

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ ЗАКЛАД
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Навчально-науковий інститут фізики, математики та інформаційних
технологій

Кафедра інформаційних технологій та систем

Хомич Дмитро Вікторович

**СИСТЕМА ЗАХИЩЕНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ З
ВИКОРИСТАННЯМ БЛОКЧЕЙНУ**

кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня
освітньої програми «Комп’ютерні науки»
за спеціальністю F3 Комп’ютерні науки

Особистий підпис _____ Дмитро ХОМИЧ

Науковий керівник _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”

Факультет (інститут)

Навчально-науковий інститут фізики,
математики та інформаційних технологій

Кафедра

Інформаційних технологій та систем

Рівень освіти

перший (бакалаврський)

Спеціальність

Ф3 Комп'ютерні науки

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТС

Микола СЕМЕНОВ

(підпис)

(ім'я, прізвище)

“ ”

2024 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Хомича Дмитра Вікторовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Система захищеного обміну повідомленнями з
використанням блокчейну

Керівник кваліфікаційної роботи

Семенов М.А.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету

Від“ ”

2024 року №

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи (проекту) у результаті виконання роботи

повинно бути розроблена система захищеного обміну повідомленнями з
використанням блокчейну

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

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

ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА МЕТОДОЛОГІЙ ОБМІНУ

РЕКОМЕНДАЦІЇ ЩОДО ОПТИМІЗАЦІЇ НАЯВНИХ ТЕХНОЛОГІЙ

РОЗРОБКА СЕРВІСУ ОБМІНУ ПОВІДОМЛЕННЯМИ.

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

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

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання „_____” _____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з / п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 15 жовтня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	Другий тиждень листопада (10 листопада)	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 15 грудня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 28 січня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	Перший тиждень березня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 31 березня	
	Попередній захист роботи на кафедрі	квітень	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент

підпис

Дмитро ХОМИЧ

Керівник проекту (роботи)

підпис

Микола СЕМЕНОВ

АНОТАЦІЯ

Хомич Д.В.

Тема: Система захищеного обміну повідомленнями з використанням блокчейну.

Спеціальність: F3 «Комп'ютерні науки»

Установа: ЛНУ імені Тараса Шевченка, 2025 р.

Бакалаврська робота містить: 68 с., 15 рис., 3 табл., 1 додат., 17 джерел.

Об'єкт дослідження: процеси організації та реалізації системи захищеного обміну повідомленнями з використанням блокчейну.

Предмет дослідження: система захищеного обміну повідомленнями з використанням блокчейну.

Мета роботи – розробка системи захищеного обміну повідомленнями на основі блокчейну, яка забезпечує високий рівень конфіденційності та цілісності даних..

Результати роботи. У роботі проведено порівняльний аналіз наявних методів та алгоритмів для реалізації технології блокчейну та розроблено застосунок та програмне середовище для захищеного обміну повідомлень за допомогою блокчейну.

Ключові слова. БЛОКЧЕЙН, ОБМІН ПОВІДОМЛЕННЯМИ, СМАРТ-КОТРАКТ, SOLID.

ABSTRACT

Khomych D.V.

Title: Secure Messaging System Using Blockchain

Specialization: F3 "Computer Science"

Institution: LTSNU, 2025

Bachelor's thesis includes: 68 pages, 15 figures, 3 tables, 1 appendix, 17 references.

Research Object: Processes of organizing and implementing a secure messaging system using blockchain.

Research Subject: Secure messaging system using blockchain.

Objective: Development of a secure messaging system based on blockchain technology that ensures a high level of data confidentiality and integrity.

Results: The study includes a comparative analysis of existing methods and algorithms for blockchain implementation. A software application and environment for secure messaging using blockchain have been developed.

Keywords: BLOCKCHAIN, MESSAGING, SMART CONTRACT, SOLID.

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ БЛОКЧЕЙНУ, ПРОБЛЕМИ ОРГАНІЗАЦІЇ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ	10
1.2 Огляд основних характеристик технології блокчейну	10
1.3 Аналіз платформ блокчейну для реалізації системи обміну повідомленнями та обґрунтування вибору інструментів та мови Python	21
РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ СИСТЕМИ ДЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ	26
2.1 Функціональні вимоги до системи обміну повідомленнями	26
2.2 Архітектура системи	30
2.3 Методологія розробки системи обміну повідомленнями на основі технологій блокчейну	33
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ	35
3.1 Створення та налаштування локального середовища для функціонування системи блокчейну	35
3.2 Реалізація системи обміну повідомленнями на основі Python	48
3.3 Тестування функціональності системи обміну повідомленнями	57
ВИСНОВКИ	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТКИ	64

ВСТУП

Актуальність дослідження полягає в тому, що в умовах сучасного цифрового світу зростає потреба у захищених методах обміну повідомленнями, які забезпечують конфіденційність, цілісність та автентичність даних. Традиційні системи обміну повідомленнями часто стикаються з проблемами безпеки, такими як можливість злому, втручання або втрати даних. Використання блокчейну як технології, що забезпечує децентралізоване та захищене зберігання інформації, є перспективним рішенням для подолання цих проблем.

Аналіз проблеми показує, що існуючі системи обміну повідомленнями часто не забезпечують достатнього рівня захисту, особливо в умовах зростання кількості кібератак.

Дослідженням подібних питань займалися такі вчені та фахівці, як Накамото Сатоші (вигадане ім'я особи чи групи осіб, які розробили протокол криптовалюти Біткойн та розробили першу версію програмного забезпечення для його реалізації), а також численні дослідники, які розглядали застосування блокчейну в різних сферах, включаючи фінанси, логістику та комунікації.

До теми дослідження ми прийшли через аналіз сучасних тенденцій у сфері кібербезпеки та децентралізованих технологій. Блокчейн, як інноваційна технологія, пропонує нові підходи до організації захищеного обміну даними, що стало основою для вибору теми кваліфікаційної роботи.

Предмет дослідження – система захищеного обміну повідомленнями з використанням блокчейну.

Об'єкт дослідження – процеси організації та реалізації системи захищеного обміну повідомленнями з використанням блокчейну.

Метою роботи є розробка системи захищеного обміну повідомленнями на основі блокчейну, яка забезпечує високий рівень конфіденційності та цілісності даних.

Для досягнення цієї мети було поставлено такі завдання:

1. Провести аналіз технологій блокчейну та визначити їх придатність для організації захищеного обміну повідомленнями.
2. Розробити модель системи, яка включає функціональні вимоги, архітектуру та механізми забезпечення безпеки.
3. Реалізувати систему з використанням мови Python та бібліотеки Web3.py, розробити смарт-контракти для організації обміну повідомленнями.
4. Провести тестування системи для оцінки її функціональності, безпеки та продуктивності.

Розділ 1 присвячений аналізу технологій блокчейну, проблем організації обміну повідомленнями та обґрунтуванню вибору інструментів для реалізації системи.

Розділ 2 описує розробку моделі системи, включаючи функціональні вимоги, архітектуру.

Розділ 3 містить опис реалізації системи, розробку смарт-контрактів, тестування функціональності, безпеки та продуктивності системи.

Таким чином, у роботі представлено комплексний підхід до створення системи захищеного обміну повідомленнями з використанням блокчейну, що може стати основою для подальших досліджень у цій галузі.

РОЗДІЛ 1. АНАЛІЗ ТЕХНОЛОГІЙ БЛОКЧЕЙНУ, ПРОБЛЕМИ ОРГАНІЗАЦІЇ ОБМІНУ ПОВІДОМЛЕННЯМИ ТА ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ

1.2 Огляд основних характеристик технології блокчейну

Технологія блокчейну вперше була представлена у 2008 році в роботі під псевдонімом Сатоші Накамото, яка описувала концепцію децентралізованої цифрової валюти — Біткойн. Блокчейн став основою для реалізації цієї валюти, забезпечуючи прозорість, безпеку та децентралізацію. Історія створення блокчейну тісно пов'язана з пошуком рішень для забезпечення довіри між учасниками мережі без необхідності використання централізованих інститутів. Основна ідея блокчейну полягає в тому, що дані зберігаються у вигляді ланцюжка блоків, кожен з яких містить інформацію про попередній блок, що забезпечує їхню незмінність та захищеність.

Математичні основи блокчейну ґрунтуються на криптографічних алгоритмах, таких як хешування (наприклад, SHA-256) та цифрові підписи. Хешування використовується для створення унікальних ідентифікаторів блоків, що робить їх неможливими для підробки. Цифрові підписи забезпечують автентифікацію учасників мережі та цілісність транзакцій. Крім того, блокчейн використовує механізми консенсусу (наприклад, Proof of Work або Proof of Stake), які дозволяють учасникам мережі домовлятися про стан системи без централізованого контролю.

Розглянемо основні властивості хеш-функцій.

Хеш-функція $H(m)$:

- приймає вхідні дані m (рядок або файл будь-якої довжини);
- повертає хеш-значення h , яке має фіксовану довжину (256 біт для SHA-256).

Властивості хеш-функцій:

1. **Детермінованість:** Для одного і того ж вхідного m завжди повертається однакове h .
2. **Односторонність:** Обчислити m за h неможливо за прийнятний час.
3. **Строга чутливість до змін:** Навіть незначна зміна у m призводить до повністю іншого h (ефект лавини).
4. **Колізійна стійкість:** Неможливо знайти два різних m_1 і m_2 , для яких $H(m_1) = H(m_2)$.
5. **Стійкість до другого прообразу:** Неможливо знайти m_2 для заданого m_1 , такого що $H(m_1) = H(m_2)$.

Хеш-функції базуються на наступних концепціях:

1. **Розбиття на блоки:** вхідний текст m ділиться на блоки фіксованої довжини B . Для SHA-256 $B = 512$ біт. Якщо довжина m не кратна B , використовується доповнення (padding).
2. **Компресійна функція:** хеш-функція складається з багаторазового застосування компресійної функції f яка зменшує блок до меншої фіксованої довжини.

$$h_i = f(h_{i-1}, m_i) \quad (1.1)$$

де:

- h_0 — початкове значення (ініціалізаційний вектор, IV).
- m_i — блок даних.
- h_i — проміжний хеш.

3. **Перестановки та нелінійні операції:** алгоритми хешування використовують комбінацію перестановок (переміщення бітів) і нелінійних операцій (наприклад, XOR, AND, OR), щоб забезпечити односторонність і колізійну стійкість.

SHA-256 (Secure Hash Algorithm 256-bit) — це криптографічний хеш-функція, яка генерує унікальний 256-бітний (32-байтовий) хеш для будь-якого вхідного повідомлення. Цей алгоритм широко використовується для

забезпечення цілісності даних і безпеки в різних додатках, таких як цифрові підписи та блокчейн.

Розглянемо основні кроки алгоритму SHA-256:

1. **Додавання доповнення:** Вхідне повідомлення доповнюється так, щоб його довжина стала кратною 512 бітам. Спочатку додається біт "1", а потім стільки нулів, щоб довжина повідомлення стала рівною 448 бітам (мод 512). Потім додається 64-бітове представлення довжини початкового повідомлення.

2. **Ініціалізація хеш-значень:** Встановлюються початкові значення восьми 32-бітних слів ($H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$).

*$H_0 = 0x6a09e667, H_1 = 0xbb67ae85, H_2 = 0x3c6ef372, H_3 = 0xa54ff53a,$
 $H_4 = 0x510e527f, H_5 = 0x9b05688c, H_6 = 0x1f83d9ab, H_7 = 0x5be0cd19$*

3. **Обробка повідомлення блоками:** Повідомлення розбивається на блоки **M** по 512 бітів. Кожен блок обробляється окремо.

4. **Основний цикл:** Виконується 64 раунди обчислень, які включають логічні операції, додавання та побітові зсуви.

Кожен 512-бітний блок розширюється до 64 32-бітних слів.

$$(W_0, W_1, \dots, W_{63})$$

Для кожного 512-бітового блоку **M**:

$$W_t = \begin{cases} M_t & \text{для } t \in [0, 15], \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16} & \text{для } t \in [16, 63]. \end{cases}$$

де:

$$\sigma_0(x) = (x \gg 7) \oplus (x \gg 18) \oplus (x \gg 3),$$

$$\sigma_1(x) = (x \gg 17) \oplus (x \gg 19) \oplus (x \gg 10).$$

5. Оновлення хеш-значень: Після обробки кожного блоку, хеш-значення оновлюються.

