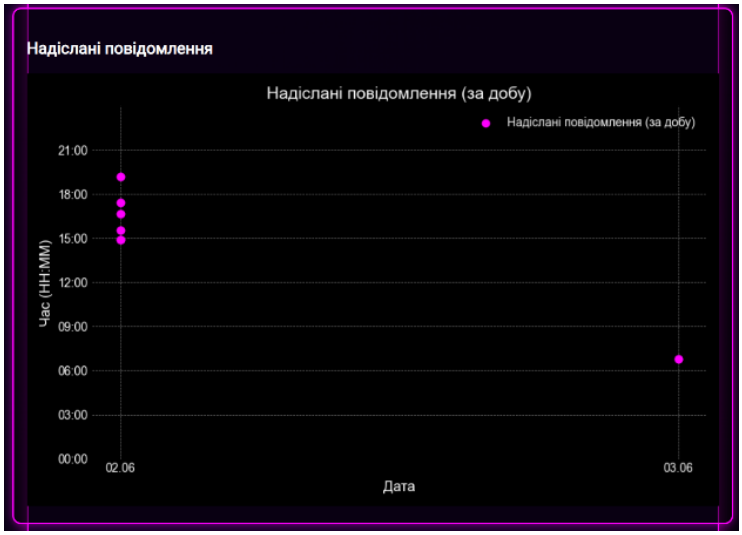

Рисунок 3.15 – Графік надсилання повідомлень у денному режимі

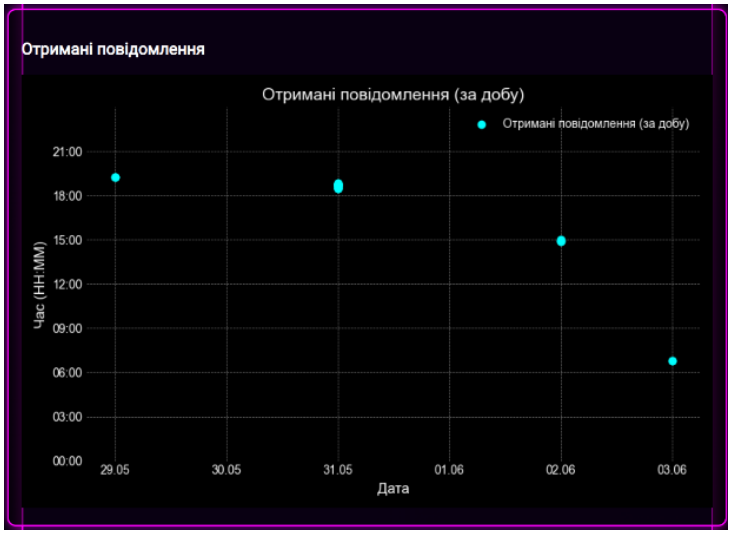

Рисунок 3.16 – Графік отримання повідомлень у денному режимі

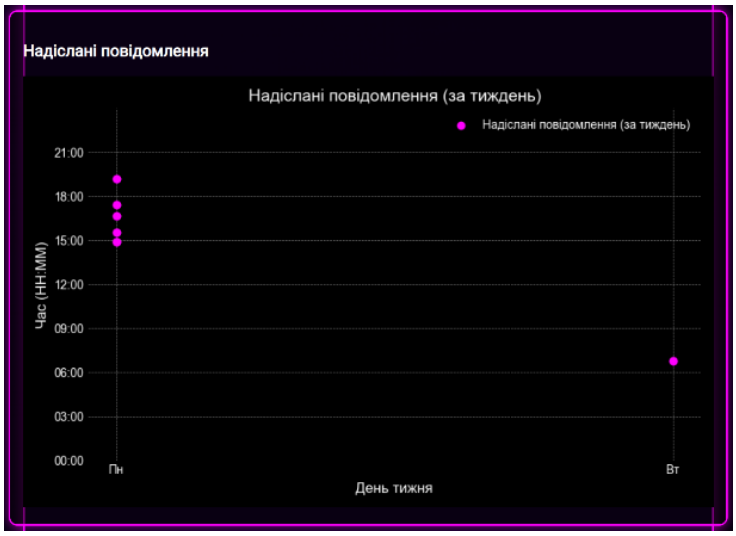

Рисунок 3.17 – Графік надсилання повідомлень у тижневому режимі

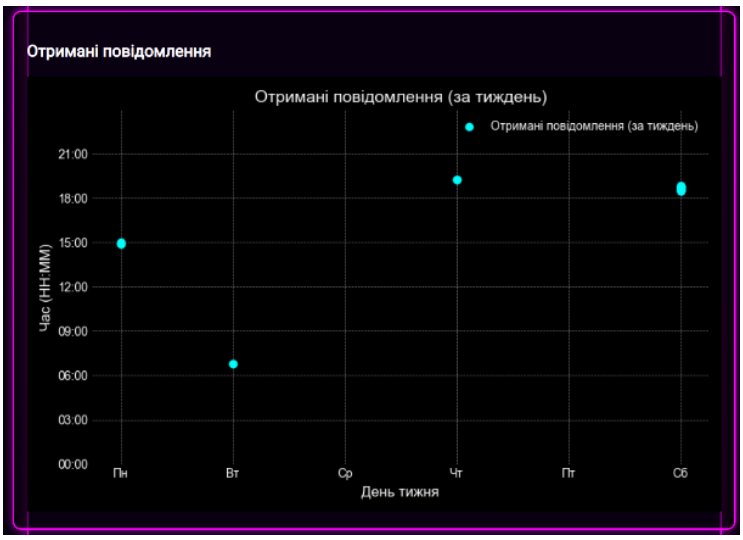

Рисунок 3.18 – Графік отримання повідомлень у тижневому режимі

Завдяки тому, що на вісі абсцис відображається дата, а на вісі ординат – точний час події, кожна точка на графіку відповідає конкретному моменту надсилання або отримання повідомлення. Така точність дозволяє з високою деталізацією аналізувати активність користувачів у системі, і що важливо – фіксувати часові мітки подій у вигляді зручної візуалізації.

На практиці це дає змогу ефективно вирішувати декілька задач. По-перше, системний адміністратор або фахівець з безпеки може використовувати графіки для виявлення несанкціонованих втручань у користувацькі облікові записи. Наприклад, якщо користувач зазвичай активний у робочі години (9:00–18:00), а

на графіку з’являються повідомлення, надіслані опівночі або в неробочі дні, це може вказувати на компрометацію облікового запису. По-друге, така інформація є корисною при аналізі внутрішньої активності, коли потрібно оцінити ефективність або навантаження на систему з боку окремих користувачів.

Крім того, у випадку інцидентів безпеки (наприклад, витоку даних чи спроби передачі забороненої інформації), графіки допомагають встановити точний часовий контекст події. Це значно прискорює розслідування та дозволяє сформувати доказову базу, прив'язану до конкретних подій та часу їх настання.

Не менш важливим є й той факт, що побудова графіків базується на об’єктивних даних, отриманих з бази повідомлень, що пройшли через механізм шифрування та були зафіксовані у блокчейні. Таким чином, виключається можливість їх фальсифікації або несанкціонованого редагування, що додатково підвищує рівень довіри до результатів аналізу.

Висновки до розділу 3

У третьому розділі було реалізовано програмне рішення, що поєднує в собі інструменти криптографії та блокчейн-технологій для безпечного обміну повідомленнями. Оскільки ефективність будь-якої системи інформаційного захисту визначається не лише її теоретичною обґрунтованістю, а й практичною реалізацією, саме цей етап дозволив перевірити життєздатність запропонованої архітектури в умовах наближених до реального використання. Зокрема, було створено інтерфейс