Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Філюка Євгена Вікторовича

на здобуття ступеня вищої освіти Бакалавра

Система конфіденційності та безпеки криптоактивів
на основі технології Multi-Party Computation

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ.200116.20.01.18 ПЗ

Виконав студент 4 курсу група КБ-20-1 Євген ФІЛЮК
Керівник канд. техн. наук, доцент Володимир ДЖУЛІЙ
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ
19 червень 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Філюку Євгену Вікторовичу

1 Тема роботи Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation

Керівник роботи Джулій Володимир Микойович. канд. техн. наук, доцент

Затверджено наказом ректора університету від 15 лютого 2024 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру 12.06.2024

3 Вихідні дані до роботи Розробити ефективну систему конфідеційності та безпеки керування криптоактивами на основі технології багатосторонніх обчислень. Дослідити предметну область, що стосується децентралізованої системи переказу активів. Дослідити наявні методи та протоколи захисту. На основі проведених досліджень розробити протокол багатосторонніх обчислень приватних ключів та підписів до криптотранзакції. Провести тестування розробленої системи.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області та постановка задачі .Проєктування системи керування криптоактивами на основі технології Multi-Party Computation. Розробка системи керування криптоактивами на основі технології багатосторонніх обчислень

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
«Принципи роботи блокчейн технології», «Алгоритм роботи протоколу багатосторонніх обчислень системи», «Схема взаємодії вузлів програмного забезпечення системи», «Розгорнута система керування криптоактивами на основі технології багатосторонніх обчислень»

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Мостовий С.В., старший викладач кафедри кібербезпеки		

7 Дата видачі завдання 16 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконано
Ознайомлення з предметною областю	Лютий	виконано
Дослідження існуючих рішень	Лютий	виконано
Постановка задачі	Березень	виконано
Визначення загальних принципів рішення задачі	Березень	виконано
Деталізація принципів рішення задачі	Квітень	виконано
Розробка проєктних рішень	Квітень	виконано
Апробація проєктних рішень	Травень	виконано
Оформлення пояснювальної записки згідно вимог	Травень	виконано
Оформлення графічної частини	Червень	виконано
Захист КР	Червень	виконано

Студент _____ Євген ФІЛЮК

Керівник кваліфікаційної роботи _____ Володимир ДЖУЛІЙ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation.

Автор роботи: Філюк Євген Вікторович.

Керівник роботи: Джулій Володимир Миколайович.

Пояснювальна записка: 66 с., 2 додатки, 31 рис., 42 джерел.

Графічна частина: 4 плакати, 11 презентаційних слайдів.

СИСТЕМА КЕРУВАННЯ КРИПТОАКТИВАМИ, КОНФІДЕНЦІЙНА ІНФОРМАЦІЯ, БЕЗПЕКА КРИПТОАКТИВІВ, ТЕХНОЛОГІЯ MULTI-PARTY COMPUTATION.

Метою цієї роботи було створення системи конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation. Головним завданням було запобігання витоку конфіденційної інформації про приватний ключ через єдине джерело зберігання та забезпечення безпечних переказів криптоактивів.

Для досягнення цих цілей було проведено дослідження і аналіз області блокчейну, систем криптографічних гаманців, існуючих методів та протоколів на Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation, а також теоретичної інформації про такі системи і їх моделі створення. На основі аналізу була розроблена система, яка може розподілено згенерувати пару ключів та здійснювати підписання транзакцій багатьма приватними ключами учасників протоколу, використовуючи технології Multi-Party Computation.

Розроблену систему розгорнуто глобально для двох видів криптоактивів Ethereum та Bitcoin, забезпечено безпечний зв'язок та надано доступ до користування через API інтерфейс. Також було розгорнуто локально для тестування перед глобальним розгортанням. Це дозволило забезпечити розподілену генерацію пар ключів та безпечне підписання транзакцій, запобігши вразливості єдиної точки взлому.

12.06.2024

ANNOTATION

Course project: System of confidentiality and security of cryptoassets based on the Multi-Party Computation technology.

Author of the work: Filyuk Evgen Viktorovych.

Supervisor: Dzhuliy Volodymyr Mykolayovych.

Amount: 66 p., 2 appendix, 31 figures, 42 sources.

Graphic part: 4 schemes, 11 presentation slides.

CRYPTOASSETS MANAGEMENT SYSTEM, CONFIDENTIAL INFORMATION, SECURITY OF CRYPTOASSETS, MULTI-PARTY COMPUTATION TECHNOLOGY.

The purpose of this work was to create a system of confidentiality and security of cryptoassets based on the Multi-Party Computation technology. The main task was to prevent the leakage of confidential information about the private key through a single source of storage and to ensure secure transfers of crypto assets.

To achieve these goals, a study and analysis of the blockchain, cryptographic wallet systems, existing methods and protocols for the System of confidentiality and security of cryptoassets based on Multi-Party Computation technology, as well as theoretical information about such systems and their creation models, was carried out. Based on the analysis, a system was developed that can generate a pair of keys in a distributed manner and sign transactions with many private keys of protocol participants using Multi-Party Computation technologies.

The developed system is deployed globally for two types of cryptoassets Ethereum and Bitcoin, secure communication is provided and access to use is provided through the API interface. Also deployed locally for testing before global deployment. This allowed for distributed generation of key pairs and secure signing of transactions, preventing a single point of compromise.

12.06.2024

ЗМІСТ

					КРБКБ.200116.20.01.18 ПЗ		
Зм.	Арк.	№докум.	Підпис	Дата			
Виконав		Філюк Є.В.		19.06.24	Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation	Літера	Аркуш 6 / Аркушів
Перевір.		Джулій В.М		19.06.24			
Н.контр.		Мостовий С.В.		19.06.24		ХНУ, КБ-20-1	
Затвер.		Кльоц Ю.П.		19.06.24			

ВСТУП

З кожним роком популярність використання криптовалюти зростає, так само як і попит на безпечні криптовалютні гаманці. Люди використовують криптовалюту з різних причин: як засіб обміну та збереження вартості, інструмент інвестування та спосіб здійснення міжнародних переказів. Криптовалюти є цифровими активами, які використовують криптографічні протоколи для забезпечення безпеки транзакцій та контролю за створенням нових одиниць цифрової валюти. Вони працюють на базі технології блокчейн, яка дозволяє записувати транзакції у розподіленій базі даних, забезпечуючи безпеку та непідробність операцій.

Основною перевагою криптовалют є їх децентралізація: вони не контролюються центральними органами управління, такими як уряди чи банки, і можуть бути використані у будь-якій частині світу без обмежень. Додатковими перевагами є анонімність, що дозволяє користувачам зберігати конфіденційність своїх фінансових операцій, та зручність, що забезпечує швидкі та прості перекази без посередників. Найвідомішою криптовалютою є біткоїн, але на сьогоднішній день існує безліч інших криптовалют, таких як Ethereum, Ripple, Tether та інші.

Криптовалютні гаманці є ключовими інструментами для зберігання, відправлення та отримання криптовалюти. Вони можуть бути онлайн-сервісами, програмами для смартфонів або настільними програмами, дозволяючи користувачам відстежувати свої баланси та історію транзакцій. У зв'язку з цим створення ефективного та безпечного криптовалютного гаманця є актуальним завданням для багатьох галузей, які працюють з криптовалютами, таких як фінансові послуги, торгівля, інтернет-магазини, туризм та благодійність.

Сучасні криптовалютні гаманці розробляють на основі технології Multi-Party Computation (MPC). Використання MPC дозволяє значно підвищити безпеку криптовалютних транзакцій. Ця технологія забезпечує захист даних

навіть у випадку компрометації частини учасників системи, оскільки вона розподіляє обробку і зберігання інформації між декількома учасниками, уникаючи центральної точки уразливості.

MPC використовує криптографічні методи для розподіленої обробки даних, що дозволяє кільком сторонам виконувати обчислення над даними, не розкриваючи їх один одному. Це гарантує, що жодна сторона не має повного доступу до всієї інформації, що значно знижує ризик витоку даних або зловживань. Таким чином, навіть якщо один або декілька учасників системи будуть зламані або скомпрометовані, конфіденційність та цілісність транзакцій залишаться під захистом.

Загалом, предметом дослідження є створення ефективного та безпечного криптовалютного гаманця на основі технології багатосторонніх обчислень, який може забезпечити надійне та захищене використання в різних галузях, що працюють з криптовалютами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАВЛЕННЯ ЗАДАЧІ

1.1 Принципи та операції у мережі блокчейн

Блокчейн – це децентралізована та розподілена база даних, яка використовується для безпечного, незмінного та прозорого зберігання записів. Ця технологія функціонує на основі принципу послідовного об'єднання блоків даних у ланцюг. Кожен блок містить список транзакцій і пов'язаний з попереднім блоком за допомогою криптографічного хешу, що утворює ланцюг блоків, або блокчейн.