Малі сигма-функції:

$$\Sigma_0(x) = (x \gg 2) \oplus (x \gg 13) \oplus (x \gg 22),$$

$$\Sigma_1(x) = (x \gg 6) \oplus (x \gg 11) \oplus (x \gg 25).$$

Вибір (Choose):

$$C_h(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$$

Мажоритарність (Majority):

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z).$$

Основний цикл:

$$T_1 = h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t,$$

$$h=g, g=f, f=e, e=d+T_1, d=c, c=b, b=a, a=T_1+T_2.$$

Де K_t — фіксовані 64 константи.

Додавання до попереднього хешу:

$$H_i = H_i + [a, b, c, d, e, f, g, h].$$

6. Формування кінцевого хешу: Після обробки всіх блоків, кінцевий хеш формується шляхом конкатенації восьми 32-бітних слів.

$$H = H_0 \parallel H_1 \parallel H_2 \parallel \dots \parallel H_7.$$

Де \parallel означає конкатенацію.

Для формування цифрових підписів в криптографії, та як наслідок в технологіях блокчейн (підтвердження транзакцій), використовуються математичні методи побудовані на використанні еліптичних кривих.

Розглянемо цей підхід, щоб продемонструвати, що існують різні методи отримання ланцюжка хешів.

Еліптична крива над полем F_p (де p — просте число) задається рівнянням:

$$y^2 = x^3 + ax + b(\text{mod } p)$$

Де:

- a, b — коефіцієнти, які визначають форму кривої;
- $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ — умова відсутності особливих точок (ломаних, точок перетину).

Точки на еліптичній кривій мають властивість:

- можна визначити операцію додавання двох точок P і Q на кривій.
- визначається "складання" точок за геометричним правилом: проведення прямої через P і Q . Третя точка перетину відображається симетрично відносно осі X (дивись рис. 1.1).

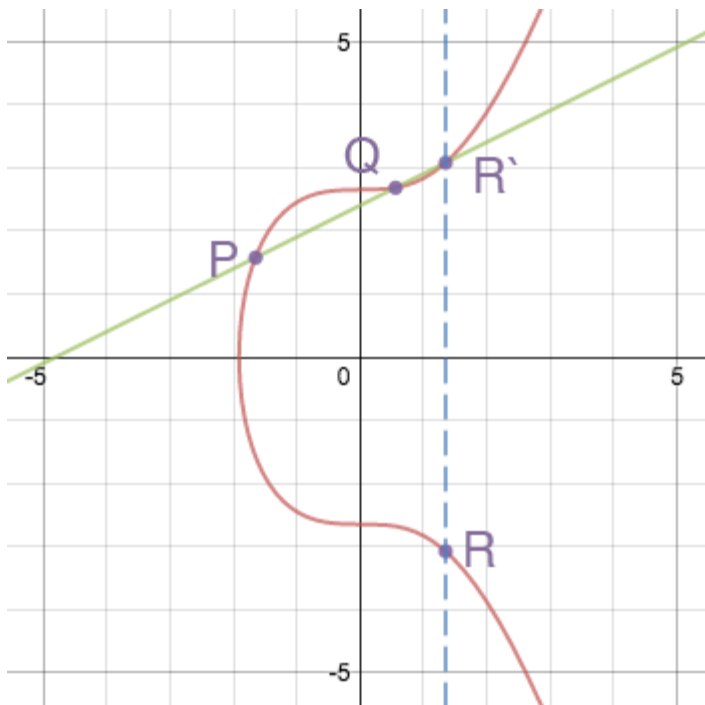


Рис. 1.1 Складання P і Q [].

На рис. 1.1 точки R та R' отримані за операцією додавання P і Q .

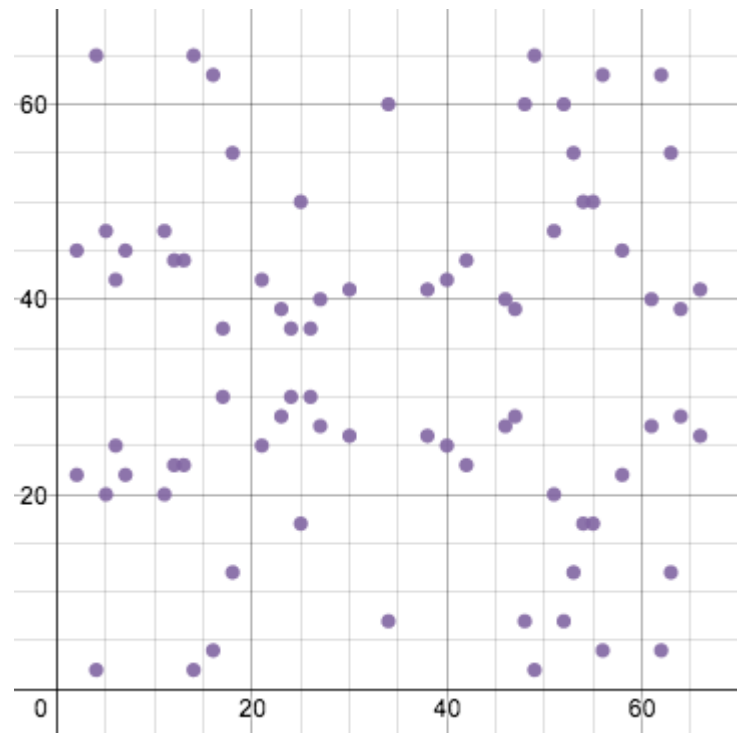


Рис. 1.2 F_{67} еліптична крива за модулем 67 [].

На рис. 1.2 представлена еліптична крива за модулем 67 - F_{67} , яка є множиною чисел діапазону від 0 до 66. Досягає значення 67 результатом будуть знову числа починаючи з 0.

На основі цього використовують ECC та ECDSA. ECC (Elliptic Curve Cryptography) — це криптографічна схема, яка базується на складності обчислення дискретного логарифма на еліптичних кривих. ECDSA (Elliptic Curve Digital Signature Algorithm) використовується для підпису транзакцій у блокчейні.

Термінологія, пов'язана з блокчейном, включає такі поняття, як "блок" (структура даних, що містить транзакції), "ланцюг блоків" (послідовність блоків, зв'язаних через хеш-значення), "децентралізація" (відсутність центрального управління), "нода" (вузол мережі, який підтримує роботу блокчейну), "смарт-контракт" (автоматичний договір, виконаний у вигляді програмного коду) та "майнінг" (процес створення нових блоків шляхом

вирішення складних математичних задач). Ці терміни є ключовими для розуміння принципів роботи блокчейну та його застосування в різних сферах.

Таким чином, блокчейн як технологія поєднує в собі криптографічні методи, децентралізовану архітектуру та механізми консенсусу, що робить його потужним інструментом для забезпечення безпеки, прозорості та довіри в цифрових системах.

2.2. Обґрунтування використання блокчейну для вирішення проблем безпечного обміну повідомленнями

Сучасні месенджери, такі як Telegram, WhatsApp, Facebook Messenger і WeChat, стали невід'ємною частиною повсякденного життя мільйонів користувачів. Вони пропонують зручні засоби комунікації, передавання мультимедійного контенту та інтеграцію з іншими сервісами. Основною технологією для забезпечення конфіденційності та безпеки обміну повідомленнями в цих платформах є шифрування. Наприклад, Telegram використовує власний протокол MTProto, WhatsApp і Facebook Messenger (в секретних чатах) базуються на протоколі Signal з підтримкою наскрізного шифрування (End-to-End Encryption, E2EE), а WeChat — здебільшого спирається на менш захищені протоколи для зберігання повідомлень на серверах.

Розглянемо протокол MTProto (месенджер Telegram). MTProto (Mobile Telegram Protocol) — це власний протокол шифрування, розроблений компанією Telegram для забезпечення безпечного, швидкого та стабільного обміну повідомленнями. Основною метою його створення було поєднання високого рівня захисту даних із мінімальними затримками під час передавання інформації.

MTProto складається з трьох рівнів.

Транспортний рівень: забезпечує передавання даних між клієнтом і сервером за допомогою TCP, UDP або HTTPS. Вибір методу залежить від умов мережі.

Рівень авторизації: здійснює захищену аутентифікацію користувачів і встановлює шифроване з'єднання між клієнтом і сервером за допомогою асиметричної криптографії (RSA та Diffie-Hellman).

Рівень високого рівня (High-level API): відповідає за передавання структурованих даних, таких як текстові повідомлення, файли, мультимедіа тощо.

Узагальнено використання протоколу MTPROTO можна представити схемою на рис 1.3:

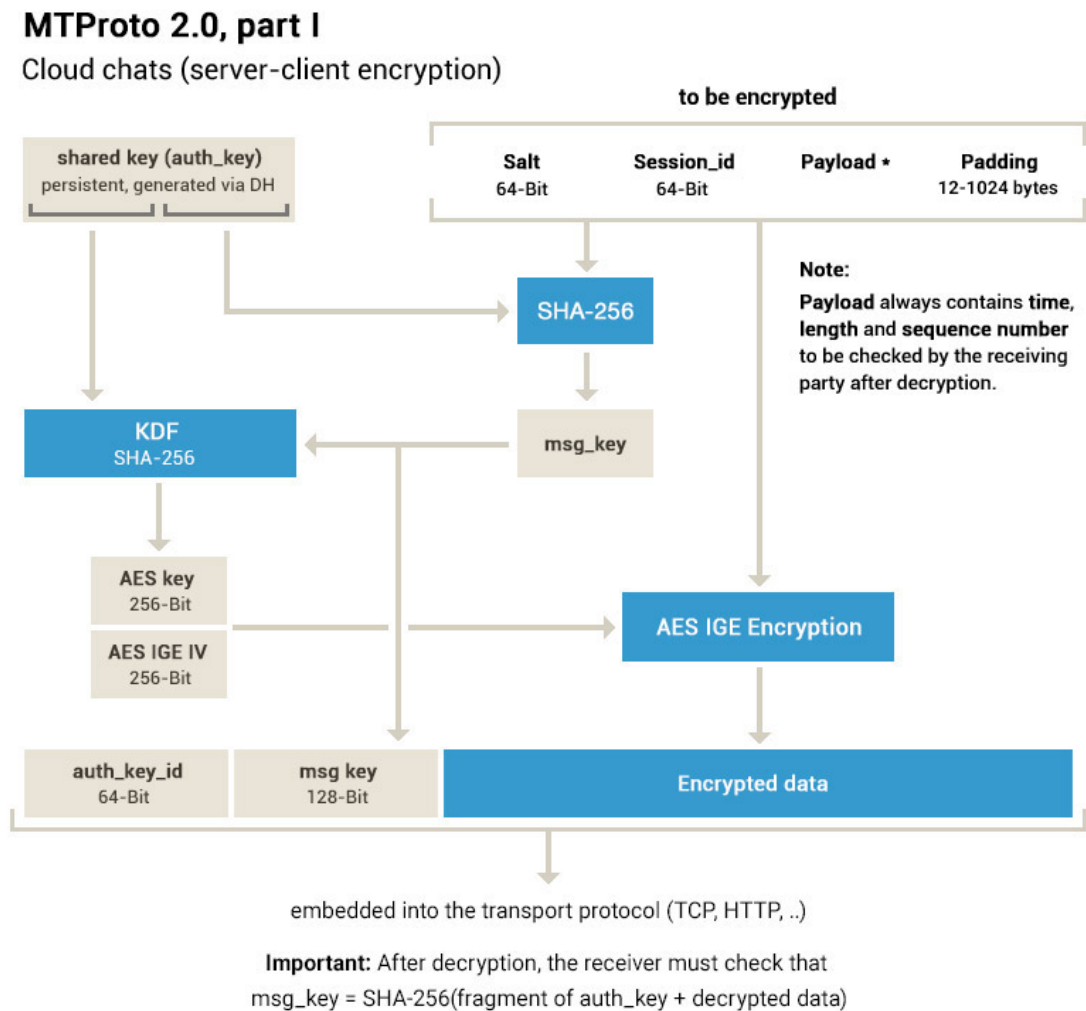


Рис. 1.3 Схема протоколу MTPROTO

Як бачимо з рис. 1.3 MTPROTO поєднує кілька криптографічних методів:

- асиметричне шифрування: використовується на етапі встановлення сесії з використанням алгоритмів RSA та Diffie-Hellman.