користувача, який забезпечує шифрування, верифікацію та відображення повідомлень у блокчейн-ланцюгу, внаслідок чого досягнуто високого рівня прозорості та довіри до кожної транзакції. Незважаючи на певні технічні виклики система продемонструвала задовільні результати під час тестування. Отже, практична реалізація підтвердила доцільність обраного підходу, а також засвідчила, що поєднання гібридного шифрування та блокчейн-підходу є перспективним напрямом у створенні безпечних комунікаційних платформ.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досягнуто основну мету дослідження – розроблено вебзастосунок для захищеного обміну повідомленнями, який поєднує сучасні криптографічні механізми з децентралізованими блокчейн-технологіями. Це дозволило забезпечити високий рівень безпеки цифрової комунікації, зокрема – конфіденційність, цілісність, автентичність та незмінність переданої інформації.

У процесі дослідження проаналізовано існуючі підходи до захисту інформації при її передачі в мережі, а також виявлено специфічні загрози для різних типів цифрових повідомлень . Окрему увагу приділено аналізу актуальних кіберзагроз, таких як фішинг, спуфінг, ін’єкції шкідливого коду, атаки через метадані та технології deepfake. Унаслідок цього було встановлено, що централізовані системи захисту мають суттєві обмеження, пов’язані з втратою контролю над даними та ризиками компрометації серверів.

З метою подолання вказаних недоліків, у роботі розроблено криптографічні механізми: асиметричне шифрування на основі RSA для текстових повідомлень та гібридне шифрування на базі RSA і Fernet для мультимедійних даних. Такий підхід дозволив забезпечити високу ефективність обробки великих обсягів інформації та надійний захист передавання ключів.

Далі, у межах дослідження побудовано структуру приватного блокчейну, в якій кожне повідомлення зберігається у вигляді транзакції з хешем і часовою міткою. Це унеможливлює приховану зміну або видалення даних, забезпечуючи прозорість і незмінність інформації.

Крім того, створено вебзастосунок із застосуванням мови програмування Python, фреймворку Flask, а також бібліотек для реалізації криптографічних операцій і блокчейн-структур. Система включає функціональність генерації RSA-ключів, шифрування та дешифрування повідомлень, автентифікації користувачів та взаємодії з блокчейн-ланцюгом. Створено користувацький

інтерфейс, який забезпечує авторизацію, надсилання, перегляд і верифікацію повідомлень у захищеному середовищі.