Децентралізація у блокчейні – це ідея про те, що контроль і повноваження щодо прийняття рішень у мережі розподіляються між її користувачами, а не контролюються однією особою, такою як уряд чи корпорація. Це може бути корисним у ситуаціях, коли людям необхідно координувати свої дії з незнайомцями або коли вони хочуть подбати про безпеку й цілісність своїх даних[1].

У децентралізованій блокчейн-мережі немає центрального органу чи посередника, який контролює потік даних чи транзакцій. Натомість транзакції перевіряються й записуються розподіленою мережею комп'ютерів, які працюють разом для підтримки цілісності мережі[1].

Безпека досягається за допомогою криптографічних методів, які захищають дані від несанкціонованого доступу і маніпуляцій.

У цьому контексті фундаментальне значення мають так звані криптографічні хеш-функції. Хешування – це процес, при якому алгоритм (хеш-функція) отримує вхідні дані будь-якого розміру і повертає результат (хеш), що містить передбачуваний та фіксований розмір (або довжину)[2]. Ця властивість хеш-функцій широко використовується в інформаційній безпеці, зокрема для захисту паролів. Коли ви реєструєте обліковий запис на веб-сайті, ваш пароль перетворюється за допомогою хеш-функції, у результаті чого

утворюється хеш-дайджест, який потім зберігається службою. Після входу в систему пароль, який ви вводите, проходить ту саму хеш-функцію, а отриманий хеш порівнюється зі збереженим, щоб підтвердити вашу особу.

Хеші в блокчейнах використовуються як унікальні ідентифікатори для блоків даних і створюють ланцюжок зв'язаних блоків (рисунок 1.1).

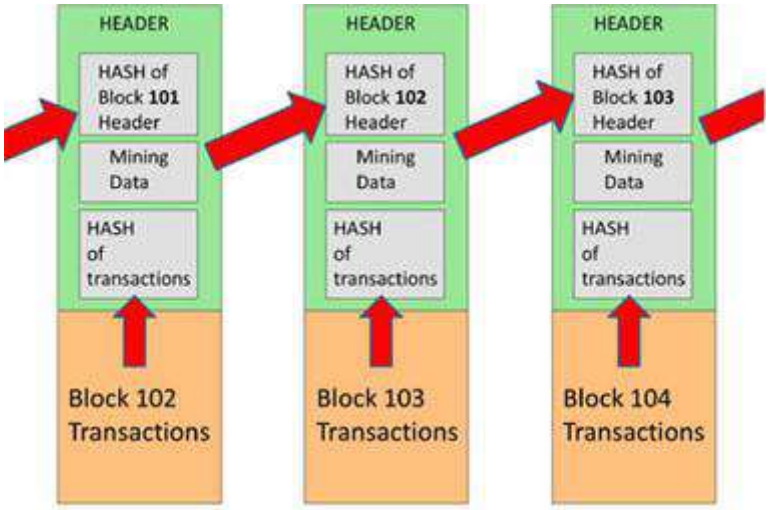


Рисунок 1.1 – Схематична структура блокчейну[7]

Для обробки всіх транзакцій у блоці використовується метод, відомий як алгоритм дерева Меркла(рисунок 1.2), і в цьому процесі застосовується хешування.

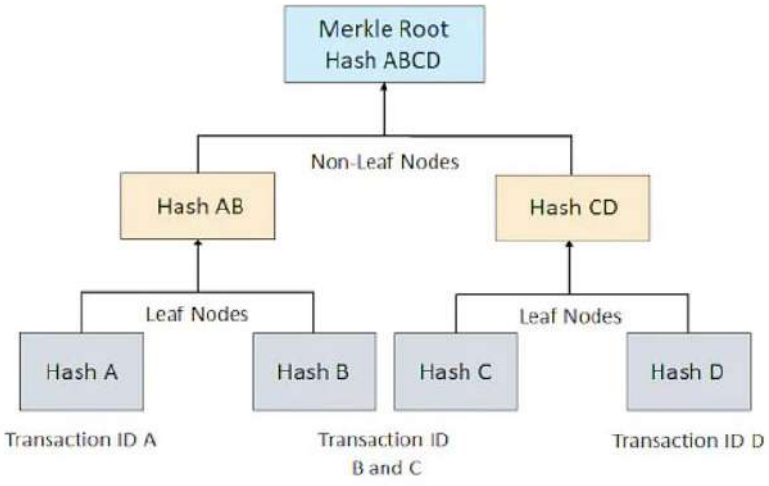


Рисунок 1.2 Дерево Меркла[8]

Кожна транзакція в блоці спочатку хешується індивідуально. Потім хеші кожної транзакції об’єднуються і хешуються разом. Якщо кількість транзакцій непарна, останній хеш дублюється і хешується сам із собою, щоб створити парну кількість хешів. Цей процес повторюється доки залишиться лише одне кореневий хеш, чи корінь Меркла.

Кожен хеш блоку генерується на основі даних в ньому та хешу попереднього блоку(рисунок 1.3).

Рисунок 1.3 – Схематична структура блоку[9]

Ця взаємозалежність гарантує безпеку та незмінність блокчейну. Хешування також використовується в алгоритмах консенсусу, які в свою чергу використовуються для перевірки транзакцій.

Алгоритм консенсусу – це механізм, який дозволяє користувачам чи машинам координувати свої дії у розподіленому середовищі. Він повинен гарантувати, що всі агенти в системі можуть домовитися про одне джерело правди, навіть якщо деякі агенти зазнають невдачі[1].

У блокчейні Bitcoin використовується алгоритм Proof of Work (PoW), процес виконання консенсусу Proof of Work (PoW) в основному полягає у тому, щоб мережа визначала, який блок буде доданий до ланцюжка блоків (blockchain). У спрощеному вигляді, учасники мережі, такі як майнери, змагаються за право знайти блок і додати його до ланцюжка. Для цього вони повинні вирішити складну математичну задачу, яка вимагає великої обчислювальної потужності. PoW, зазвичай базуються на хеш-функції SHA-256 (Secure Hash Algorithm 256-bit). Валідатори мережі повинні знайти створений хеш для нового блоку, який задовольняє певні умови (наприклад, має певну кількість нулів на початку)(рисунок 1.4).

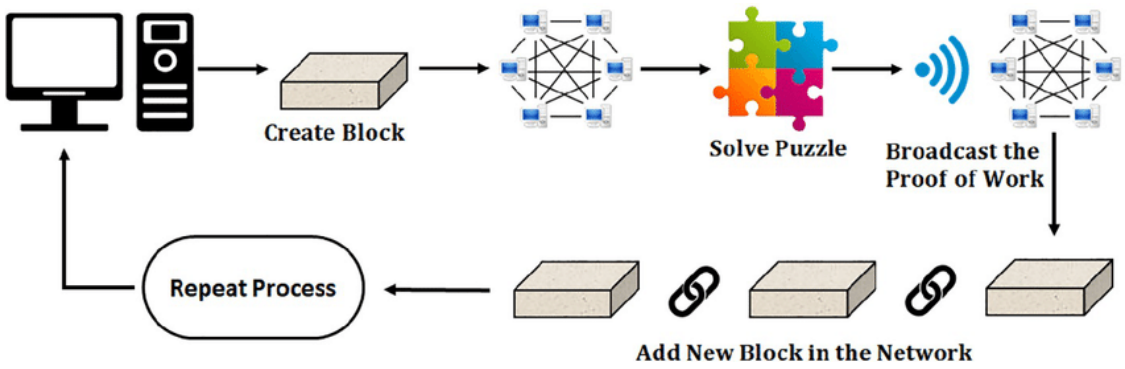


Рисунок 1.4 – Схематичне зображення Proof of Work (PoW)[10]

Саме завдяки консенсусу мережа захищена від атак, таких як подвійне витрачання та модифікація історії транзакцій. Він робить такі атаки вкрай важкими, оскільки будь-яка спроба зміни історії потребує повторного обчислення хешів для всіх попередніх блоків у ланцюжку, а також виграшу у конкуренції з іншими валідаторами для створення нових блоків. Чим більша обчислювальна потужність у мережі, тим важче здійснити подібні атаки, що робить мережу більш стійкою до зловживань

Для здійснення будь яких операцій у мережі, звичайні користувачі, повинні ініціювати крипто транзакції.

Крипто транзакція – це підтверджені підписом секції даних, які передаються в мережу блокчейна та збираються в блоки. Їх можна розглядати як переказ цифрової валюти з однієї крипто валютної адреси на іншу[3].

Для того, щоб відправити крипто валюту, кожному учаснику необхідно мати свій рахунок. Акаунт управляється парою ключів, де відкритий ключ це адреса, яку користувач вказує для отримання цифрових активів, а закритий контролює дії всередині облікового запису, тобто підписує транзакції. Кожен рахунок має тільки один закритий ключ[3].

Для підписання транзакції у блокчейн мережі використовується алгоритм ECDSA (Elliptic Curve Digital Signature Algorithm)(рисунок 1.5). Для підписання алгоритм використовує приватний ключ, що є випадковим числом, а публічний ключ генерується за допомогою множення приватного ключа на генераторну точку еліптичної кривої[4]. Перевірка включає використання публічного ключа для перевірки автентичності підпису. Процес полягає в обчисленні двох точок на еліптичній кривій і перевірці, чи збігаються ці точки з підписом та хешем повідомлення[4]. Ефективність та безпека ECDSA забезпечується властивостями еліптичних кривих, які дозволяють досягнути високого рівня стійкості до криптографічних атак при відносно невеликих обчислювальних витратах.