- симетричне шифрування: для безпечного передавання даних застосовується алгоритм AES-256 у режимі IGE (Infinite Garble Extension).
- хеш-функції: SHA-256 використовується для забезпечення цілісності даних та генерування секретних кодів (*KDF - Key derivative function*).

Signal Protocol — це сучасний криптографічний протокол, розроблений компанією Signal Foundation для забезпечення безпечного обміну повідомленнями та дзвінками. Його використовують багато популярних месенджерів, зокрема Signal, WhatsApp, Facebook Messenger (секретні чати) та Google Messages. Головною особливістю протоколу є наскрізне шифрування (E2EE), яке гарантує, що повідомлення доступні лише відправнику та отримувачу.

Протокол Signal використовує наскрізне шифрування (E2EE): повідомлення шифруються на пристрої відправника та розшифровуються лише на пристрої отримувача. Навіть сервери оператора не можуть прочитати повідомлення. Для протоколу Signal притомана ефектна пряма секретність (Perfect Forward Secrecy, PFS): кожне нове повідомлення використовує унікальний ключ шифрування, що зводить до мінімуму ризик компрометації всього листування, навіть якщо один із ключів буде скомпрометовано. Протокол автоматично змінює ключі шифрування після кожної сесії, зберігаючи високий рівень безпеки.

Signal застосовує наступні криптографічні механізми:

Диффі-Геллман (X3DH) для аутентифікації: використовується для створення початкового шифрованого каналу зв'язку.

Двоетапний ратифікаційний алгоритм Double Ratchet: після встановлення з'єднання ключі шифрування динамічно змінюються для кожного нового повідомлення.

Криві Едвардса (Curve25519): застосовуються для ефективної асиметричної криптографії.

Алгоритм шифрування AES-256 у режимі CBC: забезпечує симетричне шифрування повідомлень.

Хешування HMAC-SHA256: гарантує цілісність даних і захист від підробок.

Попри застосування сучасних криптографічних технологій, безпека таких платформ має певні недоліки. Насамперед, централізована архітектура багатьох месенджерів створює ризики витоку даних через компрометацію серверів або незаконний доступ до них. Крім того, алгоритми шифрування часто залежать від серверної інфраструктури компанії, яка може втручатися у процес передавання інформації. Це ставить під сумнів повну конфіденційність повідомлень.

Використання блокчейн-технологій може стати перспективним вирішенням цих проблем. Завдяки децентралізованій архітектурі блокчейн дозволяє зберігати та передавати інформацію без необхідності довіри до централізованого сервера. Протоколи консенсусу та криптографічний захист забезпечують цілісність і автентичність повідомлень. Використання смарт-контрактів може автоматизувати процес керування доступом до повідомлень та перевірки правомірності дій користувачів. Таким чином, блокчейн може не лише усунути проблему центральних точок уразливості, але й забезпечити прозорість і підвищену конфіденційність у системах обміну повідомленнями.

1.3 Аналіз платформ блокчейну для реалізації системи обміну повідомленнями та обґрунтування вибору інструментів та мови Python

Для реалізації системи захищеного обміну повідомленнями необхідно вибрати відповідну блокчейн-платформу, яка забезпечує необхідний рівень безпеки, масштабованості та підтримки смарт-контрактів. Серед найпопулярніших платформ, які можна розглянути, є Ethereum, Hyperledger Fabric, Binance Smart Chain та Polkadot. Кожна з них має свої переваги та недоліки, які варто враховувати під час вибору.

Ethereum — одна з найпоширеніших платформ для розробки децентралізованих додатків (dApps). Вона підтримує смарт-контракти, що дозволяє реалізувати логіку обміну повідомленнями на рівні блокчейну. Ethereum має велику спільноту розробників і багато готових інструментів для інтеграції. Однак, мережа Ethereum може страждати від високих комісій за транзакції (gas fees) та обмеженої пропускної здатності, що може вплинути на продуктивність системи.

Hyperledger Fabric — це корпоративна блокчейн-платформа, розроблена під егідою Linux Foundation. Вона відрізняється високою продуктивністю та підтримкою приватних транзакцій, що може бути важливим для системи обміну повідомленнями. Однак, Hyperledger Fabric потребує більш складного налаштування та менш підходить для публічних децентралізованих систем.

Binance Smart Chain (BSC) — це блокчейн, який сумісний з Ethereum, але пропонує нижчі комісії за транзакції та вищу швидкість обробки. BSC підтримує смарт-контракти та має активну екосистему, що робить його привабливим вибором для розробки. Однак, він менш децентралізований порівняно з Ethereum, що може вплинути на рівень довіри до системи.

Polkadot — це платформа, яка забезпечує взаємодію між різними блокчейнами (interoperability). Вона підтримує смарт-контракти та пропонує

високу масштабованість. Однак, Polkadot є відносно новою платформою, і її екосистема ще розвивається.

Для реалізації системи обміну повідомленнями була обрана мова програмування Python через її простоту, велику кількість бібліотек та активну спільноту розробників. Python є ідеальним вибором для швидкої розробки прототипів та інтеграції з блокчейн-платформами. Серед ключових інструментів, які будуть використовуватися, можна виділити:

Web3.py — бібліотека для взаємодії з блокчейном Ethereum (або сумісними платформами, як Binance Smart Chain). Вона дозволяє відправляти транзакції, взаємодіяти з смарт-контрактами та отримувати дані з блокчейну.

Solidity — мова програмування для написання смарт-контрактів на платформах, як Ethereum та BSC. Вона дозволяє реалізувати логіку обміну повідомленнями на рівні блокчейну.

Truffle або Hardhat — фреймворки для розробки та тестування смарт-контрактів. Вони забезпечують зручне середовище для розробки та автоматизації процесів.

IPFS (InterPlanetary File System) — децентралізована система зберігання файлів, яка може бути використана для зберігання великих повідомлень або вкладень поза блокчейном, зберігаючи в блокчейні лише посилання на дані.

Вибір Python та супутніх інструментів обумовлений їхньою зручністю, широкими можливостями інтеграції та підтримкою блокчейн-технологій. Це дозволяє створити ефективну, масштабовану та безпечну систему обміну повідомленнями з використанням блокчейну.

Rinkeby — це публічна тестова мережа Ethereum, яка імітує основну мережу, але використовує тестові ЕТН для транзакцій. Ethereum — це децентралізована блокчейн-платформа, яка була запущена в 2015 році Віталіком Бутеріном та командою розробників. Вона є другою за популярністю криптовалютною платформою після Біткойна, але, на відміну від останнього, Ethereum не обмежується лише функцією цифрової валюти.

Основна мета Ethereum — надати інфраструктуру для створення децентралізованих додатків (dApps) та смарт-контрактів, що робить її універсальним інструментом для розробки різноманітних блокчейн-рішень.

Обираючи систему організації блокчейн, враховано те, що Ethereum має такі інструменти:

Смарт-контракти. Ethereum став першою платформою, яка реалізувала концепцію смарт-контрактів — програмного коду, що автоматично виконує умови угоди між сторонами без необхідності посередників. Смарт-контракти дозволяють створювати складні логічні операції, такі як фінансові транзакції, управління активами, голосування тощо.

Мова програмування Solidity. Для написання смарт-контрактів на Ethereum використовується мова програмування Solidity, яка спеціально розроблена для створення децентралізованих додатків. Solidity є мовою високого рівня, що дозволяє розробникам легко створювати складну логіку для своїх додатків.

Ефір (Ether, ETH). Ether — це власна криптовалюта платформи Ethereum, яка використовується для оплати транзакцій та виконання смарт-контрактів. Кожна операція в мережі Ethereum вимагає витрати "газу" (gas), який оплачується в Ether. Це забезпечує стимул для валідаторів (раніше майнерів) підтримувати роботу мережі.

Децентралізована віртуальна машина (EVM). Ethereum Virtual Machine (EVM) — це середовище виконання для смарт-контрактів. EVM забезпечує ізоляцію коду смарт-контрактів від основної мережі, що робить їх безпечними та незалежними. Кожен вузол мережі Ethereum запускає EVM для виконання смарт-контрактів.

Децентралізація та безпека. Ethereum, як і інші блокчейни, є децентралізованою мережею, що складається з тисяч вузлів по всьому світу. Це робить її стійкою до цензури та атак. Крім того, Ethereum використовує криптографічні алгоритми для забезпечення безпеки транзакцій та даних.

Екосистема dApps. Ethereum є основою для величезної кількості децентралізованих додатків, які охоплюють різні сфери: фінанси (DeFi), мистецтво (NFT), ігри, управління даними тощо. Популярність Ethereum як платформи для dApps обумовлена її гнучкістю та широкими можливостями.

Перехід на Ethereum 2.0. Ethereum зазнає значних змін у рамках оновлення до Ethereum 2.0, яке включає перехід з консенсусу Proof of Work (PoW) на Proof of Stake (PoS). Це оновлення покликане збільшити масштабованість, енергоефективність та швидкість мережі.

Вибір Ethereum обумовлено також можливістю використовувати технології блокчейн в індивідуальному бакалаврському дослідженні. Для розробки та тестування блокчейн-додатків на платформі Ethereum можна використовувати два підходи: локальну мережу через Ganache і публічні тестові мережі, такі як Rinkeby. Кожен із цих способів має свої переваги та особливості.

Ganache — це інструмент для локального тестування та розробки блокчейн-додатків, який надається компанією Truffle Suite. Він дозволяє створювати локальну блокчейн-мережу для тестування смарт-контрактів та децентралізованих додатків (dApps) без необхідності підключатися до основної мережі Ethereum чи інших блокчейнів. Ganache є незамінним інструментом для швидкої ітерації та налагодження проектів.

До переваг Ganache в рамках цього дослідження відмічаємо:

Ganache дозволяє запустити повноцінну блокчейн-мережу на локальному комп'ютері. Це дозволяє тестувати свої смарт-контракти та додатки в ізольованому середовищі без ризику втрати коштів або інших наслідків, пов'язаних з роботою в основній мережі.

Ganache працює значно швидше, ніж публічні блокчейни, оскільки не потребує консенсусу між вузлами. Це дозволяє швидко розгортати смарт-контракти та виконувати транзакції.

Ganache автоматично створює кілька тестових акаунтів (зазвичай 10), кожен з яких має велику кількість тестових коштів (ETH). Це дозволяє розробникам відразу почати тестування без необхідності створення акаунтів або отримання тестових коштів.

Ganache доступний у двох версіях: Ganache CLI — версія для командного рядка, яка підходить для автоматизації та інтеграції з іншими інструментами. Ganache GUI — графічна версія, яка надає зручний інтерфейс для моніторингу транзакцій, блоків та стану смарт-контрактів.

Ganache легко інтегрується з іншими інструментами для розробки блокчейн-додатків, такими як Truffle, Hardhat та Remix. Це дозволяє створювати комплексні тестові середовища для розробки та налагодження.

Ganache підтримує розгортання та виконання смарт-контрактів, написаних на Solidity або інших сумісних мовах. Він також надає детальну інформацію про виконання транзакцій, що допомагає розробникам швидко знаходити та виправляти помилки.

Ganache надає детальний огляд усіх транзакцій, блоків та стану смарт-контрактів. Це дозволяє розробникам легко відстежувати та аналізувати роботу своїх додатків.

Rinkeby — це публічна тестова мережа Ethereum, яка імітує основну мережу, але використовує тестові ETH для транзакцій (ETH (Ether) — це криптовалюта, яка використовується в блокчейні Ethereum).

Наш вибір Ganache обумовлений тим, що він є більш доступним і простим, особливо внаслідок того, що проект зосереджується на розробці та тестуванні смарт-контрактів локально.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ СИСТЕМИ ДЛЯ ОБМІНУ ПОВІДОМЛЕННЯМИ

2.1 Функціональні вимоги до системи обміну повідомленнями

На основі отриманих в розділі 1 результатів, аналізу та додаткових міркувань сформулюємо функціональні вимоги до системи захищеного обміну повідомленнями з використанням блокчейну

Система захищеного обміну повідомленнями з використанням блокчейну повинна забезпечувати високий рівень конфіденційності, цілісності та автентичності даних, а також надавати зручний інтерфейс для користувачів. Основні функціональні вимоги до системи включають:

1. Реєстрація та автентифікація користувачів

Система повинна надавати можливість реєстрації нових користувачів та автентифікації через механізми цифрових підписів або інші криптографічні методи. Кожен користувач повинен мати унікальний ідентифікатор, який використовується для взаємодії з системою.