Проведене тестування підтвердило працездатність системи та відповідність її архітектури вимогам сучасної інформаційної безпеки. Система гарантує збереження цілісності та автентичності повідомлень, запобігає несанкціонованому доступу й забезпечує незмінність усіх транзакцій завдяки фіксації їх у блокчейні.

Таким чином, усі поставлені в роботі задачі виконано, а мету дослідження досягнуто. Проаналізовано актуальні загрози та існуючі підходи до захисту інформації; розроблено криптографічні механізми шифрування; побудовано блокчейн-структуру для зберігання повідомлень; створено повноцінний інтерфейс користувача для взаємодії з системою.

Практична цінність роботи полягає у можливості застосування створеного рішення в таких сферах, як електронне урядування, банківська справа та корпоративна комунікація. Застосування open-source технологій забезпечує гнучкість, масштабованість та можливість розширення – зокрема, впровадження багатофакторної автентифікації чи мобільних клієнтів.

Незважаючи на виклики, пов’язані з обчислювальними витратами, управлінням ключами та підтримкою продуктивності при збільшенні кількості користувачів, запропонована модель залишається актуальною. У перспективі подальший розвиток таких систем може включати інтеграцію із зовнішніми сервісами ідентифікації (OAuth, eID), реалізацію наскрізного шифрування для групових чатів, а також застосування штучного інтелекту для перевірки автентичності файлів.

Отже, результати кваліфікаційної роботи підтверджують гіпотезу про те, що поєднання сучасних криптографічних механізмів із технологіями блокчейну дозволяє створити захищене, надійне та масштабоване середовище для обміну повідомленнями, яке не потребує централізованих довірчих вузлів і відповідає сучасним стандартам безпеки цифрових сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Л.В. Ковальчук, А.М. Кудін, Н.В. Кучинська Вступ до технології блокчейн та криптовалют: навч посіб. К.: Електронне мережне видання, 2022. 142 с.

2. Converging blockchain and next-generation artificial intelligence technologies to decentralize and accelerate biomedical research and healthcare.Baltimore / P. Mamoshina та ін. Insilico Medicine, 2017. 5665-5690 с.

3. Asymmetric Key Cryptography. Geeksforgeeks.org. 02.05.2024. URL: https://www.geeksforgeeks.org/asymmetric-key-cryptography/ (дата звернення: 02.05.2025).

4. Paar C., Pelzl J. Understanding Cryptography: A Textbook for Students and Practitioners.. Springer, 2009. 372 с.

5. Stallings W. Cryptography and Network Security: Principles and Practice. 7-ме вид. Pearson Education, 2016. 768 с.

6. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. / A. Narayanan та ін. Princeton University Press., 2016. 336 с.

7. Advanced Encryption Standard. Geeksforgeeks.org. 03.02.2025. URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/ (дата звернення: 08.05.2025).

8. Symmetric Key Cryptography. Geeksforgeeks.org. 02.05.2024. URL: https://www.geeksforgeeks.org/symmetric-key-cryptography/ (дата звернення: 02.05.2025).

9. RSA Algorithm in Cryptography. Geeksforgeeks.org. 06.01.2025. URL: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/ (дата звернення: 26.05.2025).

10. prof. Bernhard E. RSA (explained step by step). Cryptool.org. 09.03.2025. URL: https://www.cryptool.org/en/cto/rsa-step-by-step/ (дата звернення: 27.05.2025).

11. Stinson D. R. Cryptography: Theory and Practice.. CRC Press, 2005. 593 с.

12. Kinza Y. What is the RSA algorithm. Techtarget.com. Identity and access management. 11.02.2025. URL: https://www.techtarget.com/searchsecurity/definition/RSA (дата звернення: 27.05.2025).

13. Rivest R. L., Shamir A., Adleman L. A. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 1978. 126 с.

14. Ющенко Н. Л. Розвиток блокчейн-технологій в Україні та світі Економіка та суспільство. 2018. № 19. 269 – 275 с. DOI: https://doi.org/10.32782/2524-0072/2018-19-40.

15. Andreas M. Antonopoulos Mastering Bitcoin. O'Reilly Media, Inc., 2014. 282 с.

16. Documentation. Signal Messenger. URL: https://signal.org/docs/.

17. Matplotlib: Visualization with Python. Matplotlib.org. URL: https://matplotlib.org/ (дата звернення: 30.05.2025).

18. Tanvi B. RSA Algorithm: Theory and Implementation in Python. Askpython.com. 27.02.2023. URL: https://www.askpython.com/python/examples/rsa-algorithm-in-python (дата звернення: 27.05.2025).

19. Cryptography Library Documentation. Cryptography.io. URL: https://cryptography.io/en/latest/ (дата звернення: 29.05.2025).

20. Blockchain Hash Function. Geeksforgeeks.org. 13.10.2022. URL: https://www.geeksforgeeks.org/blockchain-hash-function/ (дата звернення: 29.05.2025).