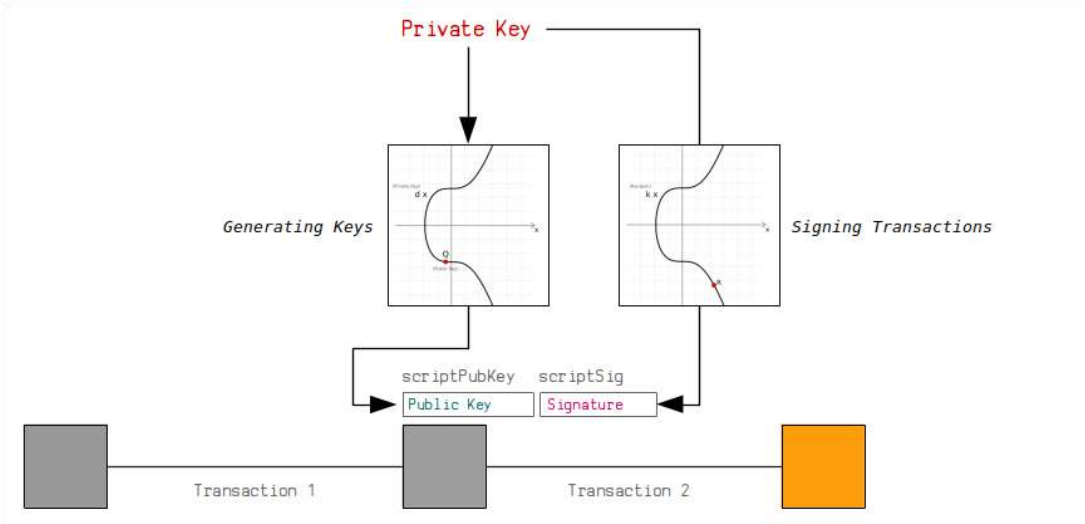


Рисунок 1.5 – Схематичне зображення роботи алгоритму ECDSA[11]

Весь процес обробки крипто валютних транзакцій (рисунок 1.6) можна поділити на три окремі етапи: створення, підписання, трансляцію та підтвердження (автентифікація та авторизація).

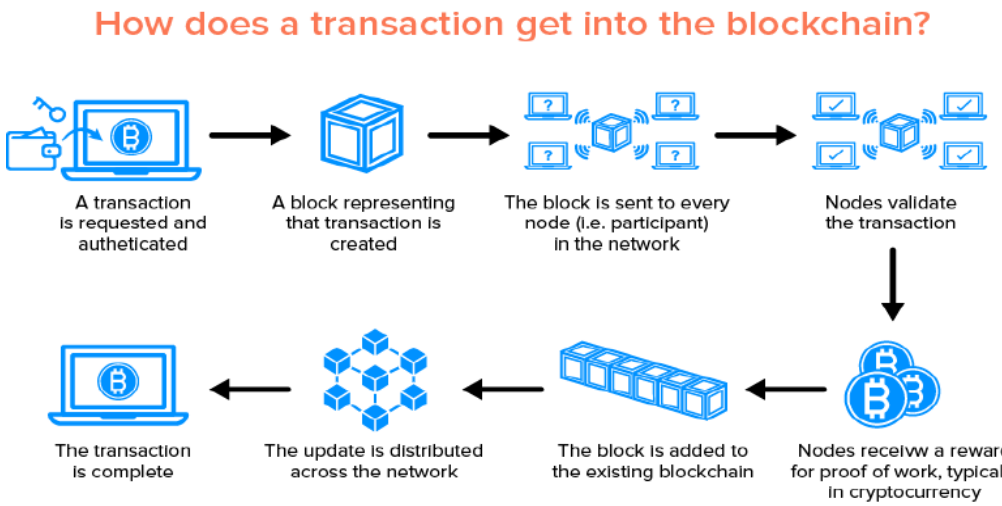


Рисунок 1.6 – Схематичне зображення взаємодії блокчейна та крипто транзакції[12]

1.2 Системи криптогаманців та їх різновиди

Криптовалютний гаманець — це пристрій, фізичний носій, програма або сервіс, який зберігає публічні та приватні ключі для проведення транзакцій з криптовалютою. Окрім основної функції зберігання ключів, криптовалютний гаманець часто пропонує додаткові можливості, такі як шифрування даних та підписання інформації. Це підписання може, наприклад, дозволити виконання смарт-контрактів, проведення транзакцій з криптовалютою, ідентифікацію або юридичне підписання документів.

Криптовалютний гаманець дає змогу зберігати, надсилати та отримувати криптовалюту. Деякі гаманці також підтримують мультивалютні операції, що дозволяє користувачам керувати кількома видами криптовалют одночасно. Важливою функцією криптогаманців є забезпечення безпеки активів

користувачів через різні методи аутентифікації, криптографічні протоколи та шифрування, які захищають від несанкціонованого доступу і кібератак.

Криптовалюта, така як Bitcoin, Ethereum або Litecoin, не зберігається у фізичному вигляді, як готівка, а знаходиться у цифровому форматі на блокчейні – загальнодоступному реєстрі транзакцій[6].

Криптогаманець необхідний для забезпечення безпеки та управління криптовалютними активами. У ньому зберігається приватний ключ (private key), який дає користувачеві можливість керувати своїми криптовалютними засобами. Приватний ключ використовується для підписання транзакцій та підтвердження їх відправлення.

Основні вимоги ,що ставляться до криптогаманців:

– криптогаманець повинен гарантувати надійний захист від несанкціонованого доступу. Це може включати шифрування приватних ключів, використання мультипідпису для підтвердження транзакцій, фізичні засоби захисту (як у випадку апаратних гаманців) та інші заходи безпеки[6];

– виборі способу зберігання криптовалюти необхідно оцінити рівень конфіденційності, який він забезпечує. Це означає, що гаманець не повинен зберігати особисту інформацію про користувача та його транзакції, а також не передавати цю інформацію третім сторонам без дозволу.[6];

– якщо ви плануєте зберігати не лише Біткоїни, а й іншу крипту, ваше сховище повинно підтримувати широкий спектр монет для зберігання криптовалюти. Особливо варто звернути увагу на вибір холодних гаманці[6];

– ще один вирішальний фактор, який необхідно враховувати при виборі типу криптогаманця, є можливості контролю своїх коштів. Такі рішення, як апаратні та програмні гаманці, дають користувачу повний контроль над приватними ключами, тоді як, наприклад, біржі зберігають ці ключі за клієнта[6].

За класифікацією криптовалютні гаманці поділяються на кілька типів, кожен з яких має свої особливості, переваги та недоліки. Ця класифікація стосується способу керування приватними ключами користувача.

Кастодіальний криптовалютний гаманець – це гаманець, в якому приватні ключі зберігаються третьою стороною. Це означає, що третя сторона буде зберігати та керувати приватними ключами від імені користувача. Іншими словами, у користувача не буде ні повного контролю над своїми коштами, ні можливості підписувати транзакції[14]. Важливо зазначити, що зберігання криптовалюти на кастодіальних платформах означає довіру до третьої сторони, що підвищує ризик втрати коштів у разі злому біржі або її банкрутства. Крім того, користувачі можуть розглядати гібридні рішення, такі як гаманці з мультипідписом, які вимагають кількох підписів для проведення транзакції, що забезпечує додатковий рівень безпеки.

Некастодіальний гаманець – це гаманець, в якому тільки власник володіє і контролює приватні ключі. Цей варіант найкраще підходить для користувачів, яким важливо мати повний контроль над своїми коштами. Оскільки посередників немає, користувач можете торгувати криптовалютою прямо зі своїх гаманців. Це хороший варіант для досвідчених трейдерів та інвесторів, які знають, як керувати та захищати свої приватні ключі і seed фрази[14].

З точки зору безпеки, апаратні гаманці вважаються найнадійнішими. Це фізичні пристрої, спеціально розроблені для зберігання приватних ключів – рядків даних, які забезпечують доступ до ваших коштів. Апаратні гаманці не потребують підключення до Інтернету, що робить їх невразливими до онлайн-атак. Однак, вони мають один суттєвий недолік: у разі втрати приватного ключа, користувач, швидше за все, втратить доступ до своїх коштів назавжди, оскільки немає можливості скидання пароля чи звернення до служби підтримки. Прикладами таких гаманців є Ledger, Trezor, CoolWallet, Digital Box, KeepKey та інші. Апаратні гаманці також часто використовуються для

довгострокового зберігання великих сум криптовалюти, оскільки вони забезпечують максимальний рівень безпеки.