2. Шифрування повідомлень

Усі повідомлення повинні шифруватися перед відправкою, щоб забезпечити конфіденційність даних. Для шифрування можуть використовуватися сучасні криптографічні алгоритми, такі як AES або RSA.

3. Збереження повідомлень у блокчейні

Повідомлення або їх хеш-значення повинні зберігатися в блокчейні для забезпечення їхньої незмінності та довгострокової цілісності. Це дозволяє гарантувати, що повідомлення не можуть бути змінені або вилучені після відправки.

4. Використання смарт-контрактів

Система повинна використовувати смарт-контракти для автоматизації процесів обміну повідомленнями, таких як перевірка автентичності користувачів, шифрування та дешифрування даних, а також логування подій.

5. Децентралізоване зберігання даних

Для зберігання великих обсягів даних (наприклад, файлів або мультимедіа) може використовуватися децентралізована система зберігання, така як IPFS (InterPlanetary File System). У блокчейні зберігатимуться лише посилання на ці дані.

6. Можливість перевірки цілісності повідомлень

Користувачі повинні мати можливість перевіряти, чи не були повідомлення змінені після відправки. Це може бути реалізовано через збереження хеш-значень повідомлень у блокчейні.

7. Інтерфейс для користувачів

Система повинна надавати зручний інтерфейс для відправки, отримання та перегляду повідомлень. Інтерфейс може бути реалізований у вигляді веб-додатку або мобільного застосунку.

8. Масштабованість та продуктивність

Система повинна бути розроблена з урахуванням можливості масштабування для обслуговування великої кількості користувачів. Для цього можуть використовуватися рішення другого рівня (Layer 2), такі як sidechains або плазмові ланцюги.

9. Аудит та моніторинг

Система повинна надавати можливість аудиту всіх операцій, включаючи відправку та отримання повідомлень. Це дозволяє забезпечити прозорість та довіру до системи.

10. Захист від атак

Система повинна включати механізми захисту від кібератак, таких як DDoS-атаки, спроби підробки повідомлень або несанкціонованого доступу до даних.

Ці функціональні вимоги забезпечують створення надійної, безпечної та зручної системи для обміну повідомленнями, яка використовує переваги блокчейн-технологій для забезпечення конфіденційності та цілісності даних.

Графічно функціональні вимоги зображено на рис. 2.1 у вигляді діаграми UML.



Рис. 2.1 Діаграма варіантів використання для системи захищеного обміну повідомленнями.

На рис. 2.1 описані такі елементи Актори (Actors):

Користувач — особа, яка використовує систему для обміну повідомленнями.

Адміністратор — особа, яка керує системою, налаштовує параметри та моніторить її роботу.

Блокчейн-мережа — зовнішня система, яка забезпечує зберігання даних та виконання смарт-контрактів.

IPFS — децентралізована система зберігання файлів (якщо використовується для зберігання великих даних).

Реєстрація та автентифікація користувача

- Користувач реєструється в системі.
- Користувач автентифікується за допомогою цифрового підпису або іншого механізму.

Відправка повідомлення

- Користувач шифрує повідомлення.
- Користувач відправляє повідомлення через систему, яка зберігає його в блокчейні або IPFS.

Отримання повідомлення

- Користувач отримує повідомлення від іншого користувача.
- Система дешифрує повідомлення та відображає його.

Перевірка цілісності повідомлення

- Користувач перевіряє, чи не було повідомлення змінено після відправки, використовуючи хеш-значення, збережене в блокчейні.

Керування смарт-контрактами

- Адміністратор розгортає та оновлює смарт-контракти для управління логікою обміну повідомленнями.

Моніторинг системи

- Адміністратор переглядає логи транзакцій, стан мережі та продуктивність системи.

Зберігання даних у IPFS

- Система зберігає великі файли або вкладення в IPFS та зберігає посилання на них у блокчейні.

Захист від атак

- Система автоматично виявляє та блокує спроби DDoS-атак або несанкціонованого доступу.

2.2 Архітектура системи

Загальна архітектура системи має складові:

1. Клієнтська частина: Графічний інтерфейс або консольний застосунок на Python для відправки та отримання повідомлень.
2. Блокчейн: Смарт-контракт для запису та верифікації повідомлень.
3. Шифрування: Захист повідомлень за допомогою криптографії (наприклад, бібліотека cryptography).
4. API для взаємодії: Інтерфейс, що використовує бібліотеку web3.py для взаємодії з блокчейном.

Деталізуємо архітектуру, для цього сформуємо нову діаграму (рис. 2.2)

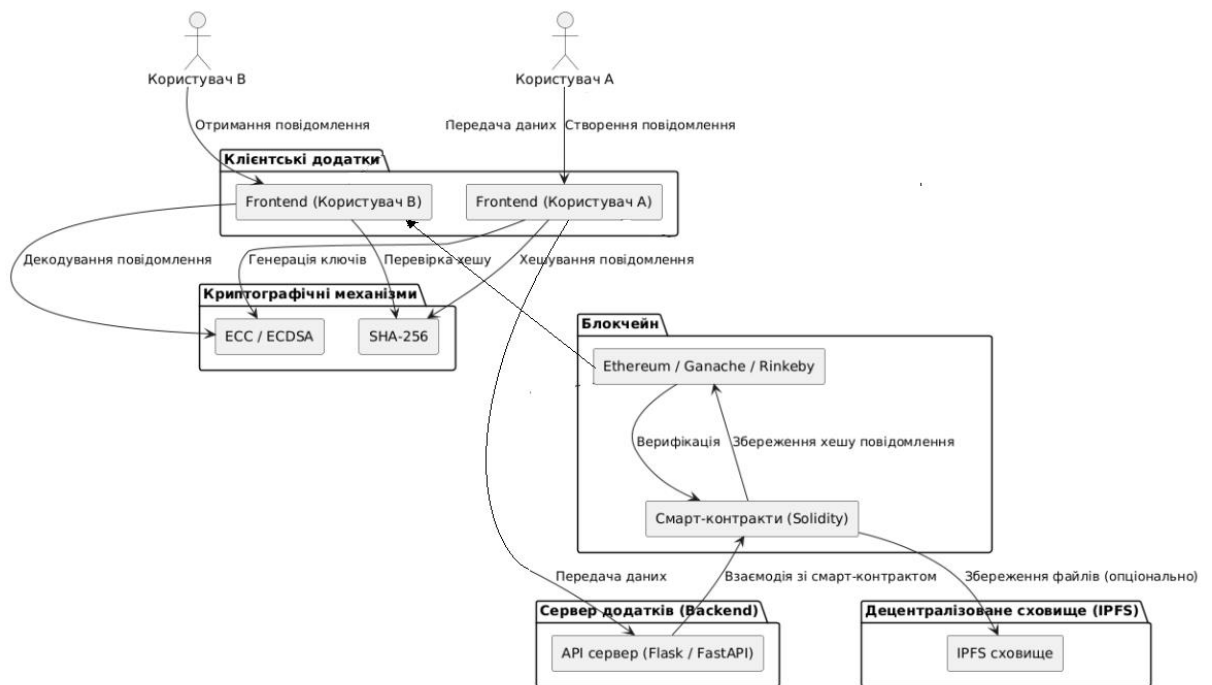


Рис. 2.2 Архітектура системи обміну повідомленнями

На рис. 2.2 двоє користувачів створюють та отримують повідомлення.

Клієнтські додатки відповідають за генерацію ключів, шифрування повідомлень і їх відправлення. Backend виконує обробку повідомлень та взаємодію з блокчейном. Блокчейн: Ethereum (Ganache) для збереження

транзакцій і смарт-контрактів. Криптографічні механізми використовуються для шифрування та забезпечення цілісності повідомлень. IPFS: додатковий компонент для зберігання файлів.

На рис. 2.3 зображено змодельовану діаграму класів для системи обміну повідомленнями.

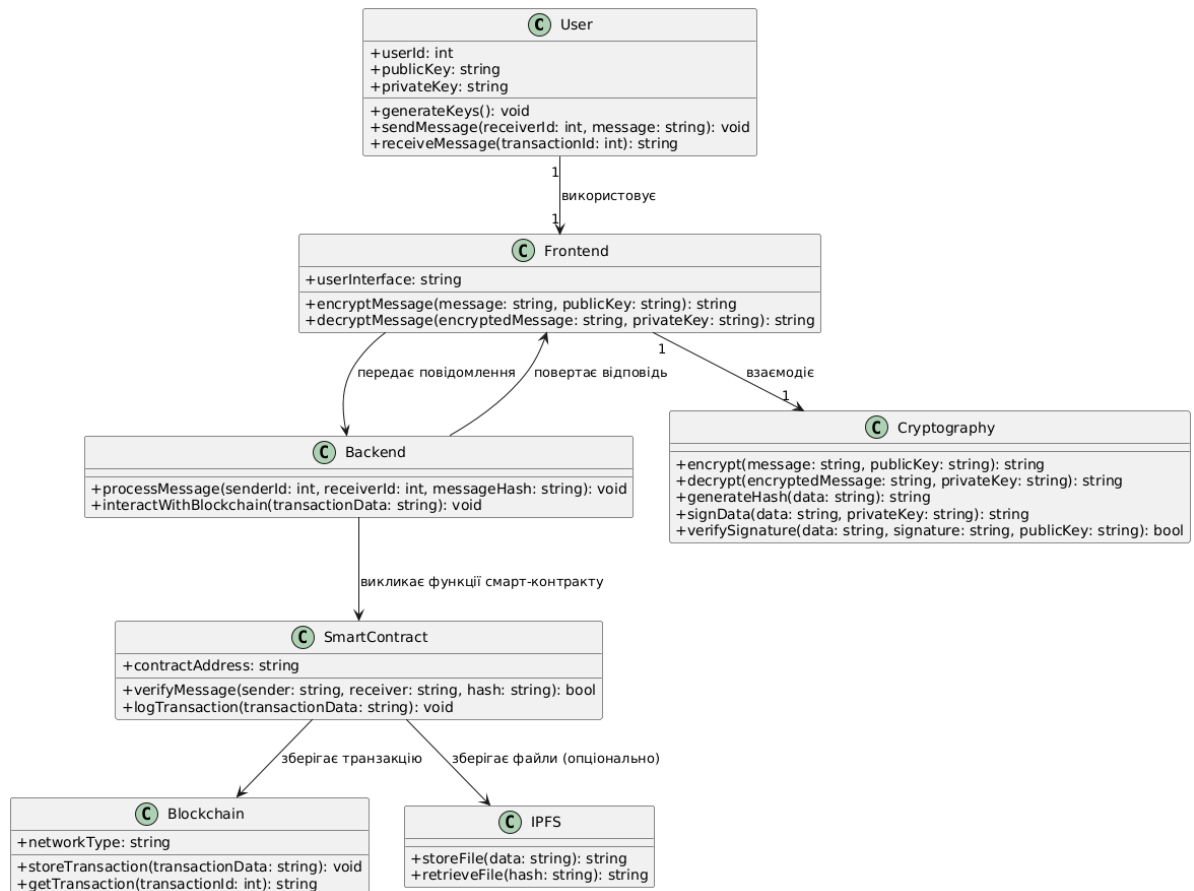


Рис. 2.3 Діаграма класів для системи обміну повідомленнями

На рис. 2.3 клас User представляє користувача системи із методами для генерації ключів та обміну повідомленнями. Клас Frontend відповідає за шифрування та розшифрування повідомлень, а також взаємодію з користувачем. Клас Backend обробляє повідомлення та взаємодіє з блокчейном. Клас Blockchain відповідає за зберігання транзакцій у блокчейні. Клас SmartContract виконує функції перевірки автентичності повідомлень та

логування транзакцій. Клас Cryptography містить методи шифрування, хешування та цифрового підпису. Клас IPFS опціональний компонент для збереження файлів у децентралізованому сховищі.