21. Create simple Blockchain using Python. Geeksforgeeks.org. 27.11.2024. URL: https://www.geeksforgeeks.org/create-simple-blockchain-using-python/ (дата звернення: 29.05.2025).

22. Flask Documentation User's Guide. Flask.palletsprojects.com. URL: https://flask.palletsprojects.com/en/stable/ (дата звернення: 29.05.2025).

23. Flask Tutorial. Geeksforgeeks.org. 08.04.2025. URL: https://www.geeksforgeeks.org/flask-tutorial/ (дата звернення: 29.05.2025).

24. Fernet (symmetric encryption) using Cryptography module in Python. Geeksforgeeks.org. 28.09.2020. URL: https://www.geeksforgeeks.org/fernet-symmetric-encryption-using-cryptography-module-in-python/ (дата звернення: 26.05.2025).

25. SQLAlchemy Documentation. SQLAlchemy.com. URL: https://www.sqlalchemy.org/ (дата звернення: 22.05.2025).

26. Ashwin J. A Deep Dive Into Fernet Module in Python. Pythonistaplanet.com. 09.03.2025. URL: https://pythonistaplanet.com/fernet/ (дата звернення: 26.05.2025).

27. Flask-Login Documentation. Flask-login.readthedocs.io. URL: https://flask-login.readthedocs.io/en/latest/ (дата звернення: 29.05.2025).

ДОДАТКИ

Додаток А
Програмний код розробленого в кваліфікаційній роботі вебзастосунку для обміну повідомленнями

Файл app.py:

```
import os
import json
import binascii
import base64
from datetime import datetime
from flask import Flask, render_template, redirect, url_for, flash, request
from flask_sqlalchemy import SQLAlchemy
from flask_login import LoginManager, login_user, logout_user, login_required, current_user
from models import db, User, Message
from encryption import (generate_rsa_keys, encrypt_message, decrypt_message,
                        encrypt_private_key, decrypt_private_key)
from blockchain import Blockchain
from werkzeug.security import generate_password_hash, check_password_hash
from cryptography.fernet import Fernet
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as ticker
import io
import locale
from datetime import datetime, time

MASTER_KEY = "0Dn4nm68GGsuyLf9kn-ZCnIq_Emp4O1Gzc1y1iz3ATs="
app = Flask(__name__)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY', 'your-secret-key')
```

Продовження додатку А

```
    app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL',
'sqlite:///blockchain_messenger.db')
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False


    db.init_app(app)


    login_manager = LoginManager()
    login_manager.login_view = 'login'
    login_manager.init_app(app)


    blockchain = Blockchain()


    @app.context_processor
    def inject_hash_block():
        return dict(hash_block=lambda block: Blockchain.hash(block))


    @login_manager.user_loader
    def load_user(user_id):
        return db.session.get(User, int(user_id))


    @app.route('/')
    def index():
        return render_template('index.html')


    @app.route('/register', methods=['GET', 'POST'])
    def register():
        if request.method == 'POST':
            username = request.form['username']
            email = request.form['email']
            password = request.form['password']


            if User.query.filter_by(username=username).first():
                flash('Ім’я користувача вже існує.')
```

Продовження додатку А

```
            return redirect(url_for('register'))
        if User.query.filter_by(email=email).first():
            flash('Електронна адреса вже зареєстрована.')
            return redirect(url_for('register'))

        password_hash = generate_password_hash(password)
        private_key, public_key = generate_rsa_keys()
        encrypted_private_key = encrypt_private_key(private_key, MASTER_KEY)

        new_user = User(username=username, email=email, password_hash=password_hash,
                        public_key=public_key, private_key=encrypted_private_key)
        db.session.add(new_user)
        db.session.commit()

        flash('Реєстрація успішна. Будь ласка, увійдіть.')
        return redirect(url_for('login'))

    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username_or_email = request.form['username']
        password = request.form['password']
        user = User.query.filter((User.username == username_or_email) | (User.email ==
username_or_email)).first()
        if user and check_password_hash(user.password_hash, password):
            login_user(user)
            flash('Вхід успішний.')
            return redirect(url_for('dashboard'))
        else:
            flash('Невірні дані для входу.')
            return redirect(url_for('login'))
```

Продовження додатку А

```
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    flash('Ви вийшли з системи.')
    return redirect(url_for('index'))

@app.route('/dashboard')
@login_required
def dashboard():
    users = User.query.filter(User.id != current_user.id).all()
    return render_template('dashboard.html', users=users)

@app.route('/send_message', methods=['GET', 'POST'])
@login_required
def send_message():
    users = User.query.filter(User.id != current_user.id).all()
    if request.method == 'POST':
        recipient_ids = request.form.getlist('recipients')
        message_type = request.form['message_type']
        file = request.files.get('file')
        if file and file.filename != '':
            file_bytes = file.read()
            original_content = base64.b64encode(file_bytes).decode('utf-8')
        else:
            original_content = request.form.get('content', '')

        if message_type in ['image', 'video']:
            # Гібридне шифрування: симетричний ключ для зашифрованого base64 рядка
            sym_key = Fernet.generate_key()
            f_sym = Fernet(sym_key)
```