Як випливає з назви, програмні гаманці є цифровими і зазвичай підключені до Інтернету. Вони можуть працювати на різних електронних пристроях. Серед програмних гаманців виділяють настільні, мобільні та вебгаманці. Наприклад, настільний гаманець – це програма, встановлена на комп'ютер, яка дає повний контроль над ключами та засобами.

Мобільні гаманці діють аналогічно, але встановлюються на мобільні пристрої, дозволяючи керувати засобами через додатки для смартфонів. Одним з таких додатків є Trust Wallet – децентралізований мобільний гаманець з відкритим кодом, що підтримує понад три мільйони криптовалют і близько 60 блокчейн-мереж. Trust Wallet також підтримує зберігання NFT, стейкінг та вбудований DeFi-браузер. Інші популярні програмні гаманці включають MetaMask, Exodus, і Mycelium.

Незважаючи на зручність, програмні гаманці завжди піддаються ризику зломів або проникнення шкідливих програм. Для підвищення безпеки користувачам рекомендується використовувати антивірусні програми та оновлювати програмне забезпечення до останніх версій.

Також існують паперові гаманці – фізичні документи, на яких надруковані приватні та публічні ключі. Вони не підключені до Інтернету, що робить їх невразливими до хакерських атак, але їх легко втратити або пошкодити.

Обираючи тип криптогаманця, користувачам слід враховувати свої потреби у безпеці, зручності та доступності, а також потенційні ризики, пов’язані з кожним типом гаманця.

1.3 Дослідження технології Multi-Party Computation

Багатосторонні обчислення (Multi-Party Computation ,MPC) - набір методів, за допомогою яких декілька учасників можуть спільно вирішувати обчислювальні задачі, не розкриваючи свої вхідні дані один одному. У контексті MPC, кожен учасник має свої власні приватні дані, і мета полягає в тому, щоб вони могли обчислити якийсь результат або функцію, використовуючи ці дані разом, при цьому не розкриваючи їх один одному. Такий тип обчислень стає альтернативою кастодіальним криптогаманцям з мультипідписом, так як дає можливість розділити відповідальність зберігання приватних ключів між декількома сторонами.

MPC – це підгалузь криптографії, яка виникла у 1970-х роках, а реальне використання почалося в 1980-х роках. Але на відміну від традиційних криптографічних методів, які часто використовуються для захисту інформації від сторонніх осіб, MPC використовує криптографію для забезпечення конфіденційності даних між учасниками однієї системи[16].

Щоб зрозуміти, як працюють протоколи шифрування багатосторонніх обчислень, уявімо, що ви хочете проаналізувати поточні зарплати на різних посадах у галузі охорони здоров’я. Зарплати зазвичай є конфіденційною інформацією, тому важливо, щоб окремі особи та компанії не розголошували ці дані. Однак, вивчаючи середні зарплати в різних галузях, ви сподіваєтеся отримати уявлення про те, де галузь охорони здоров’я може спостерігати вищі темпи зростання в найближчі роки.

Для цього прикладу припустимо, що вас цікавлять зарплати дипломованих медсестер, які працюють у відділеннях інтенсивної терапії. Зарплата медсестри А становить 100 тисяч доларів. Використовуючи протоколи багатосторонніх обчислень, ця сума розподіляється на три випадково згенеровані частки: 40 тисяч, 35 тисяч і 25 тисяч доларів. Потім медсестра А

залишає собі одну з цих часток (40 тисяч доларів) і роздає інші дві медсестрам Б (35 тисяч доларів) і С (25 тисяч доларів).

Зарплати медсестер Б і С також розподіляються за цим протоколом. Після завершення обміну кожна медсестра має три частки: одну із зарплати медсестри А, одну із зарплати медсестри Б і одну із зарплати медсестри С.

Таким чином, усі три медсестри надали свою особисту інформацію, але жодна з них не може визначити точну зарплату будь-якої іншої медсестри. Конфіденційність даних зберігається.

Проте ці частки залишаються корисними. Коли всі частки додано разом і поділено на три, отримуємо точне середнє значення зарплат усіх трьох медсестер.

Таким чином, технологія МРС тепер застосовується для низки варіантів використання, таких як захист цифрових активів у МРС-гаманцях або збереження певної інформації в таємниці під час цифрових аукціонів.

Є три основні властивості багатосторонніх обчислень [9]:

– сторони, залучені до багатосторонніх обчислень, ініціюють обчислювальні завдання та виконують спільні обчислення за допомогою узгодженої безпечної багатосторонньої обчислювальної функції. Вихідні дані, отримані за допомогою алгоритму, правильні (як очікувалося);

– кожна сторона повинна гарантувати, що їхні секретні дані є незалежними та що жодні локальні дані не розкриваються під час обчислень;

– пропонується децентралізована обчислювальна модель із повною рівністю кожного учасника без привілеїв для будь-якого учасника чи третьої сторони;

Далі ми розглянемо основні протоколи МРС, які застосовуються для реалізації кастодіальних криптогаманців.

Алгоритм Дженнаро та Голдфедера наразі є одним із найкращих доступних алгоритмів МРС, і багато установ, які захищають свої приватні дані за допомогою МРС, використовують саме цей алгоритм. Однак, незважаючи на

його переваги, затримка зв'язку між спільними ресурсами MPC (пристроями, які зберігають ключові спільні ресурси) не досягає найвищого рівня ефективності. Це обумовлено тим, що користувачі повинні чекати до 9 раундів підпису для завершення транзакцій.

Крім того, алгоритм Дженнаро та Голдфедера не пропонує жодної гнучкості для установ, яким потрібно використовувати холодне зберігання. Це суттєво обмежує його застосування у сценаріях, де необхідно забезпечити високий рівень безпеки при зберіганні даних офлайн.

Інший відомий алгоритм багатосторонніх обчислень, запропонований Лінделлом та ін., забезпечує незначне зменшення кількості транзакцій, які повинні бути підписані, у порівнянні з алгоритмом Дженнаро та Голдфедера – до 8 раундів. Однак цей алгоритм все ще не досягає рівня операційної ефективності, необхідного для сучасних ринків. Як і алгоритм Дженнаро та Голдфедера, рішення Лінделла також не підтримує холодне зберігання, що обмежує його використання в певних сферах[17].

Алгоритм MPC Доернера та ін. демонструє значний прогрес, досягаючи порогу лише за 6 підписів. Проте, навіть цей рівень ефективності не відповідає потенціалу сучасних технологій. Більше того, подібно до попередніх двох алгоритмів, рішення Доернера також не надає можливості використання холодного зберігання разом із MPC[17].

Таким чином, хоча сучасні алгоритми багатосторонніх обчислень, такі як Дженнаро та Голдфедера, Лінделл та ін., і Доернер та ін., пропонують значні переваги у захисті даних, вони мають певні обмеження. Основними з них є недостатня ефективність у кількості раундів підпису та відсутність підтримки холодного зберігання. Це підкреслює необхідність подальших досліджень і розробок у сфері MPC для досягнення більш високого рівня операційної ефективності та гнучкості[17].

Далі ми розглянемо алгоритми, які застосовуються для розробки протоколів у MPC гаманцях.

Схема розподілу секрету Шаміра (Shamir's Secret Sharing) — це алгоритм, який був вперше запропонований у 1979 році відомим ізраїльським криптографом Аді Шаміром. Вона дозволяє розділити інформацію на багато часток, при цьому для відновлення оригінального секрету потрібно лише частину з цих часток(рисунок 1.7).

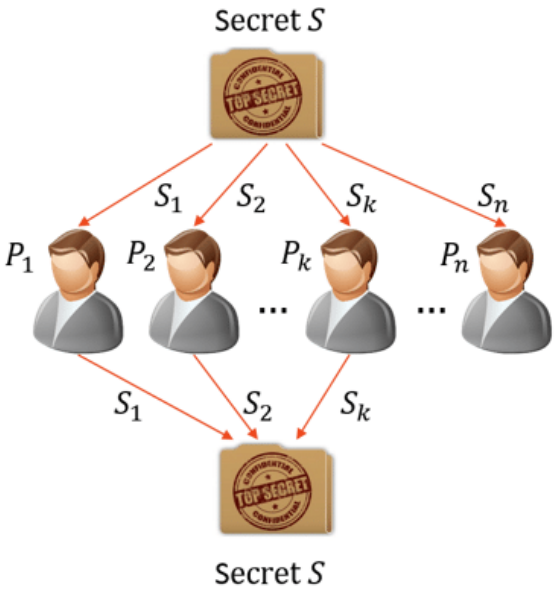


Рисунок 1.7 – Схематичне зображення роботи схеми розподілу секрету[13]

Це означає, що замість вимоги всіх часток для відновлення оригінального секрету, схема Шаміра вимагає мінімальну кількість часток — ця мінімальна кількість називається порогом. Для відновлення секрету необхідно досягти порогу. Якщо кількість часток менша за порогову, секрет не може бути відновлений, що робить схему розподілу секрету Шаміра захищеною від зловмисників з необмеженою обчислювальною потужністю; у криптографії це називається теоретичною безпекою інформації.