2.3 Методологія розробки системи обміну повідомленнями на основі технологій блокчейну

Методологія розробки – це сукупність принципів, методів, інструментів та практик, які використовуються для планування, управління та реалізації програмного або інформаційного проєкту. Вона визначає структуру та послідовність етапів розробки, що допомагає ефективно організувати роботу та забезпечити досягнення запланованих результатів.

Представлена нижче методологія розробки системи обміну повідомленнями на основі технологій блокчейну ґрунтувалася на поєднанні гнучких та ітеративних підходів до розробки програмного забезпечення. Основні принципи цієї методології включали забезпечення високого рівня безпеки, децентралізацію даних та ефективну інтеграцію з блокчейн-технологіями.

Результати отриманні в першому розділі та в п. 2.1 та 2.2 дозволили визначити структуру та послідовність етапів розробки:

Аналіз вимог: на початковому етапі проводився аналіз потреб користувачів та функціональних вимог системи. Особливу увагу приділяли питанням безпеки передачі повідомлень та конфіденційності даних.

Вибір технологій: обрання мови програмування Python та бібліотеки Web3.py було обґрунтовано їхньою зручністю для інтеграції з блокчейном Ethereum. Python забезпечує широкий набір інструментів для роботи з криптографією та підтримується великою спільнотою розробників.

Проектування архітектури: було розроблено модульну архітектуру, яка включала смарт-контракти, блокчейн-інтеграцію, систему зберігання даних IPFS та веб-інтерфейс для користувачів.

Розробка смарт-контрактів: створення смарт-контракту на мові Solidity, повинно забезпечувати функції збереження хеш-значень повідомлень,

автентифікацію користувачів та управління доступом до даних. Для розробки використовуються інструменти Truffle.

Інтеграція та тестування: після розгортання смарт-контракту в тестовій мережі Ethereum його буде інтегровано з Python-додатком за допомогою Web3.py.

Шифрування повідомлень реалізується за допомогою бібліотеки *cryptography*, яка містить сучасні криптографічні алгоритми на мові Python.

Оптимізація зберігання даних відбувається завдяки інтеграції з децентралізованою системою зберігання IPFS для ефективного зберігання великих обсягів даних.

Інтерфейс створюється за допомогою TKInter або веб-інтерфейс за допомогою Web3.py або Flask, що дозволяло користувачам відправляти, отримувати та переглядати повідомлення.

У підсумку, методологія розробки системи обміну повідомленнями на основі технологій блокчейну дозволяє створити безпечну та функціональну платформу, яка відповідає сучасним вимогам щодо захисту даних та децентралізації.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ОБМІНУ ПОВІДОМЛЕННЯМИ

3.1 Створення та налаштування локального середовища для функціонування системи блокчейну

Для розгортання локального середовища блокчейну встановили необхідні інструменти (програмне забезпечення):

- 1) Node.js (<https://nodejs.org/>) — для роботи з npm.
- 2) Ganache (<https://trufflesuite.com/ganache/>) — локальний блокчейн;
- 3) Truffle – для створення смарт-контрактів.

Node.js — це серверна платформа, побудована на рушії JavaScript V8 від Google. Вона дозволяє запускати JavaScript-код поза браузером, що відкриває можливість створювати бекенд-додатки, сервери та інші серверні служби. Node.js підтримує неблокуючу, подієво-орієнтовану модель вводу-виводу, що забезпечує високу продуктивність і масштабованість.

Основні можливості Node.js: обробка великої кількості одночасних запитів завдяки асинхронній архітектурі; підтримка веб-серверів, RESTful API, WebSocket-з'єднань; велика кількість бібліотек через npm; (Node Package Manager) – це менеджер пакетів для Node.js, який дозволяє встановлювати пакети: завантажувати сторонні бібліотеки та модулі з глобального репозиторію npm.

У нашому випадку ми використовуємо Node.js для того, щоб за допомогою npm встановити систему Truffle.

Для інсталяції Node.js заходимо на сайт (рис. 3.1), встановлюємо останню версію (v 22.13.1).

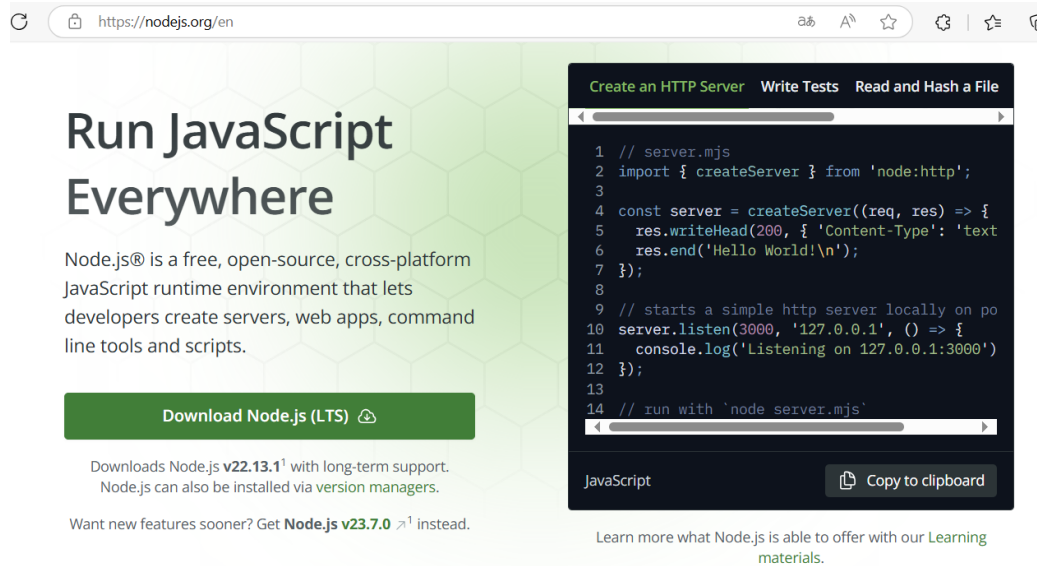


Рис. 3.1 Сайт Node.js

По завершенню інсталяції переходимо на Node.js command prompt та інсталуємо Truffle за допомогою bush команди:

```
npm install -g truffle
```

Налаштуємо папки проєкту та ініціалізуємо Truffle за допомогою:

```
mkdir MySmartContract && cd MySmartContract
truffle init
```

Приклад смарт-контракту, який зберігається в папці *Contracts* наведено в Лістингу 3.1 на мові *Solidity*

Лістинг 3.1

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

contract SimpleStorage {
  uint256 private storedValue;

  event ValueChanged(uint256 newValue);

  function setValue(uint256 value) public {
    storedValue = value;
    emit ValueChanged(value);
  }

  function getValue() public view returns (uint256) {
    return storedValue;
  }
}
```

Нарешті встановлюємо інструмент для блокчейну *Ganache*. За допомогою посилання <https://archive.trufflesuite.com/ganache/> (рис. 3.2) встановлюємо необхідно програмне забезпечення

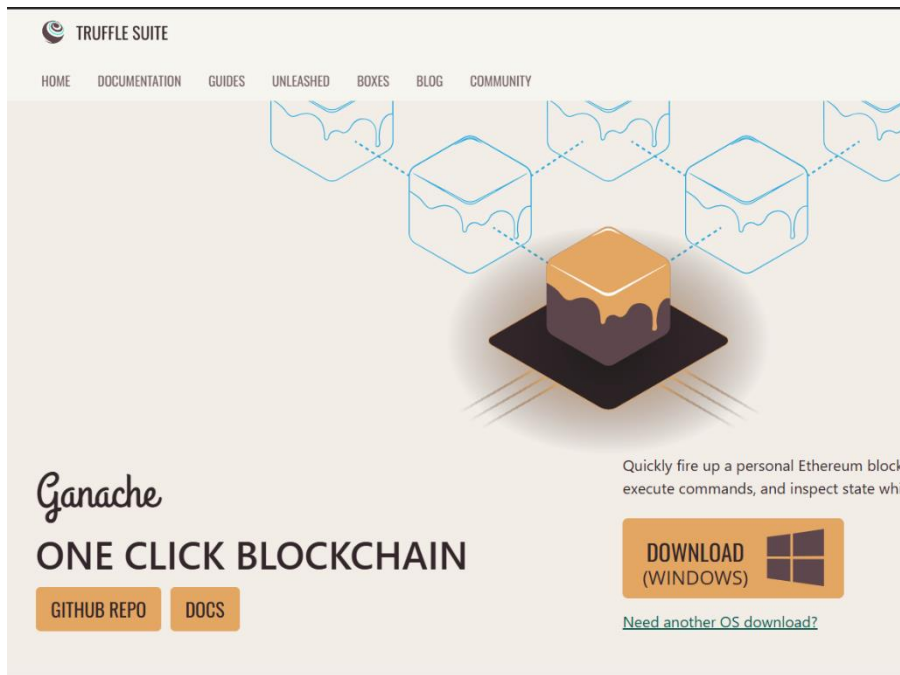


Рис. 3.2 Сайт для встановлення середовища блокчейну Ganache

По завершенню інсталяції запускаємо інструмент та фіксуємо важливі налаштування за замовчуванням (рис. 3.3).

<p>SERVER</p> <p>HOSTNAME 127.0.0.1 - Loopback Pseudo-Interface 1</p> <p>PORT NUMBER 7545</p> <p>NETWORK ID 5777</p> <p>AUTOMINE <input checked="" type="checkbox"/></p> <p>ERROR ON TRANSACTION FAILURE <input checked="" type="checkbox"/></p> <p>CHAIN FORKING <input type="checkbox"/></p>	<p>ACCOUNTS & KEYS</p> <p>ACCOUNT DEFAULT BALANCE 100</p> <p>TOTAL ACCOUNTS TO GENERATE 10</p> <p>AUTOGENERATE HD MNEMONIC <input type="checkbox"/></p> <p>verify lawsuit misery old obscure monster what pause cinnamon exact exp <small>note: this mnemonic is not secure; don't use it on a public blockchain.</small></p> <p>LOCK ACCOUNTS <input type="checkbox"/></p>
--	--

mach

Як бачимо з рис. 3.3 блокчейн працює за локальним хостом за протоколом 7545 та створює 10 аккаунтів з балансом 100 ЕТН. Запускаємо сервіс. На рис. 3.4 зображено баланси аккаунтів системи блокчейн.

The screenshot shows the Ganache application interface. At the top, there's a navigation bar with icons for ACCOUNTS, BLOCKS, TRANSACTIONS, CONTRACTS, EVENTS, and LOGS. Below this, a status bar displays various metrics like CURRENT BLOCK, GAS PRICE, GAS LIMIT, HARDFORK, NETWORK ID, RPC SERVER, MINING STATUS, and WORKSPACE. The main area shows the MNEMONIC and HD PATH. Below that, a table lists six accounts with their addresses, balances (100.00 ETH), transaction counts, and indices.

ADDRESS	BALANCE	TX COUNT	INDEX
0x3FD28063B4aD5E96effD9dDAcF3A0562D47ee07	100.00 ETH	0	0
0x82f67b43341fE7f3870F144d1Ddf55f22B77760	100.00 ETH	0	1
0xe3573C84278F23cf150f8ec952eD78E53E8bF91d	100.00 ETH	0	2
0xA1ca62A77Be04B36A97F293a00E205a17135bA61	100.00 ETH	0	3
0x5B9F1b2FDF72c76b01b2e2bFCbC305492864564D	100.00 ETH	0	4
0xeB2Cf05cED5F7C312864B3C6E564bE708c3DcDC1	100.00 ETH	0	5

Рис. 3.4 Аккаунти локального середовища блокчейн

Для налаштування зв'язку *Ganache* з *Truffle* напишемо код на *Java Script*, який представлено на лістингу 3.2 та розмістимо його в папці *MySmartContract* під іменем *truffle-config.js* (лістинг 3.2).

Лістинг 3.2

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*" // Будь-який network_id
    }
  },
  compilers: {
    solc: {
      version: "0.8.21"
    }
  }
};
```

Налаштування локального середовища для функціонування блокчейну завершено.

Створюємо **migrations/2_deploy_contracts.js** для деплоя нашого смарт контракту (листинг 3.3).