Продовження додатку А

```
        encrypted_data = f_sym.encrypt(original_content.encode())
        encrypted_messages = {}
        for rid in recipient_ids:
            recipient = User.query.get(int(rid))
            try:
                enc_sym_key = encrypt_message(recipient.public_key, sym_key.decode())
                payload = {
                    "key": binascii.hexlify(enc_sym_key).decode('utf-8'),
                    "data": base64.b64encode(encrypted_data).decode('utf-8')
                }
                encrypted_messages[rid] = payload
            except Exception as e:
                flash(f"Не вдалося зашифрувати повідомлення для користувача {recipient.username}.")
                continue
        combined_encrypted = json.dumps(encrypted_messages)
    else:
        # Текстове повідомлення – просте RSA-шифрування
        encrypted_messages = {}
        for rid in recipient_ids:
            recipient = User.query.get(int(rid))
            try:
                enc = encrypt_message(recipient.public_key, original_content)
                encrypted_messages[rid] = binascii.hexlify(enc).decode('utf-8')
            except Exception as e:
                flash(f"Не вдалося зашифрувати повідомлення для користувача {recipient.username}.")
                continue
        combined_encrypted = json.dumps(encrypted_messages)

    # Додаємо транзакцію в блокчейн
    block_index = blockchain.new_transaction(sender=current_user.id,
                                             recipients=recipient_ids,
                                             message_type=message_type,
                                             content=combined_encrypted)
```

Продовження додатку А

```
            last_proof = blockchain.last_block['proof']
            proof = blockchain.proof_of_work(last_proof)
            previous_hash = blockchain.hash(blockchain.last_block)
            block = blockchain.new_block(proof, previous_hash)

            new_message = Message(sender_id=current_user.id,
                                  recipients=",".join(recipient_ids),
                                  message_type=message_type,
                                  content=combined_encrypted,
                                  original_content=original_content,
                                  block_index=block['index'])
            db.session.add(new_message)
            db.session.commit()

            flash('Повідомлення відправлено і записано в блокчейн.')
            return redirect(url_for('dashboard'))
        return render_template('send_message.html', users=users)


    @app.route('/message_history')
    @login_required
    def message_history():
        sent_messages = Message.query.filter_by(sender_id=current_user.id).all()
        received_messages = Message.query.filter(Message.recipients.like(f"%{current_user.id}%")).all()
        return render_template('message_history.html', sent_messages=sent_messages,
received_messages=received_messages)


    @app.route('/message_details/<int:message_id>')
    @login_required
    def message_details(message_id):
        msg = Message.query.get_or_404(message_id)
        details = {}
        details['message_type'] = msg.message_type
```

Продовження додатку А

```
        if current_user.id == msg.sender_id:
            details['role'] = 'sender'
            details['original_content'] = msg.original_content
            details['encrypted_message'] = msg.content
            details['encrypted_key'] = current_user.private_key
            if msg.message_type in ['image', 'video']:
                details['media_data'] = msg.original_content
        else:
            details['role'] = 'receiver'
            enc_dict = json.loads(msg.content)
            enc_msg = enc_dict.get(str(current_user.id))
            if msg.message_type in ['image', 'video']:
                try:
                    payload = enc_msg
                    if isinstance(payload, str):
                        payload = json.loads(payload)
                    enc_sym_key_hex = payload.get("key")
                    encrypted_data_b64 = payload.get("data")
                    user_private_key = decrypt_private_key(current_user.private_key, MASTER_KEY)
                    dec_sym_key = decrypt_message(user_private_key,
binascii.unhexlify(enc_sym_key_hex.encode('utf-8')))
                    f_sym = Fernet(dec_sym_key.encode())
                    decrypted_data = f_sym.decrypt(base64.b64decode(encrypted_data_b64))
                    details['decrypted_message'] = decrypted_data.decode('utf-8')
                    details['media_data'] = decrypted_data.decode('utf-8')
                except Exception as e:
                    details['decrypted_message'] = "Не вдалося розшифрувати"
            else:
                details['encrypted_message'] = enc_msg
                try:
                    user_private_key = decrypt_private_key(current_user.private_key, MASTER_KEY)
                    decrypted_message = decrypt_message(user_private_key,
binascii.unhexlify(enc_msg.encode('utf-8')))
```