Порогова схема підпису (Threshold Signature Scheme, TSS) є криптографічним примітивом для розподіленого генерації ключів та підписання(рисунок 1.8). Використання TSS у клієнтах блокчейн, системах зберігання ключів є новою парадигмою, яка може надати численні переваги,

особливо в аспекті безпеки. У ширшому сенсі, TSS може вплинути на дизайн систем управління ключами (таких як криптогаманці) та прокласти шлях для підтримки нативних випадків використання в DeFi.

Втім, варто зазначити, що TSS все ще є новою технологією, тому ризики та обмеження також слід враховувати.

Порогові підписи вимагають підмножину сторін, уповноважених створювати підписи від імені групи. Повідомлення вважається підписаним лише в тому випадку, якщо поріг (t із n, де t — це поріг підписувачів, а n — кількість усіх підписувачів) підписувачів підписують це повідомлення.

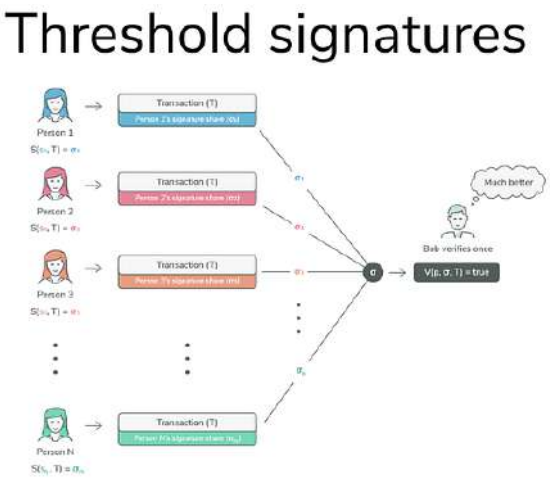


Рисунок 1.8 – Схематичне зображення роботи протоколу схеми порогового підпису[14]

Розподілена генерація ключів (Distributed Key Generation, DKG) — це криптографічний протокол, який дозволяє спільно генерувати криптографічні ключі багатьма учасниками, кожен з яких має частину секретного ключа)(рисунок 1.9). Розподіляючи процес генерації ключів між групою учасників, DKG гарантує, що жоден суб’єкт не матиме повного контролю, таким чином суттєво сприяючи безпеці та відмово стійкості розподілених систем. За своєю суттю DKG включає групу учасників, які об’єднуються для створення колективного закритого ключа. Процес організований таким чином, що кожен учасник вносить частину ключа, не розкриваючи свій конкретний

внесок іншим. Ця криптографічна магія полягає в її здатності поєднувати ці окремі внески для створення спільного приватного ключа без розкриття точних внесків.

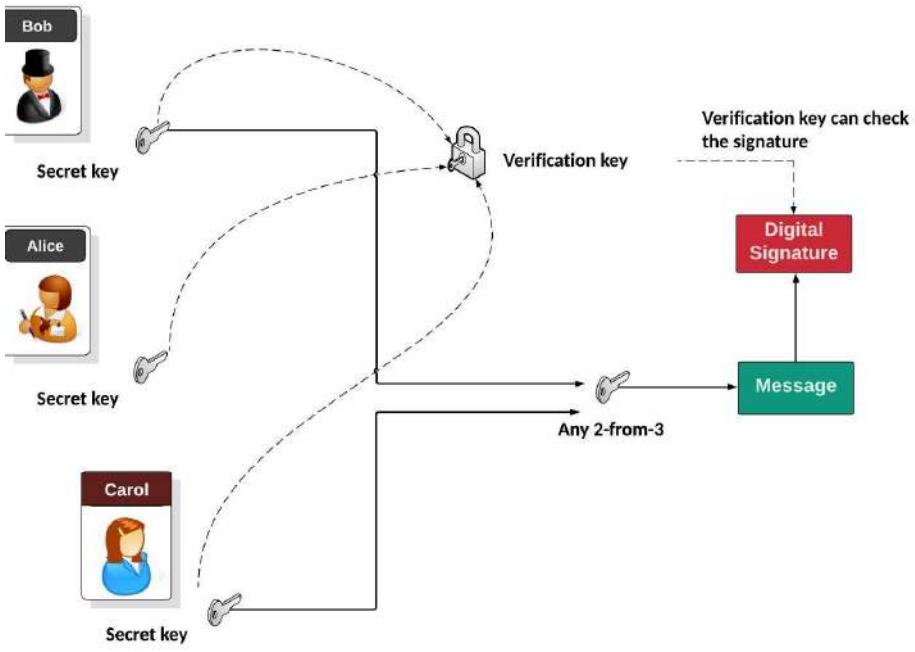


Рисунок 1.9 – Схематичне зображення розподіленої генерації ключа (DKG)[15]

Порівнюючи ці алгоритми можна виділити те що, SSS (Схема розподілу секрету Шаміра) забезпечує можливість зберігання приватного ключа (PK) у розподіленому вигляді, що підвищує безпеку. Проте, SSS має деякі недоліки, такі як необхідність "дилера" для генерації частин секрету та вимога для всіх сторін відновити повний PK для підпису[21].

Між тим, TSS (Протокол порогового підпису) пропонує певні переваги. TSS ґрунтується на чистій криптографії, що робить його незалежним від конкретних реалізацій блокчейн клієнтів. TSS усуває потребу в центральному "дилері", оскільки його роль розподілена, і повний PK ніколи не зберігається в одному місці. Крім того, TSS дозволяє розподілене підписання без відновлення частин секрету, покращуючи як безпеку, так і ефективність.

1.4 Постановка задачі

За результатами проведеного дослідження наявних технологій та проблем, пов'язаних з ними, а також з урахуванням зростаючої популярності криптовалют, постає задача розробки та створення системи конфеденційності та безпеки криптоактивів на основі технології багатосторонніх обчислень.

Об'єктом роботи була визначена розробка системи для надійного зберігання активів, а саме у вигляді приватних ключів до створених користувачем гаманців Ethereum мережі. Це є важливою метою, що передбачає створення рішення для забезпечення безпеки, приватності та зручності для користувачів.

Мета розробки системи конфіденційності та безпеки криптоактивів на сонові технології багатосторонніх обчислень для Ethereum мережі полягає в:

– забезпеченні високого рівня безпеки для збереження приватних ключів та процесу підписання транзакції, використовуючи передові методи криптографічного захисту;

– гарантуванні конфіденційності та приватності користувачів, захищаючи їх особисту інформацію від несанкціонованого доступу або витоку даних.

– використанні передових технологій, включаючи технологію Multi-Party Computation (MPC), для підвищення рівня безпеки та приватності зберігання активів.

2 ПРОЄКТУВАННЯ СИСТЕМИ КЕРУВАННЯ КРИПТОАКТИВАМИ НА ОСНОВІ ТЕХНОЛОГІЇ MULTI-PARTY COMPUTATION

2.1 Вибір стеку технологій для реалізації програмного забезпечення системи

Python - це мова програмування високого рівня, яка відома своєю простотою та читабельністю коду. Завдяки своїй простоті та виразності, Python став популярним вибором для початківців у програмуванні, а також для професійних розробників.

Відкритий інтерпретатор Python дозволяє швидко та ефективно створювати різноманітні програми, від веб-додатків до наукових обчислень.

Крім того, Python має велику спільноту користувачів та розробників, яка постійно розширює його екосистему бібліотек і інструментів для різних сфер застосування.

Django - це високорівневий фреймворк для розробки веб-додатків на Python, який надає потужні засоби для швидкої розробки та підтримки великих проектів. Django для реалізації центрального сервера (дилера) через його вбудовану систему аутентифікації та авторизації, через наявність потужного інстурменту[36].

Для безпечної взаємодії з базою даних використаємо Django ORM (Object Relational Mapping)[36] який надає можливість робити з базами даних через синтаксис мови програмування Python, а не чистим SQL(рисунок 2.1), це досягається через міграції, які є набором інструкцій для створення або зміни таблиць бази даних на основі змін у моделях.