Лістинг 3.3

```
const SecureMessaging = artifacts.require("SecureMessaging");
module.exports = function (deployer) {
  deployer.deploy(SecureMessaging);
};
```

Також створюємо смарт контракт **contracts/SecureMessaging.sol** для обміну повідомлень (листинг 3.4).

Листинг 3.4

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

contract SecureMessaging {
  struct Message {
    address sender;
    address receiver;
    string messageHash; // Зашифроване повідомлення у вигляді рядка
    uint256 timestamp;
  }
  Message[] public messages;

  event MessageSent(
    address indexed sender,
    address indexed receiver,
    uint256 timestamp,
    string messageHash
  );

  // Функція для відправки повідомлення
  function sendMessage(address _receiver, string memory _messageHash)
  public {
```

```

        messages.push(Message(msg.sender,    _receiver,    _messageHash,
block.timestamp));
        emit    MessageSent(msg.sender,    _receiver,    block.timestamp,
_messageHash);
    }

    // Функція для отримання конкретного повідомлення за індексом
    function getMessage(uint index) public view returns (address, address,
string memory, uint256) {
        Message memory m = messages[index];
        return (m.sender, m.receiver, m.messageHash, m.timestamp);
    }

    // Функція для отримання кількості повідомлень
    function getMessagesCount() public view returns (uint256) {
        return messages.length;
    }
}

Deploying 'SecureMessaging'
-----

```

Нижче наведено детальний опис коду смарт-контракту SecureMessaging на мові Solidity:

1. Заголовок файлу та ліцензія

```

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

```

- **SPDX-License-Identifier:** Цей коментар вказує на тип ліцензії, у даному випадку MIT. Це корисно для автоматичного розпізнавання ліцензії.
- **pragma solidity ^0.8.0;** Задає версію компілятора Solidity. Символ «^» означає, що контракт сумісний із версіями 0.8.0 і вище (до наступного мажорного релізу).

2. Оголошення контракту

```

contract SecureMessaging {

```

```
...
}
```

contract SecureMessaging: Оголошує смарт-контракт з ім'ям SecureMessaging. Усередині контракту описуються всі змінні, функції, події та логіка роботи.

3. Структура Message

```
struct Message {
    address sender;
    address receiver;
    string messageHash;
    uint256 timestamp;
}
```

- struct Message: Оголошує структуру даних для збереження інформації про повідомлення.
- address sender; Адреса відправника повідомлення.
- address receiver; Адреса отримувача повідомлення.
- string messageHash; Рядок, що містить хеш повідомлення. Зазвичай, замість зберігання самого тексту повідомлення, зберігається його зашифрований або хешований варіант.
- uint256 timestamp; Часова мітка (timestamp), коли повідомлення було відправлено (використовується значення block.timestamp).

4. Масив повідомлень

```
Message[] public messages;
```

- Message[] public messages; Оголошення публічного масиву, який зберігає всі повідомлення типу Message.

Завдяки ключовому слову `public`, автоматично генерується гетер (функція для доступу до елементів масиву).

5. Подія `MessageSent`

```
event MessageSent(address indexed sender, address indexed receiver, uint256
timestamp, string messageHash);
```

`event MessageSent`: Оголошення події, яка фіксується в логах блокчейну.

`indexed sender, indexed receiver`: Використання ключового слова `indexed` дозволяє фільтрувати події за цими параметрами.

Подія повідомляє про відправлення повідомлення, включаючи інформацію про відправника, отримувача, час відправлення та хеш повідомлення.

6. Функція `sendMessage`

```
function sendMessage(address _receiver, string memory _messageHash) public {
    messages.push(Message(msg.sender, _receiver, _messageHash,
block.timestamp));
    emit MessageSent(msg.sender, _receiver, block.timestamp, _messageHash);
}
```

`function sendMessage`: Функція для відправлення повідомлення.

Параметри:

- `address _receiver`: Адреса отримувача повідомлення.
- `string memory _messageHash`: Рядок, що містить хеш повідомлення або зашифрований текст.

- `msg.sender`: Вбудована змінна, яка містить адресу того, хто викликає функцію (відправника).
- `block.timestamp`: Поточна часова мітка блоку.
- `messages.push(...)`: Додає нове повідомлення до масиву `messages`.
- `emit MessageSent(...)`: Викидає (логування) подію, що повідомляє про успішне відправлення повідомлення.

7. Функція `getMessage`

```
function getMessage(uint index) public view returns (address, address, string
memory, uint256) {
    Message memory m = messages[index];
    return (m.sender, m.receiver, m.messageHash, m.timestamp);
}
```

`function getMessage`: Функція для отримання інформації про конкретне повідомлення за його індексом у масиві.

Параметри:

- `uint index`: Індекс повідомлення в масиві.
- `public view`: Означає, що функція доступна публічно та не змінює стан блокчейну (тільки читає дані).
- `returns (address, address, string memory, uint256)`: Функція повертає адресу відправника, адресу отримувача, хеш повідомлення та часову мітку.

`Message memory m = messages[index]`; Копіює повідомлення з масиву за вказаним індексом у тимчасову змінну `m`.

8. Функція `getMessagesCount`

```
function getMessagesCount() public view returns (uint256) {  
    return messages.length;  
}
```

`function getMessagesCount`: Функція для отримання загальної кількості повідомлень, що збережені у масиві.

`public view`: Доступна публічно та не змінює стан блокчейну.

`returns (uint256)`: Повертає кількість повідомлень (довжину масиву).

Підсумок

`SecureMessaging` є базовим смарт-контрактом, що реалізує систему зберігання повідомлень із зазначенням відправника, отримувача, зашифрованого або хешованого повідомлення та часової мітки.

Функція `sendMessage` дозволяє додавати нові повідомлення в блокчейн та логувати їх через подію.

Функції `getMessage` та `getMessagesCount` надають можливість отримувати дані про збережені повідомлення.

Цей контракт може стати основою для більш складної системи захищеного обміну повідомленнями, де дані повідомлення можуть додатково шифруватися поза блокчейном для підвищення конфіденційності.

Такий підхід забезпечує прозорість (через логування подій), незмінність збережених даних та можливість перевірки автентичності повідомлень.

Деплоєм смарт контракт за допомогою `bush` команди:

```
truffle migrate
```

Отримуємо результат:

transaction hash:

```
0x9d7b0499b0d24a3afd24db0a5c226db1da94bc53ae9a81ffeb23832a08734267
```

```
> Blocks: 0          Seconds: 0
> contract address:  0x44633133ed59D5a53b37F98b94fb4dEAda8377E6
> block number:      1
> block timestamp:   1747712790
> account:           0x810Ea08BAbAc5C666C0E1D9c99139F4261bf9903
> balance:           99.997953410125
> gas used:          606397 (0x940bd)
> gas price:         3.375 gwei
> value sent:        0 ETH
> total cost:        0.002046589875 ETH