Продовження додатку А

```
            details['decrypted_message'] = decrypted_message
        except Exception as e:
            details['decrypted_message'] = "Не вдалося розшифрувати"
    return render_template('message_details.html', details=details, message=msg)

@app.route('/blockchain_view')
@login_required
def blockchain_view():
    chain = blockchain.chain

    times = []
    for block in chain:
        ts = block['timestamp']
        if isinstance(ts, (int, float)):
            times.append(datetime.fromtimestamp(ts))
        else:
            try:
                times.append(datetime.fromisoformat(ts))
            except Exception:
                times.append(datetime.now())

    total_blocks = len(chain)
    total_transactions = sum(len(block['transactions']) for block in chain)
    avg_transactions = total_transactions / total_blocks if total_blocks > 0 else 0

    if total_blocks > 1:
        time_diffs = [(t2 - t1).total_seconds() for t1, t2 in zip(times[:-1], times[1:])]
        avg_block_time = sum(time_diffs) / len(time_diffs)
    else:
        avg_block_time = 0

    return render_template('blockchain_view.html', chain=chain,
                           total_blocks=total_blocks, total_transactions=total_transactions,
```

Продовження додатку А

```
                avg_transactions=avg_transactions, avg_block_time=avg_block_time)

@app.route('/metrics')
@login_required
def metrics():
    try:
        locale.setlocale(locale.LC_TIME, 'uk_UA.UTF-8')
    except:
        pass

    view = request.args.get('view', 'daily')
    sent_messages = Message.query.filter_by(sender_id=current_user.id).all()
    received_messages = Message.query.filter(Message.recipients.like(f"%{current_user.id}%")).all()

    plt.style.use('seaborn-v0_8-dark')

    if view == 'daily':
        x_sent = [msg.timestamp.date() for msg in sent_messages]
        x_received = [msg.timestamp.date() for msg in received_messages]
        xlabel = 'Дата'
        x_fmt = mdates.DateFormatter('%d.%m')
    else:
        days = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Нд']
        x_sent = [msg.timestamp.weekday() for msg in sent_messages]
        x_received = [msg.timestamp.weekday() for msg in received_messages]
        xlabel = 'День тижня'
        x_fmt = ticker.FuncFormatter(lambda x, _: days[int(x)] if 0 <= int(x) <= 6 else '')

    # Вісь Y – час у вигляді datetime
    y_sent = [datetime.combine(datetime.today(), msg.timestamp.time()) for msg in sent_messages]
    y_received = [datetime.combine(datetime.today(), msg.timestamp.time()) for msg in
received_messages]
```

Продовження додатку А

```
        def create_graph(x, y, title, color):
            fig, ax = plt.subplots(figsize=(8, 5))
            ax.scatter(x, y, color=color, marker='o', label=title)

            ax.set_title(title, fontsize=14, color='white')
            ax.set_xlabel(xlabel, fontsize=12, color='white')
            ax.set_ylabel('Час (HH:MM)', fontsize=12, color='white')

            ax.grid(True, linestyle='--', linewidth=0.5, color='gray')
            ax.tick_params(axis='x', colors='white')
            ax.tick_params(axis='y', colors='white')

            # X formatting
            if view == 'daily':
                ax.xaxis.set_major_locator(mdates.DayLocator())
                ax.xaxis.set_major_formatter(x_fmt)
            else:
                ax.xaxis.set_major_locator(ticker.FixedLocator(range(7)))
                ax.xaxis.set_major_formatter(x_fmt)

            # Y formatting – лише час
            ax.yaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
            ax.set_ylim(datetime.combine(datetime.today(), time(0, 0)), datetime.combine(datetime.today(),
time(23, 59)))

            ax.legend(facecolor='#000', edgecolor=color, labelcolor='white')
            fig.patch.set_facecolor('#000')
            ax.set_facecolor('#000')

            plt.tight_layout()
            buf = io.BytesIO()
            plt.savefig(buf, format='png', facecolor=fig.get_facecolor())
            buf.seek(0)
```

Продовження додатку А

```
        encoded = base64.b64encode(buf.getvalue()).decode('utf8')
        plt.close(fig)
        return encoded

    graph_sent = create_graph(x_sent, y_sent, 'Надіслані повідомлення' + (' (за добу)' if view == 'daily' else ' (за тиждень)'), '#ff00ff')
    graph_received = create_graph(x_received, y_received, 'Отримані повідомлення' + (' (за добу)' if view == 'daily' else ' (за тиждень)'), '#00ffff')

    return render_template('metrics.html', graph_sent=graph_sent, graph_received=graph_received, view=view)

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

Файл blockchain.py:

```
import hashlib
import json
import time

class Blockchain:
    def __init__(self, difficulty=2):
        self.chain = []
        self.current_transactions = []
        self.difficulty = difficulty
        # Створюємо генезис-блок
        self.new_block(previous_hash='1', proof=100)

    def new_block(self, proof, previous_hash=None):
        """
```

Продовження додатку А

```
    Створює новий блок у ланцюзі.
    """
    block = {
        'index': len(self.chain) + 1,
        'timestamp': time.time(),
        'transactions': self.current_transactions,
        'proof': proof,
        'previous_hash': previous_hash or self.hash(self.chain[-1]),
    }
    self.current_transactions = []
    self.chain.append(block)
    return block

def new_transaction(self, sender, recipients, message_type, content):
    """
    Додає нову транзакцію (повідомлення) до списку поточних транзакцій.
    """
    self.current_transactions.append({
        'sender': sender,
        'recipients': recipients,
        'message_type': message_type,
        'content': content,
        'timestamp': time.time()
    })
    return self.last_block['index'] + 1