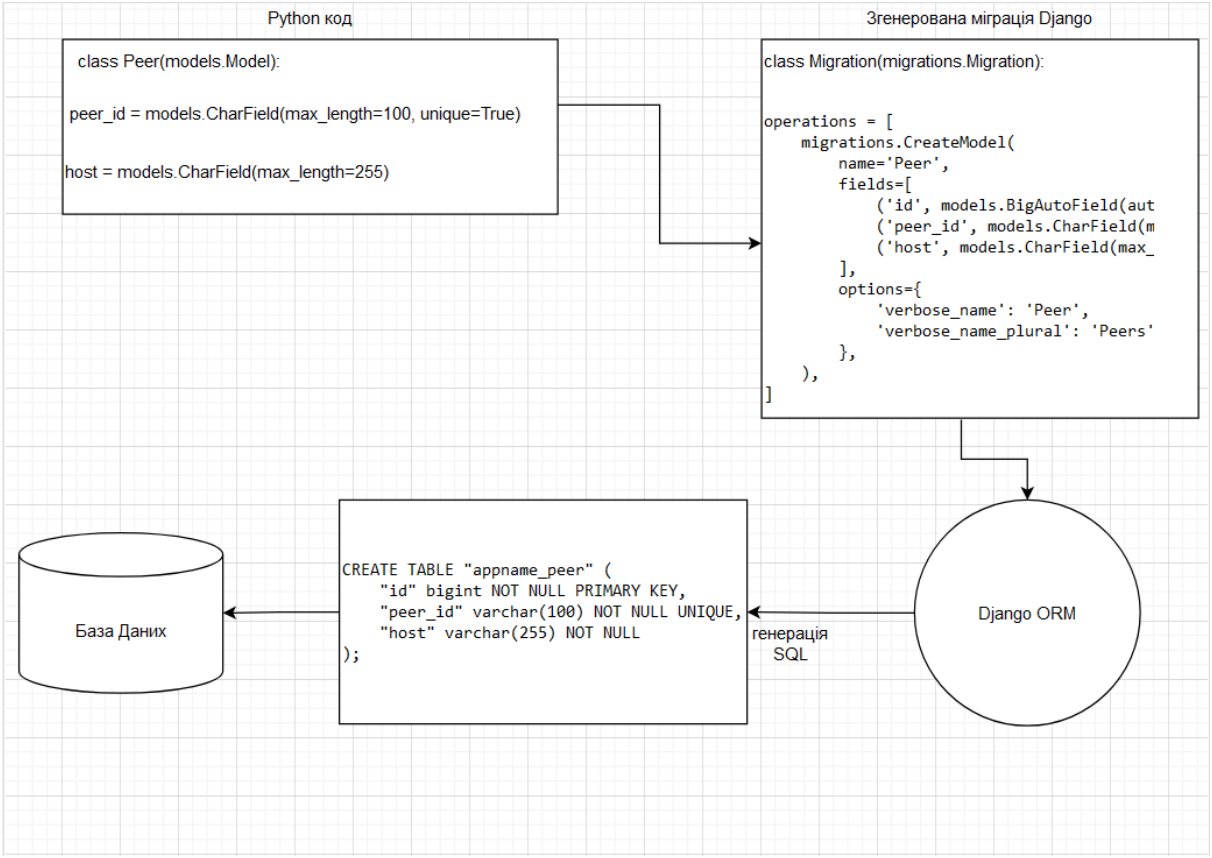


Рисунок 2.1 – Схематичне зображення роботи Django ORM

Для створення API інтерфейсу використаємо бібліотеку Django Rest Framework для взаємодії з кінцевими клієнтами.

PostgreSQL є потужною об'єктно-реляційною системою управління базами даних з відкритим вихідним кодом[37], яка надає надійність, масштабованість та розширюваність. Ми обираємо PostgreSQL для зберігання конфіденційної інформації про частки приватного ключа та інші дані, що вимагаються для роботи системи. Використання Django та PostgreSQL дозволить нам ефективно реалізувати центральний сервер (дилера) та розподілені вузли зберігання, забезпечуючи високий рівень безпеки, зручність у розробці та підтримці системи.

Для здійснення криптографічних операцій ми можемо скористатися готовими бібліотеками Python. Однією з популярних бібліотек є pycryptodome, яка містить реалізацію різних алгоритмів криптографії. Бібліотека дозволяє зручним та швидким способом розбити приватний ключ на частки та у

подальшому комбінувати часткові підписи для відновлення підписаної транзакції.

Для реалізації створення порогового підпису ми можемо скористатися бібліотекою py_ecc, яка містить інструменти для роботи з еліптичною кривою, що використовується в схемах порогового підпису. За допомогою цієї бібліотеки ми зможемо створити підписи транзакцій для Ethereum мережі за допомогою порогової схеми підпису.

Застосування готових бібліотек дозволить нам ефективно реалізувати функціональність секретного розподілу приватного ключа та схеми порогового підпису без необхідності власноручної імплементації цих складних алгоритмів, що забезпечить якість, надійність та безпеку реалізації.

Для реалізації комунікації між вузлами за допомогою WebSocket ми можемо використати бібліотеку Django Channels (рисунок 2.2), яка надає підтримку для асинхронних операцій у режимі реального часу для веб додатків.

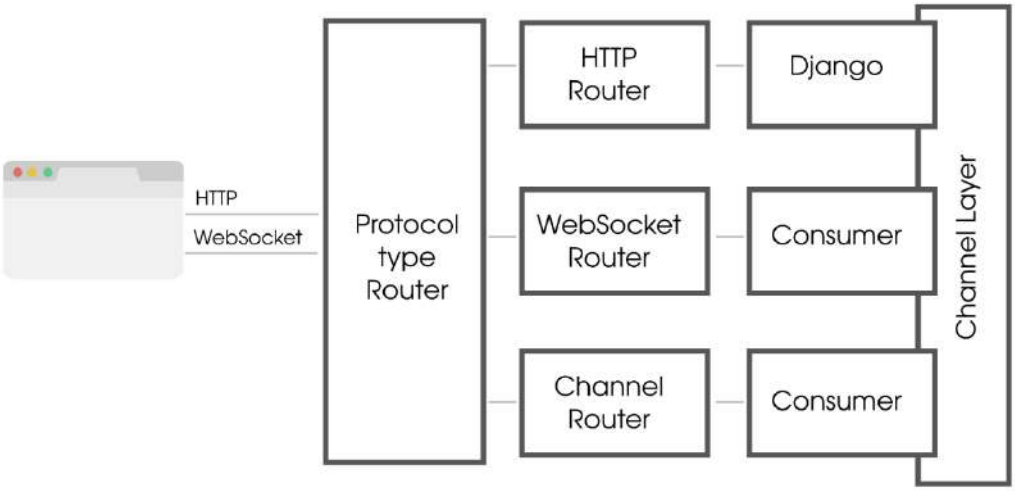


Рисунок 2.2 – Схема взаємодії Django Channels [38]

За допомогою Channels ми можемо створити захищені канали для комунікації між центральним сервером і вузлами. Ми можемо організувати поділ повідомлень на різні типи та додати обробники для цих повідомлень на

backend стороні, що дозволить нам реалізувати швидкі запити на часткові підписи транзакцій та безпечний обмін даними між сторонами.

Channels також надає можливість використовувати різні сховища для зберігання даних та сеансу, такі як Redis або база даних, для забезпечення масштабованості та надійності системи.

Redis — це потужна система кешування та сховище даних. Цей інстурмент можена використовувати для зберігання сеансів користувачів, кешування запитів до бази даних та багато іншого. Redis працює у пам’яті, що робить його дуже швидким. Він також підтримує різні структури даних, такі як ключ-значення, списки, множини та інші.

Celery — це інструмент для асинхронного виконання завдань у Python, використовуючи брокер повідомлень(у нашому випадку редіс). Він дозволяє запускати функції або процеси на основі розкладу або подій, що сприяє виконанню періодичних операцій та плануванню роботи за розкладом.

Docker – це платформа для контейнеризації додатків, що забезпечує середовище виконання, в якому програми можуть бути упаковані в контейнери разом з усіма необхідними залежностями, такими як операційна система, бібліотеки та інші компоненти.

Контейнери Docker (рисунок 2.3) надають легке та ізольоване середовище, яке дозволяє запускати та розгортати додатки на різних платформах із мінімальними проблемами сумісності [42].

Контейнери Docker можуть працювати на будь-якій платформі, що підтримує Docker, будь то локальний комп'ютер, сервер або хмарна інфраструктура. Це забезпечує високу переносимість додатків і знижує проблеми сумісності. Кожен контейнер Docker функціонує в своєму ізольованому середовищі, що гарантує відсутність взаємодії між додатками або з хост-системою. Це допомагає уникнути конфліктів залежностей та підвищує надійність і безпеку виконання додатків.