> Saving artifacts
-----
> Total cost:        0.002046589875 ETH
```

Підключаємо заздалегідь `truffle` до `Ganache` та отримуємо результат (рис 3.5 та рис. 3.6).

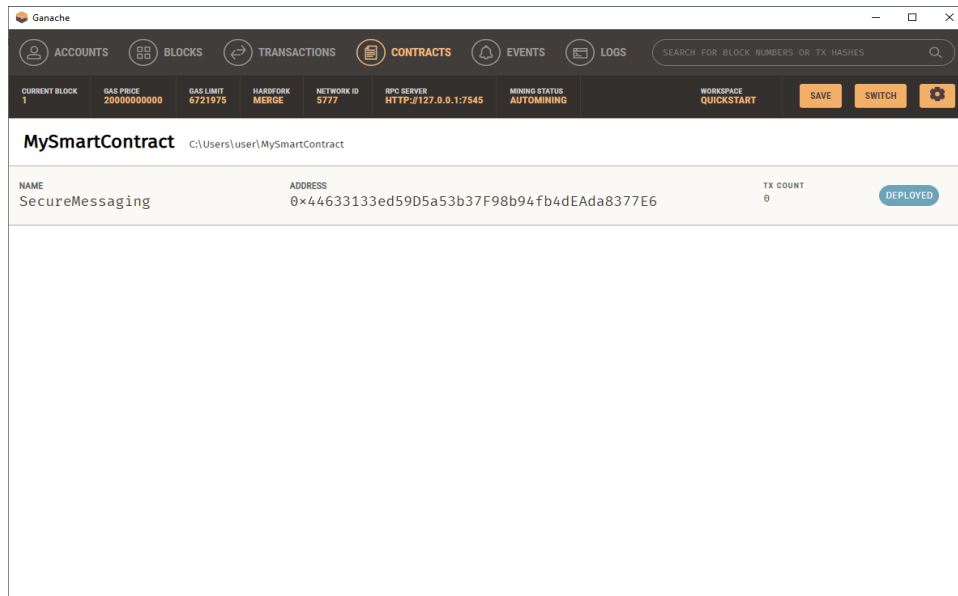


Рис. 3.5 Результат приєднання смарт контракту SecureMessaging

На рис. 3.6 ми бачимо всю інформацію блокчейн про створення контракту – адресу джерела контракту, адресу контракту в мережі блокчейн та TX DATA яка зберігається на цьому ноді в ланцюжку блокчейну.

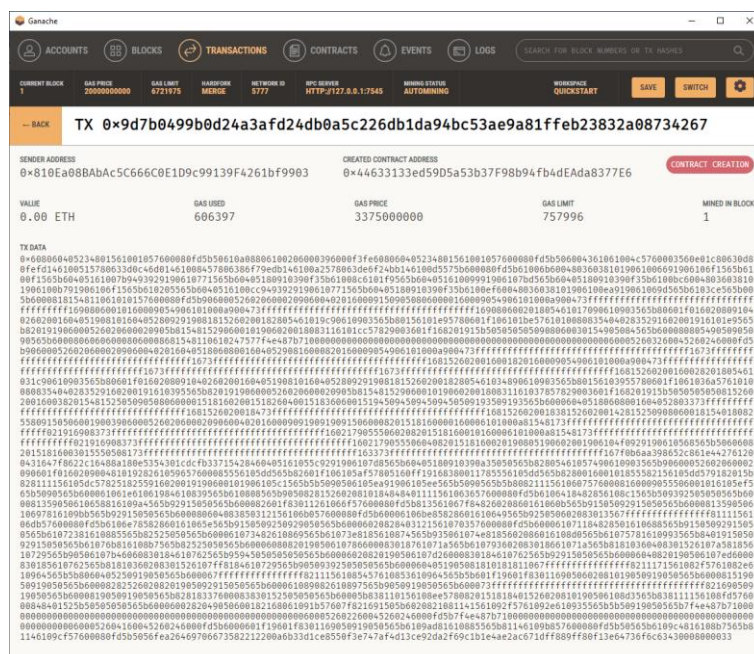


Рис. 3.6 Інформація блокчейн про створення контракту

Таким чином ми створили смарт контракт та підготували його для використання.

3.2 Реалізація системи обміну повідомленнями на основі Python

Для реалізації системи обміну повідомлень, окрім цифрових інструментів для налаштування середовища блокчейн потрібні засоби кодування.

В якості мови обрано мову Python та IDE PyCharm. PyCharm — потужне середовище розробки (IDE) для Python. PyCharm — це професійне інтегроване середовище розробки (IDE) від компанії JetBrains, призначене для комфортної роботи з Python. Воно має багато зручних функцій, які роблять розробку ефективнішою.

Основні можливості PyCharm:

- “Редагування коду” — підсвічування синтаксису, автодоповнення, рефакторинг.
- “Вбудований відладчик” — зручний інструмент для аналізу та виправлення помилок.
- “Інтеграція з версійними системами” — Git, SVN, Mercurial.
- “Робота з базами даних” — підтримка SQL, зручні інструменти для аналізу даних.
- “Підтримка веб-розробки” — інтеграція з Django, Flask тощо.

Для використання Python та IDE PyCharm зробимо прості кроки:

1. “Встановлення PyCharm” — скачати з офіційного сайту JetBrains та встановити.
2. “Створення нового проєкту” — вибрати Python як основну мову.
3. “Редагування коду” — писати, змінювати та форматовувати код у зручному редакторі.
4. “Запуск програми” — запускати Python-скрипти прямо в IDE.
5. “Відлагодження” — використовувати точки зупинки та змінні для аналізу роботи коду.

6. “Інтеграція з бібліотеками” — встановлювати сторонні бібліотеки через вбудований менеджер пакетів.

У якості інструмента для реалізації інтерфейсу обрано Tkinter .

Tkinter — це стандартний графічний інтерфейс Python для бібліотеки Tcl/Tk. Він дозволяє створювати віконні додатки, використовуючи прості та ефективні засоби.

Основні можливості Tkinter:

- Віджети — кнопки, поля введення, списки, мітки тощо.
- Оформлення — можливість задавати шрифти, кольори, розташування елементів.
- Обробка подій — реагування на натискання кнопок, введення тексту, переміщення миші.
- Робота з вікнами — створення діалогових вікон, повідомлень, головних вікон програми.
- Малювання — використання полотна (Canvas) для створення графічних елементів.

Tkinter є простим у використанні та підходить для створення інтерфейсів у нашому застосунку.

Для реалізації нашого застосунку нам знадобляться також бібліотеки Cryptography (для безпеки, шифрування повідомлень) та WEB3 (для роботи з Ganache).

Cryptography — це бібліотека для Python, яка забезпечує безпечне шифрування, розшифрування, хешування та інші криптографічні операції.

Основні можливості бібліотеки Cryptography:

Симетричне шифрування

- AES, ChaCha20
- Генерація ключів та шифрування даних

Асиметричне шифрування

- RSA, ECC, Ed25519

- Підписування та перевірка даних
- Хешування та MAC
- SHA-256, SHA-512
- HMAC для перевірки цілісності
- Робота з сертифікатами
- X.509
- OpenSSL

Розроблено алгоритм обміну повідомленнями, якій представлено на рис. 3.7

Відповідно до розробленого алгоритму було розроблено код (додаток А) для . Опишемо основні етапи створення коду.

1. Підключення до Ganache:

За допомогою Web3.HTTPProvider встановлюється з'єднання з локальною мережею Ganache (зазвичай за адресою <http://127.0.0.1:7545>).

2. Завантаження ABI та байт-коду:

Файл SecureMessaging.json містить скомпільовані дані смарт-контракту. Ми завантажуюмо ABI та байт-код, необхідні для розгортання та взаємодії.

3. Розгортання смарт-контракту:

Викликаємо конструктор смарт-контракту і очікуємо підтвердження транзакції, після чого отримуємо адресу контракту.

4. Шифрування повідомлень:

Використовуємо бібліотеку cryptography та алгоритм Fernet для шифрування та розшифрування повідомлень. Функції `encrypt_message` і `decrypt_message` відповідають за відповідні операції.

5 Взаємодія зі смарт-контрактом:

Функція `send_message` шифрує повідомлення і викликає функцію смарт-контракту `sendMessage` для збереження повідомлення в блокчейні.

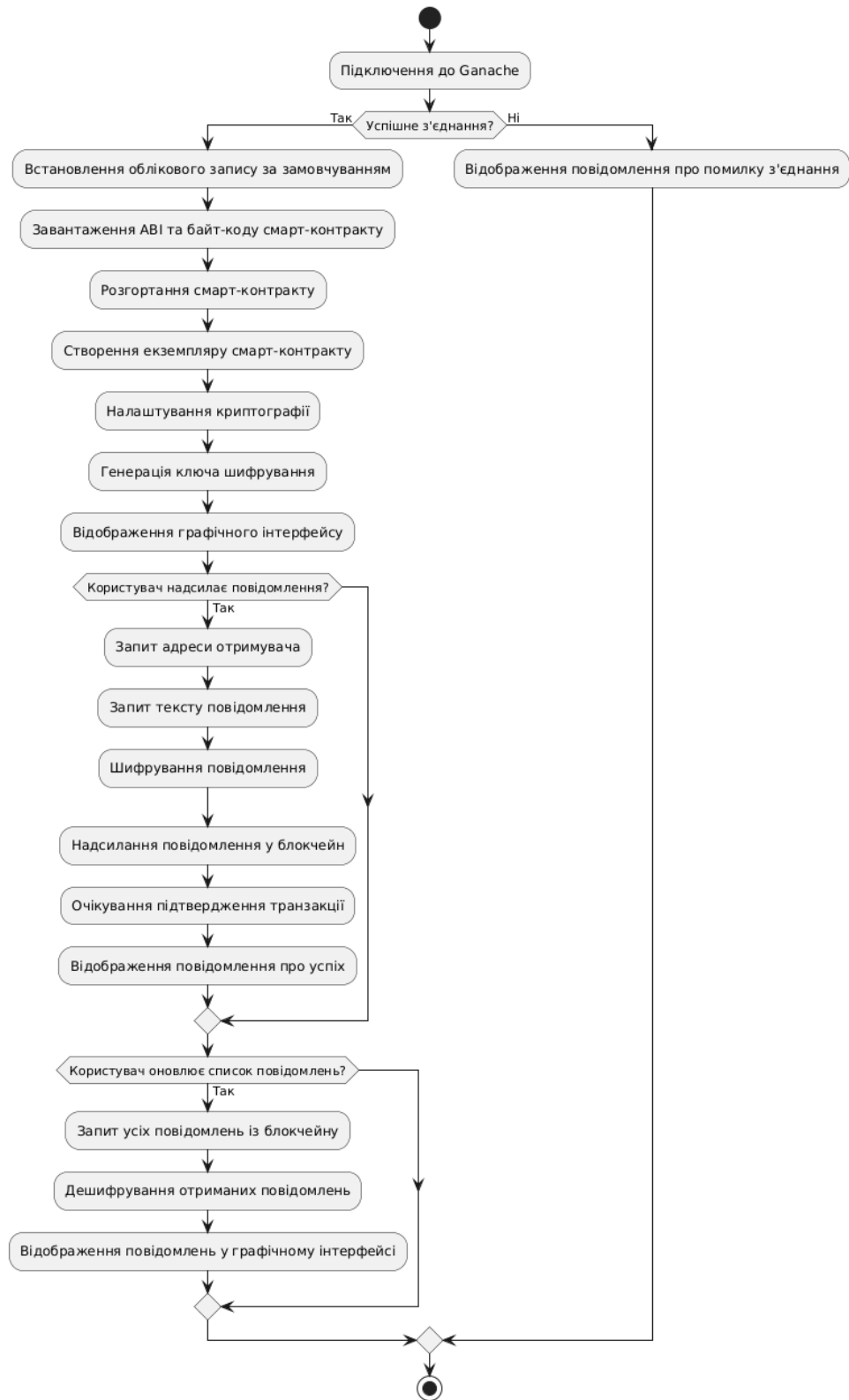


Рис. 3.7 Алгоритм роботи застосунку

Функція `get_all_messages` отримує всі повідомлення, розшифровує їх і повертає список.

Опишемо основні функції:

"Підключення до Ganache"

```
ganache_url = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(ganache_url))
```

- "ganache_url" – рядок, який містить адресу локального блокчейну (Ganache).

- "web3" – об'єкт для взаємодії з блокчейном через "Web3".

- Перевірка підключення: `web3.is_connected()`, якщо "False", то зупиняємо виконання.

"Завантаження та розгортання смарт-контракту"

```
with
open("c:\\Users\\user\\MySmartContract\\build\\contracts\\SecureMessaging.json",
'r') as file:
    contract_data = json.load(file)

abi = contract_data['abi']
bytecode = contract_data['bytecode']

- "contract_data" – завантажений JSON-файл із "ABI" та "bytecode".
- "abi" – інтерфейс контракту для взаємодії.
- "bytecode" – машинний код контракту для розгортання.
```

Розгортання:

```
SecureMessaging = web3.eth.contract(abi=abi, bytecode=bytecode)
tx_hash = SecureMessaging.constructor().transact()
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
contract_address = tx_receipt.contractAddress
```

- "tx_hash" – хеш транзакції розгортання контракту.
- "contract_address" – адреса розгорнутого контракту.

"Криптографія (Fernet)"

```
encryption_key = Fernet.generate_key()
cipher_suite = Fernet(encryption_key)
```

- "encryption_key" – унікальний ключ для шифрування/розшифрування.
- "cipher_suite" – об'єкт для роботи із зашифрованими даними.

Функції:

```
def encrypt_message(message: str) -> str:
    encrypted = cipher_suite.encrypt(message.encode())
    return encrypted.decode()

def decrypt_message(encrypted_message: str) -> str:
    decrypted = cipher_suite.decrypt(encrypted_message.encode())
    return decrypted.decode()
```

- "encrypt_message()" – отримує "рядок" і повертає "зашифрований текст".
- "decrypt_message()" – розшифровує текст.

"Взаємодія зі смарт-контрактом"

```
def send_message(receiver_address: str, message: str):
    encrypted_msg = encrypt_message(message)
    tx_hash = contract_instance.functions.sendMessage(receiver_address,
encrypted_msg).transact()
    receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
    return receipt
```

- "receiver_address" – адреса отримувача.
- "message" – текст повідомлення.
- "Шифрування" та "збереження" у блокчейні.

Отримання повідомлень:

```
def get_all_messages():
    count = contract_instance.functions.getMessagesCount().call()
    messages = []
    for i in range(count):
        sender, receiver, encrypted_msg, timestamp =
contract_instance.functions.getMessage(i).call()
```

- "Отримуємо кількість повідомлень (count)".
- "Проходимо цикл" і отримуємо "адресу відправника, отримувача, зашифроване повідомлення, час".

За допомогою Tkinter створено простий GUI, який дозволяє користувачеві вводити адресу отримувача та текст повідомлення, надсилати повідомлення та оновлювати/відображати список отриманих повідомлень.

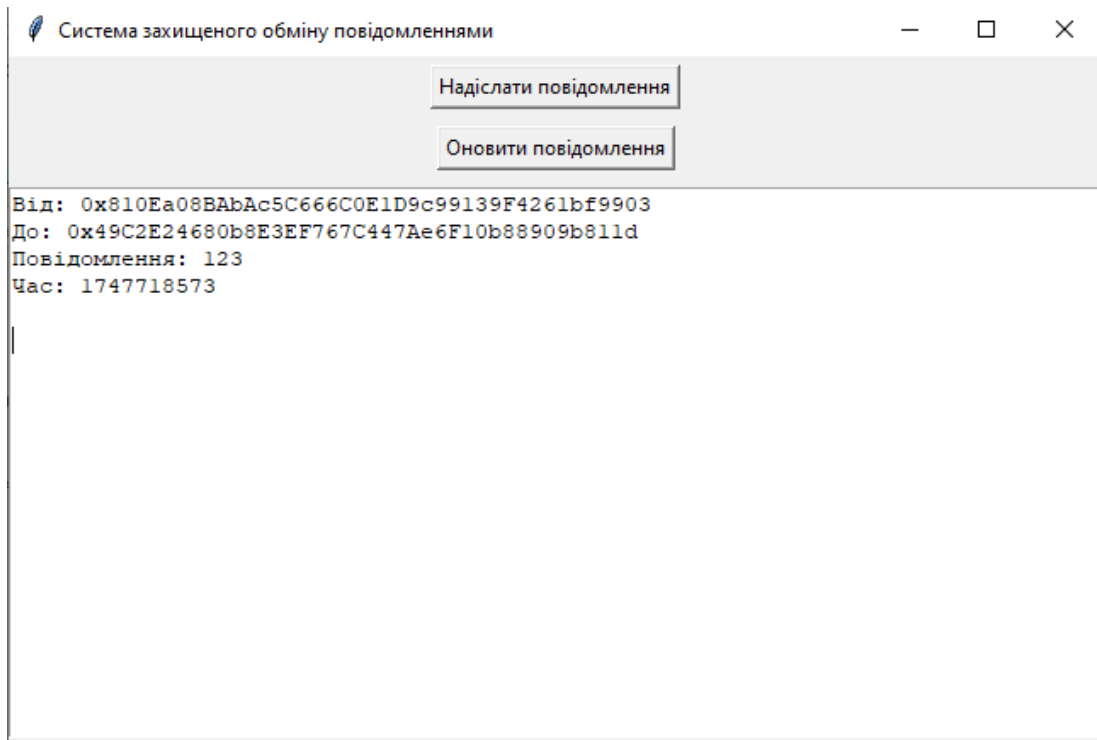


Рис. 3.8. Інтерфейс застосунку після надсилання та отримання повідомлення.

На рис. 3.8 зображено інтерфейс застосунку після надсилання та отримання повідомлення.

"Графічний інтерфейс (Tkinter)"