@staticmethod
def hash(block):
    """
    Обчислює SHA-256 хеш для даного блоку.
    """
    block_string = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_string).hexdigest()
```

Продовження додатку А

```
@property
def last_block(self):
    return self.chain[-1]


def proof_of_work(self, last_proof):
    """
    Алгоритм доказу роботи: знаходить число proof таке, що хеш з'єднання last_proof і proof містить потрібну кількість початкових нулів.
    """
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1
    return proof


def valid_proof(self, last_proof, proof):
    guess = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    return guess_hash[:self.difficulty] == "0" * self.difficulty


def is_chain_valid(self):
    """
    Перевіряє цілісність ланцюга блоків.
    """
    for i in range(1, len(self.chain)):
        prev = self.chain[i-1]
        curr = self.chain[i]
        if curr['previous_hash'] != self.hash(prev):
            return False
        if not self.valid_proof(prev['proof'], curr['proof']):
            return False
    return True
```

Продовження додатку А

```
    def save_chain(self, filename='chain.json'):
        """
        Зберігає ланцюг блоків у файл.
        """
        with open(filename, 'w') as f:
            json.dump(self.chain, f)

    def load_chain(self, filename='chain.json'):
        """
        Завантажує ланцюг блоків із файлу (якщо файл існує).
        """
        try:
            with open(filename, 'r') as f:
                self.chain = json.load(f)
        except FileNotFoundError:
            pass
```

Файл encryption.py:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization, hashes
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.fernet import Fernet

def generate_rsa_keys():
    """
    Генерує пару RSA-ключів (приватний та публічний) у форматі PEM (рядок).
    """
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
```

Продовження додатку А

```
    )
    public_key = private_key.public_key()

    pem_private = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()
    ).decode('utf-8')

    pem_public = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    ).decode('utf-8')

    return pem_private, pem_public

def encrypt_message(public_key_pem, message: str) -> bytes:
    """
    Шифрує повідомлення за допомогою публічного ключа.
    """
    public_key = serialization.load_pem_public_key(
        public_key_pem.encode('utf-8'),
        backend=default_backend()
    )
    ciphertext = public_key.encrypt(
        message.encode('utf-8'),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return ciphertext
```

Продовження додатку А

```
    def decrypt_message(private_key_pem, ciphertext: bytes) -> str:
        """
        Дешифрує повідомлення за допомогою приватного ключа.
        """
        private_key = serialization.load_pem_private_key(
            private_key_pem.encode('utf-8'),
            password=None,
            backend=default_backend()
        )
        plaintext = private_key.decrypt(
            ciphertext,
            padding.OAEP(
                mgf=padding.MGF1(algorithm=hashes.SHA256()),
                algorithm=hashes.SHA256(),
                label=None
            )
        )
        return plaintext.decode('utf-8')


    def encrypt_private_key(private_key: str, master_key: str) -> str:
        """
        Шифрує приватний RSA-ключ симетричним алгоритмом (Fernet) з використанням
MASTER_KEY.
        """
        f = Fernet(master_key)
        token = f.encrypt(private_key.encode())
        return token.decode()


    def decrypt_private_key(token: str, master_key: str) -> str:
        """
        Дешифрує симетрично зашифрований приватний RSA-ключ.
        """
```

Продовження додатку А

```
    f = Fernet(master_key)
    private_key = f.decrypt(token.encode())
    return private_key.decode()
```

Файл master_key.py:

```
from cryptography.fernet import Fernet
print(Fernet.generate_key().decode())
```

Файл master_key.py:

```
from flask_sqlalchemy import SQLAlchemy
from flask_login import UserMixin
from datetime import datetime
db = SQLAlchemy()
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    public_key = db.Column(db.Text, nullable=False)
    private_key = db.Column(db.Text, nullable=False)

class Message(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    sender_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    recipients = db.Column(db.String(200), nullable=False)
    message_type = db.Column(db.String(20), nullable=False)
    content = db.Column(db.Text, nullable=False)
    original_content = db.Column(db.Text, nullable=True)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
    block_index = db.Column(db.Integer, nullable=True)
```

Додаток Б
Рисунки, які містять деталі повідомлень в інтерфейсі вебзастосунку.

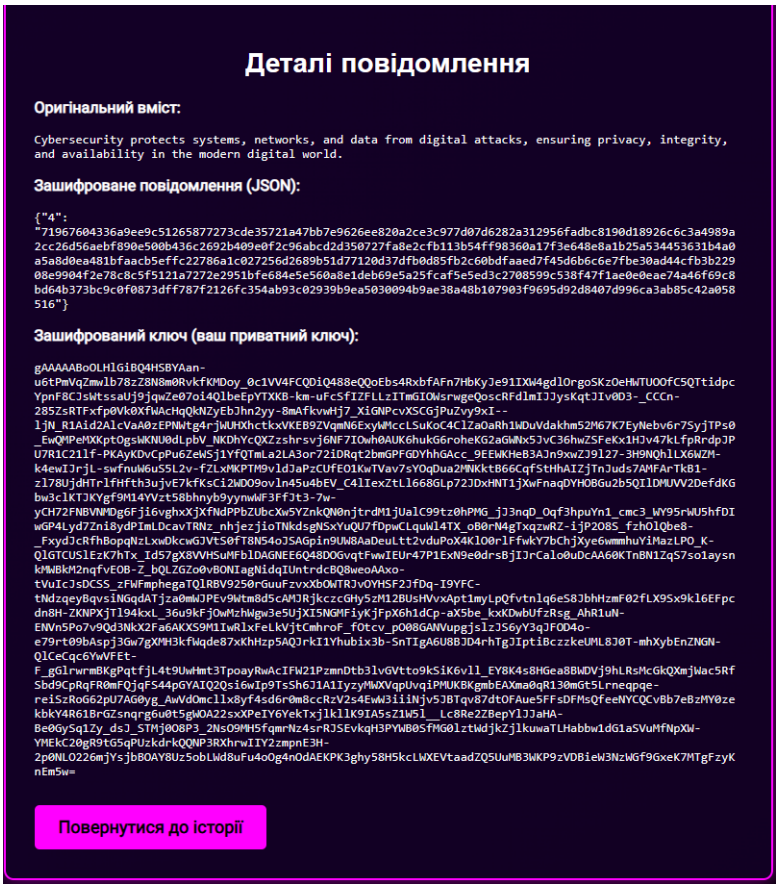

Рисунок 1. Деталі повідомлення

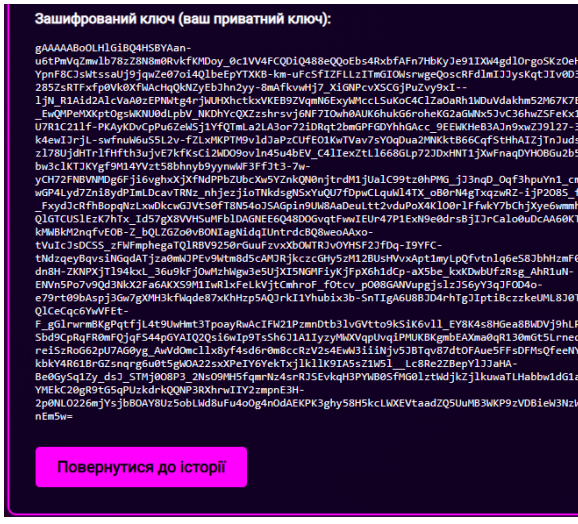

Рисунок 2. Процес відправки зображення

Продовження додатку Б

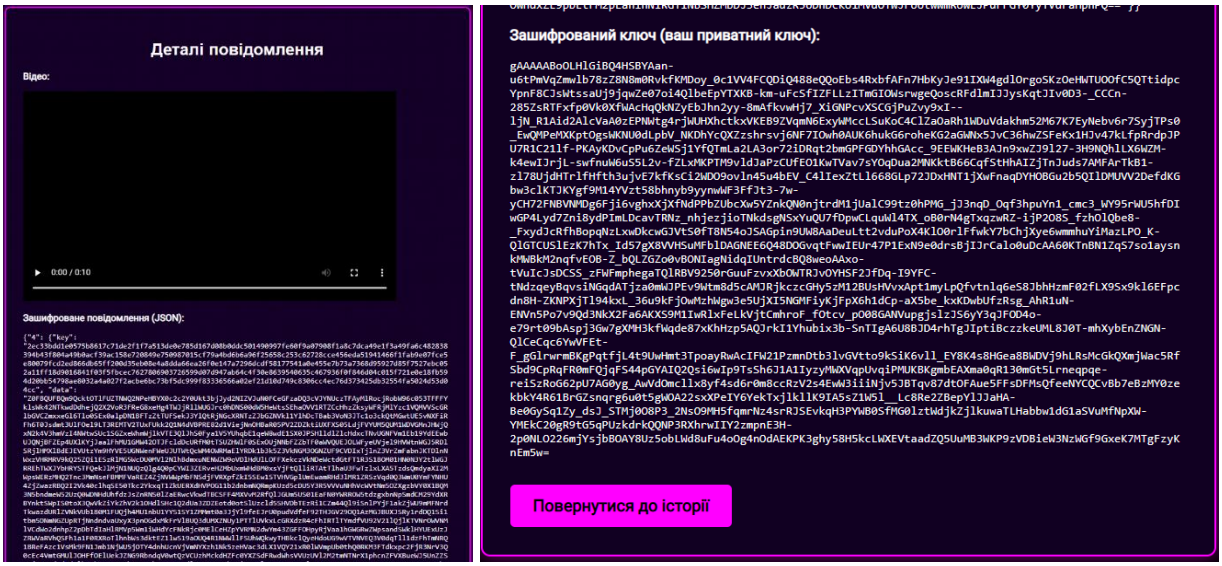

Рисунок 3. Процес відправки відео