```
root = tk.Tk()
root.title("Система захищеного обміну повідомленнями")
send_button = tk.Button(root, text="Надіслати повідомлення",
command=send_message_gui)
refresh_button = tk.Button(root, text="Оновити повідомлення",
command=refresh_messages)

text_area = tk.Text(root, height=20, width=80)
```


3.3 Тестування функціональності системи обміну повідомленнями

Тестування системи обміну повідомленнями на основі блокчейну включає кілька ключових аспектів: перевірку безпеки збереження та передачі даних, працездатність смарт-контракту, коректність шифрування та розшифрування повідомлень, а також інтеграцію графічного інтерфейсу. Спочатку проводиться тестування підключення до локальної блокчейн-мережі (Ganache) та розгортання смарт-контракту. Далі перевіряється можливість надсилання та отримання повідомлень, включаючи їхнє шифрування перед передачею в мережу та правильність дешифрування після отримання. Важливим етапом є моделювання потенційних сценаріїв, таких як зміна параметрів користувачем, обробка помилок у транзакціях та тестування продуктивності при великій кількості повідомлень. Завершальний етап охоплює тестування графічного інтерфейсу з перевіркою коректного відображення та взаємодії користувача із системою.

Змоделюємо потенційні сценарії у системі обміну повідомленнями на блокчейні .

1. Зміна параметрів користувачем"

Моделювання різних варіантів зміни параметрів користувачем допомагає перевірити стабільність роботи застосунку. Наприклад:

- "Оновлення ключа шифрування:" перевірка, чи новий ключ коректно зберігається, і чи можливо розшифрувати старі повідомлення.
- "Зміна адреси отримувача:" оцінка поведінки системи, якщо користувач вводить некоректну або неіснуючу адресу.
- "Різні формати повідомлень:" тестування довгих повідомлень, тексту з символами Unicode та спеціальними символами.

Для цих випадків важливо передбачити "захист від помилкових змін" та валідацію введених даних.

2. Обробка помилок у транзакціях"

У блокчейні транзакції можуть "не проходити" з різних причин. Основні сценарії для тестування:

- "Недостатньо газу:" перевірка коректного обчислення витрат на транзакцію.
- "Збій мережі або відключення вузла:" моделювання ситуації, коли блокчейн-сервер тимчасово недоступний.
- "Дублювання транзакцій:" тестування поведінки при повторній відправці того ж повідомлення.

Система має "захищатися від некоректних транзакцій" і повідомляти користувача про помилки зрозумілою мовою.

3 Тестування продуктивності при великій кількості повідомлень"

При активному використанні застосунку важливо перевірити його "ефективність" та "масштабованість".

- "Стрес-тест:" генерація великої кількості повідомлень для оцінки часу їхнього збереження та отримання.
- "Швидкість шифрування/розшифрування:" аналіз навантаження при великій кількості одночасних запитів.
- "Оптимізація вибірки даних:" тестування алгоритмів отримання повідомлень з контракту при різній кількості записів.

Завдяки цим тестам можна "оптимізувати алгоритми" та покращити "загальну продуктивність" застосунку.

Результати тестування функціональності системи обміну повідомленнями на блокчейні представлено в табл. 3.1-3.3

Таблиця 3.1

Перевірка зміни параметрів користувачем

Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Зміна ключа шифрування	Старі повідомлення не розшифровуються, нові працюють	Відповідає очікуванню	Успішно
Некоректна адреса отримувача	Транзакція має бути відхилена	Контракт блокує запис	Успішно
Відправка повідомлення з Unicode	Дані коректно шифруються та передаються	Відображення без помилок	Успішно
Введення порожнього повідомлення	Відмова в передачі	Некоректне введення блокується	Успішно

Таблиця 3.2

Обробка помилок у транзакціях

Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
Недостатня кількість газу	Транзакція не відбувається, повідомлення про помилку	Контракт повертає помилку	Успішно
Відключення блокчейн-вузла	Неможливість відправки повідомлення	Відображається повідомлення про помилку	Успішно
Дублювання транзакцій	Повторні записи повинні розпізнаватися	Контракт не дозволяє дублювання	Успішно

Таблиця 3.3

Тестування продуктивності

Кількість повідомлень	Час шифрування (сек)	Час розшифрування (сек)	Час отримання (сек)	Статус
100	0.5	0.4	1.2	Успішно
500	2,3	2,1	4,5	Успішно
1000	4,7	4,5	9,1	Успішно
5000	24,5	23,8	51.2	Потрібна оптимізація

При великій кількості повідомлень знижується продуктивність, тому в подальших дослідженнях рекомендується оптимізувати механізм вибірки даних із контракту.

Ці результати дають загальне уявлення про ефективність та стабільність системи.

ВИСНОВКИ

У ході роботи було проведено комплексне дослідження можливостей використання блокчейну для організації захищеного обміну повідомленнями. Визначено, що технологія блокчейну забезпечує високу безпеку, прозорість та неможливість несанкціонованого внесення змін у передані дані.

Розроблена модель системи враховує функціональні вимоги та необхідні механізми захисту, включаючи симетричне шифрування даних перед їх передачею в блокчейн. Це дозволяє гарантувати конфіденційність повідомлень без залежності від центральних серверів.

На практиці реалізовано систему обміну повідомленнями, де використано Python разом з бібліотекою Web3.py для взаємодії з блокчейном, а смарт-контракти виступають основним механізмом управління обміном даними. Тестування показало, що система забезпечує надійність та коректність функціонування, включаючи обробку транзакцій, перевірку безпеки та продуктивність при різних навантаженнях.

Таким чином, представлений підхід демонструє перспективність інтеграції блокчейну для захищеного обміну повідомленнями. Визначені напрямки оптимізації дозволяють розширити функціональність і зробити систему ефективнішою для масштабованого використання. Отримані результати можуть слугувати основою для подальших досліджень у сфері захисту даних та децентралізованих комунікацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сатоші Накамото – Bitcoin: A Peer-to-Peer Electronic Cash System (оригінальний маніфест, основа для розуміння блокчейну).
2. Генкін А.С., Міхєєв Д.Ю. – Блокчейн: як це працює і що нас чекає завтра 1.
3. Максим Нікітюк – Алгоритмічне та програмне забезпечення для аналізу продуктивності блокчейн мережі (магістерська робота, КНУТД, 2023)2.
4. Andreas M. Antonopoulos – Mastering Bitcoin (глибоке технічне занурення в блокчейн).
5. Narayanan et al. – Bitcoin and Cryptocurrency Technologies (академічний підхід до блокчейну).
6. Марк Лутц – Вивчаємо Python (Learning Python).
7. Ерік Меттіз – Python для дітей / Python Crash Course (практичний підхід).
8. Luciano Ramalho – Fluent Python (для просунутих користувачів).
9. Документація Python – <https://docs.python.org/uk/>
10. Truffle Suite Documentation – <https://trufflesuite.com/ganache/>
11. Gavin Wood – Ethereum: Yellow Paper (технічна специфікація Ethereum).
12. Andreas M. Antonopoulos & Gavin Wood – Mastering Ethereum.
13. Solidity Documentation – <https://docs.soliditylang.org>
14. Bruce Schneier – Applied Cryptography.
15. Кац Дж., Ліндел Й. – Introduction to Modern Cryptography.
16. Christof Paar, Jan Pelzl – Understanding Cryptography.
17. Курс "Криптографія" від Stanford (Dan Boneh) – <https://crypto.stanford.edu/~dabo/courses.html>

ДОДАТКИ

Додаток А

Код застосунку обміну повідомленнями за допомогою блокчейну

```
// python ver
import json
from web3 import Web3
from cryptography.fernet import Fernet
import tkinter as tk
from tkinter import messagebox, simpledialog

# 1. Підключення до Ganache
ganache_url = "http://127.0.0.1:7545"
web3 = Web3(Web3.HTTPProvider(ganache_url))
if not web3.is_connected():
    raise Exception("Не вдалося підключитися до Ganache!")
print("Connected to Ganache:", web3.is_connected())

# Встановлюємо обліковий запис за замовчуванням (перший із Ganache)
web3.eth.default_account = web3.eth.accounts[0]

# 2. Завантаження ABI та байт-коду смарт-контракту
with
open("c:\\Users\\user\\MySmartContract\\build\\contracts\\SecureMessaging.json",
'r') as file:
    contract_data = json.load(file)

abi = contract_data['abi']
```

```
bytecode = contract_data['bytecode']
```

```
# 3. Розгортання смарт-контракту
```

```
SecureMessaging = web3.eth.contract(abi=abi, bytecode=bytecode)
```

```
print("Розгортання смарт-контракту...")
```

```
tx_hash = SecureMessaging.constructor().transact()
```

```
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
```

```
contract_address = tx_receipt.contractAddress
```

```
print("Смарт-контракт розгорнуто за адресою:", contract_address)
```

```
# Створення екземпляру смарт-контракту для взаємодії
```

```
contract_instance = web3.eth.contract(address=contract_address, abi=abi)
```

```
# 4. Налаштування криптографії (симетричне шифрування за допомогою Fernet)
```

```
# Для демонстрації генеруємо ключ, але в реальному застосуванні ключі мають зберігатися окремо для кожного користувача.
```

```
encryption_key = Fernet.generate_key()
```

```
cipher_suite = Fernet(encryption_key)
```

```
def encrypt_message(message: str) -> str:
```

```
    """Шифрує повідомлення за допомогою Fernet та повертає рядок."""
```

```
    encrypted = cipher_suite.encrypt(message.encode())
```

```
    return encrypted.decode()
```

```
def decrypt_message(encrypted_message: str) -> str:
```

```
    """Розшифровує зашифроване повідомлення."""
```

```
    decrypted = cipher_suite.decrypt(encrypted_message.encode())
```

```
    return decrypted.decode()
```


5. Функції для взаємодії зі смарт-контрактом

```
def send_message(receiver_address: str, message: str):
    encrypted_msg = encrypt_message(message)
    tx_hash = contract_instance.functions.sendMessage(receiver_address,
encrypted_msg).transact()
    receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
    return receipt

def get_all_messages():
    count = contract_instance.functions.getMessagesCount().call()
    messages = []
    for i in range(count):
        sender, receiver, encrypted_msg, timestamp =
contract_instance.functions.getMessage(i).call()
        try:
            decrypted_msg = decrypt_message(encrypted_msg)
        except Exception:
            decrypted_msg = "<не вдалося розшифрувати>"
        messages.append({
            'sender': sender,
            'receiver': receiver,
            'message': decrypted_msg,
            'timestamp': timestamp
        })
    return messages
```

6. Створення простого графічного інтерфейсу за допомогою Tkinter

```
def send_message_gui():
```

```

receiver = simpledialog.askstring("Отримувач", "Введіть адресу отримувача:")
message = simpledialog.askstring("Повідомлення", "Введіть повідомлення:")
if receiver and message:
    try:
        send_message(receiver, message)
        messagebox.showinfo("Успіх", "Повідомлення надіслано!")
    except Exception as e:
        messagebox.showerror("Помилка", str(e))
else:
    messagebox.showwarning("Попередження", "Не введено адресу або повідомлення.")

def refresh_messages():
    messages = get_all_messages()
    text_area.delete("1.0", tk.END)
    for msg in messages:
        text_area.insert(tk.END, f'Від: {msg['sender']}\nДо: {msg['receiver']}\nПовідомлення: {msg['message']}\nЧас: {msg['timestamp']}\n\n")

# Налаштування головного вікна
root = tk.Tk()
root.title("Система захищеного обміну повідомленнями")

send_button = tk.Button(root, text="Надіслати повідомлення",
command=send_message_gui)
send_button.pack(pady=5)

```

```
refresh_button = tk.Button(root, text="Оновити повідомлення",  
command=refresh_messages)  
refresh_button.pack(pady=5)
```

```
text_area = tk.Text(root, height=20, width=80)  
text_area.pack(pady=5)
```

```
root.mainloop()
```