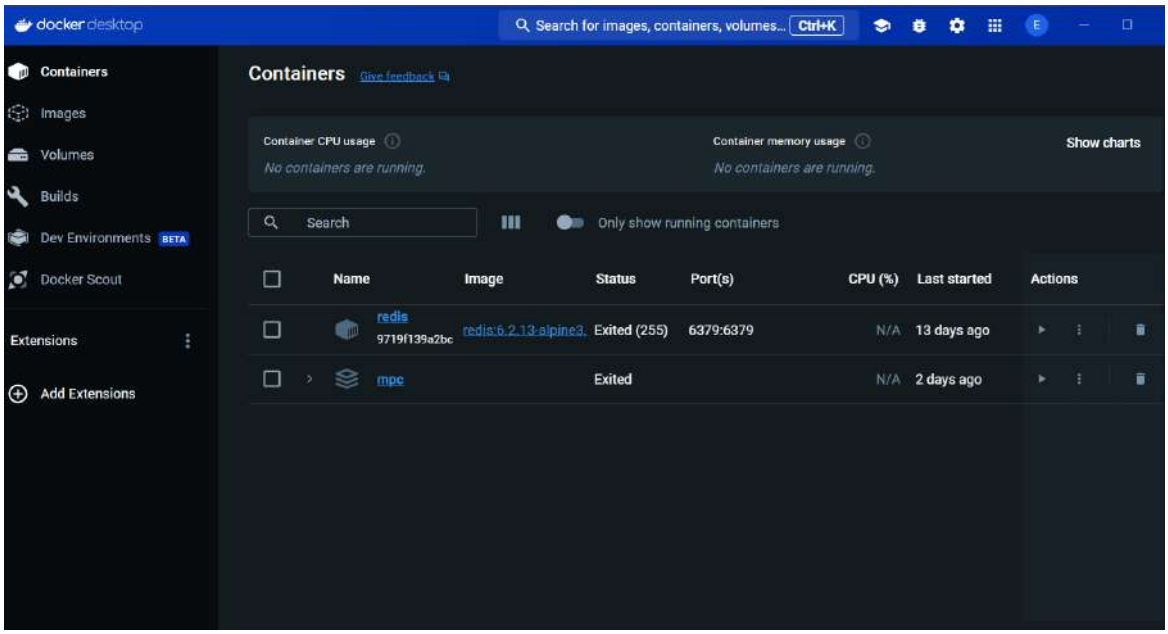


Рисунок 2.3 Головне меню Docker

Docker спрощує процес розробки та розгортання додатків. Завдяки Docker-образам, які містять всі необхідні компоненти програми, можна легко відтворювати середовище на різних етапах розробки, тестування та впровадження. Це забезпечує одноманітне та повторюване оточення для роботи над додатком.

Docker також підтримує масштабування додатків. Можна створювати кілька екземплярів контейнерів та управляти ними за допомогою оркестраторів, таких як Docker Swarm або Kubernetes. Це дозволяє ефективно керувати та масштабувати додаток залежно від навантаження. Docker оптимізує використання ресурсів, розділяючи та ізолюючи додатки в контейнерах, що дозволяє запускати кілька контейнерів на одному хості та ефективно використовувати обчислювальні ресурси, пам'ять і мережеві ресурси.

Загалом, Docker пропонує простий та ефективний спосіб упаковки, розгортання та управління додатками, роблячи його незамінним інструментом у сфері розробки та впровадження програмного забезпечення. Користуючись цим інструментом ми зможемо створити окремі образи для кожного вузла, та протестувати локально виконання протоколу багатосторонніх обчислень.

2.2 Проєктування протоколу багатосторонніх обчислень для системи керування криптоактивів

Протокол багатосторонніх обчислень для системи конфеденційності та безпеки криптоактивів є основою на якій будується принцип контролю приватних ключів користувачів та здійснення усіх необхідних операцій, таких як підписання транзакції та її валідація. При проєктуванні таких протоколів необхідно запобігти усіх наявних мінусів попередніх протоколів.

Починаючи аналіз з алгоритму секретного поділу Шаміра, можна виділити, що проблемою яка виникає є одна точка відмови , а саме центральний сервер який виконує розподіл і збирання секретного ключа, і цей фактор потрібно врахувати при проектуванні протоколу для системи. Проте перевагою є можливість розділити приватний ключ на частинки з порогом відновлення.

Хоча доведено, що протокол порогового підпису є безпечним, його реалізація також може вимагати деяких додаткових заходів для захисту секретних даних. Тому використовуючи запропоновані варіанти із статті про пороговий протокол ECDSA[24], розширимо протокол додатковими механізмами для досягнення активної безпеки. Розглянемо кожен з них.

Перший механізм це оновлення секретного ключ. Як було зазначено раніше, у пороговій криптографії секрет розподіляється між n сторонами, і щоб використовувати секретний ключ, t+1 сторони повинні об’єднати свої частки. З точки зору суперництва, зловмисник повинен порушити t+1 сторін, щоб атакувати таку схему. Якщо припустити, що зловмисник порушує сторони одну за одною, механізм оновлення ключа пом’якшує таку атаку. Точніше, під час оновлення ключа секретні спільні ресурси оновлюються через певний період. Таким чином, якщо зловмиснику вдається порушити відсоток сторін протягом цього певного періоду та порушує решту в наступному періоді, вони не можуть розкрити таємницю, оскільки акції поновлюються в кінці першого періоду.

Єдиний спосіб для зловмисника досягти успіху - атакувати t + 1 сторін одночасно, що набагато складніше.

Другий механізм це докази з нульовим знанням. У такому випадку протокол використовує «оптимістичний» підхід і повинен використовувати мінімальну кількість доказів з нульовим знанням у протоколі для отримання кращих результатів продуктивності. Одним із типів доказів нульового знання, є докази діапазону в протоколі MtA (Multiplicative to Additive), які використовуються для підтвердження того, що секретні частки як Аліси, так і Боба менші за певне число, але через ці перевірки протокол стає відносно дорогим у продуктивності, та впливає на швидкодію програми.

У статті [24] стверджується, що видалення цих протоколів із протоколу може призвести до витоку інформації про секретний ключ, який занадто обмежений і не впливає на безпеку протоколу. Таким чином, ці докази можуть бути включені в реалізацію або виключені з неї щодо компромісу між безпекою та продуктивністю.

Для знаходження компромісу між ефективністю та безпекою в системі, для покращення безпеки системи використаємо принцип з інтервальним оновленням спільних поділених частин та відмовимося від доказів нульової довіри для покращення продуктивності.

Протокол повинен відповідати властивостям багатосторонніх обчислень, а саме конфіденційності, де кожна сторона гарантує, що жодна зі сторін не знає загального секрету, але може гарантувати правильне розподілене обчислення над даними.

Це можна досягти за допомогою розподіленої генерації ключа для ECDSA (Elliptic Curve Digital Signature Algorithm) в контексті Ethereum. Такий підхід забезпечує, що приватний ключ ніколи не розкривається жодній стороні, а підписання транзакцій виконується спільно, що підвищує безпеку та конфіденційність. Даний алгоритм нам потрібно буде розширити інтервальним

оновленням частин секрету, без втрати оригінального секрету, що є важливим аспектом забезпечення довготривалої безпеки в розподілених системах.

Опишемо процес розподіленої генерації ключа математично, для подальшої розробки блок схеми.

Процес складається з наступних етапів:

– ініціалізація;

– генерація поліному;

– обрахунок частинок;

– надсилання частин сторонам;

– оубчислення власної частини стороною секрету.

Ініціалізація розпочинається з ініціалізації однієї зі сторін протоколу генерації , і тоді кожен учасник генерує свою частину секретного ключа s_i випадковим чином, але так щоб згенероване значення відповідало вимозі , що число позитивне та менше за встановлений порядок поля q , де q це кінцева точка обраної еліптичної кривої для системи. У нашому випадку використовується крива secp256k1, що задається рівнянням (2.1)

$$y^2 = x^3 + ax + b \tag{2.1}$$

де a, b – параметри задаються стандартно , для кривої secp256k1 дорівнюють 0 та 7 відповідно.

Далі відбувається генерація поліному кожною стороною з F(x) ступеня t з випадковими коефіцієнтами a_i, поліном задається наступною формулою (2.2).

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_t x^t \tag{2.2}$$

де t – поріг відновлення; n_t коефіцієнт який дорівнює s_i (згенерований стороною секрет).

Після того як поліном згенеровано, сторона обчислює частки ключа для кожної іншої сторони передаючи до поліному F(x) порядковий номер вузла у якості параметру x. Згенерований перелік частин власного секретного ключа, сторона розсилає до відповідного учасника через безпечний канал зв’язку.

Після того як усі частинки надіслані та отримані сторонами , кожна зі сторін може безпечно обрахувати спільний секретний ключ сумуючи отримані частинки за модулем порядку поля q, цього вистачить для отримання публічного ключа, яким потім можна буде перевірити на валідність транзакцію.

Зобразимо алгоритм у вигляді схеми(рисунок 2.4), яка має наступний вигляд:

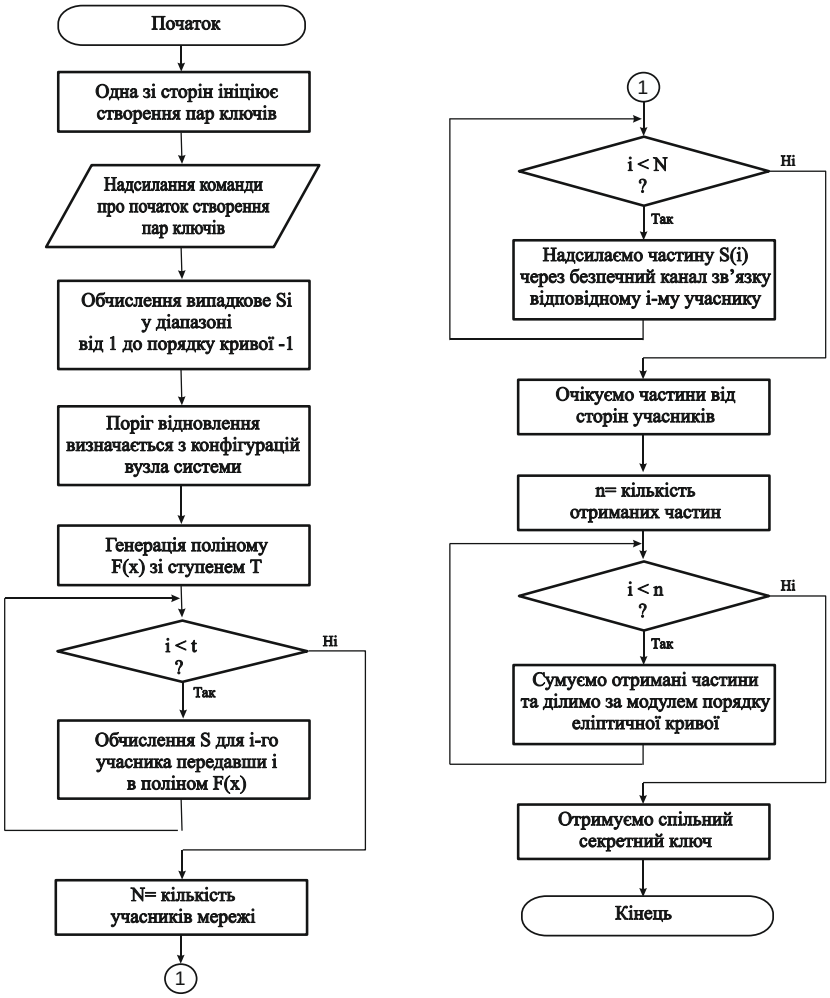


Рисунок 2.4 – Схема розподіленої генерації спільного секретного ключа

Тепер нам потрібно розробити механізм інтервального оновлення часток секретного ключа , спираючись на математичні операції які застосовувались для їх генерації.

Ознайомившись з науковою статтею про поділ секрету Шаміра[n, 16] можна виділити наступні кроки оновлення часток:

– ініціалізація;

– генерація та розподіл;

– обчислення нових часток

Оновлення ключа розпочинається з того , що один із учасників повідомляє інших про початок раунду оновлення частинок.

Тоді кожен учасник повинен згенерувати випадкове число n_i, відповідно до того яке він генерував для власного секретного ключа. Це випадкове число буде використано для зміни його частки секретного ключа. Згенеровані частки обмінюються між учасниками через безпечний канал зв'язку.

Після обміну нових часток, учасники повинні обрахувати свій новий спільний ключ через формулу (2.3).

$$s_i^{`} = s_i + \sum_{j \neq i} n_i - n_i \qquad (2.3)$$

де n_i— це поточна частка учасника j; $s_i^{`}$— це оновлена частка.

Сума n, що надходять від інших учасників, додається, а своє власне n віднімається, щоб зберегти загальну суму секретного ключа незмінною.

Зобразимо алгоритм у вигляді схеми (рисунок 2.5).

Таким чином ми можемо досягти регулярного оновлення часток секретного ключа, зменшуючи ризики компрометації і підтримуючи високий рівень безпеки в розподілених системах.

Шляхом розробки та впровадження протоколу багатосторонніх обчислень з ключами, розподіленими та періодично обновлюваними, ми ефективно покращуємо рівень безпеки. У порівнянні з традиційними методами зберігання

ключів у єдиному місці, наш підхід забезпечує значно вищий ступінь захисту від потенційних атак.

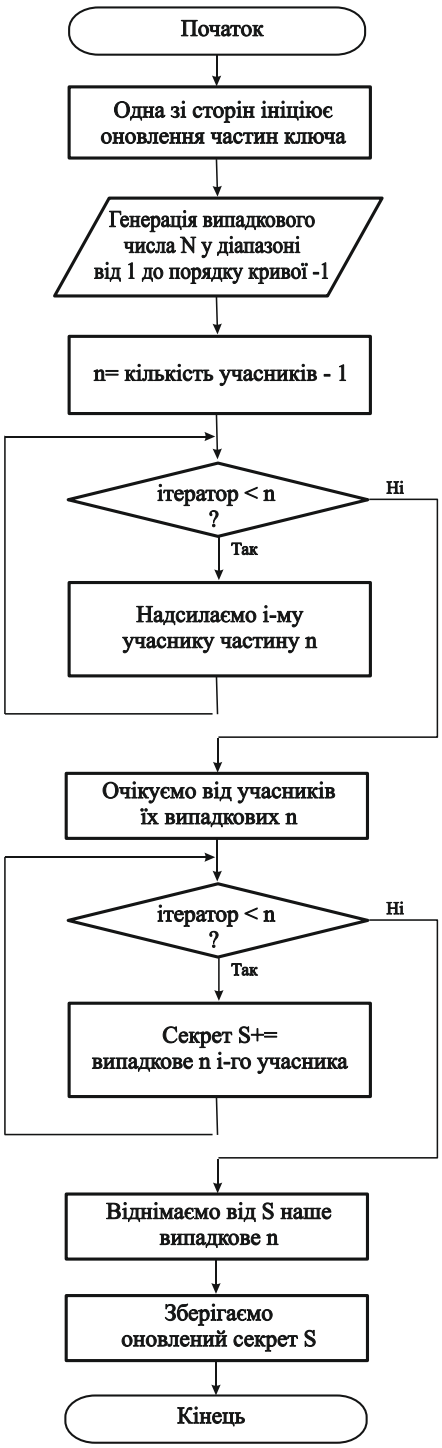


Рисунок 2.5 – Схема оновлення частинок приватного ключа учасника протоколу

Замість того, щоб мати один централізований пункт доступу, який може стати мішенню для зловмисників, ми розподіляємо ключі між декількома довіреними елементами системи. Кожен ключ періодично оновлюється або змінюється, ускладнюючи можливість несанкціонованого доступу. Цей підхід не лише робить процес взлому системи значно складнішим завданням, але й активно захищає систему від потенційних атак, надаючи більшу впевненість у безпеці та надійності обчислень.

2.3 Проєктування архітектури системи керування криптоактивами

У цьому розділі ми розглянемо архітектуру системи, що здійснює генерацію та оновлення ключів, а також підписання транзакцій у контексті порогової криптографії. Система буде реалізована як однорангова (peer-to-peer, P2P) мережа для забезпечення децентралізації та підвищеної безпеки. Окремо буде описана архітектура кожного вузла у мережі та конфігурації, необхідні для його функціонування.

Комп'ютерні мережі типу peer-to-peer (P2P)(рисунок 2.6) засновані на принципі рівноправності учасників і характеризуються тим, що їх елементи можуть зв'язуватися між собою[17].

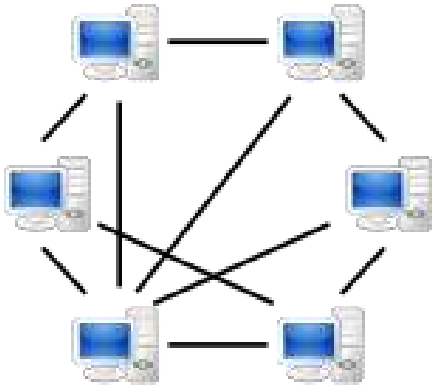


Рисунок 2.6 – Топологія мережі типу peer-to-peer[17]

На відміну від традиційної архітектури(рисунок 2.7), коли лише окрема категорія учасників, яка називається серверами, може надавати певні сервіси іншим.

Рисунок 2.7 Топологія мережі типу клієнт-сервер[17]

Так як система обчислень є багатосторонньою, і ми хочемо позбавитись єдиної точки відмови, нашим вибором буде мережа типу Peer-to-Peer , де кожен вузол виконує роль учасника в процесі генерації та оновлення ключів, а також підписання транзакцій.

Для забезпечення комунікації між вузлами у системі ми використаємо технологію веб-сокетного з’єднання.

WebSocket — це протокол, що призначений для обміну інформацією між браузером та вебсервером в режимі реального часу. Він забезпечує двонаправлений повнодуплексний канал зв'язку через один TCP-сокет. WebSocket спроектовано для втілення у веббраузерах та вебсерверах, але може також використовуватись будь-яким клієнт-серверним застосунком[23].

Використання WebSocket дозволяє нам побудувати надійний та ефективний механізм обміну даними між вузлами. При використанні WebSocket ми можемо реалізувати різні типи повідомлень для обміну даними між центральним сервером і вузлами, такі як запити на підпис транзакцій, генерацію приватних ключів, а також статусні повідомлення для синхронізації даних між вузлами.

Крім того, WebSocket дозволяє нам побудувати розширені механізми керування підключеннями, автентифікацію та авторизацію, що може бути корисним для забезпечення безпеки та надійності комунікації між центральним вузлами.

Для відносно невеликої нагруженості , але ефективної роботи системи, було вирішено розгорнути 3 вузли, у кожного з вузлів буде йти стандартна конфігурація основних параметрів, таких як поріг обрахунку поліному, та еліптична крива яка буде використовуватися.

Важливим пунктом у організації такого з’єднання , є встановлення безпечного та конфеденційного каналу зв’язку між вузлами. У реальності використовують протоколи. Тому для нашої системи теж є необхідністю реалізація аналогів таких протоколів. Загальну схему встановлення безпечного каналу зв’язку між двома вузлами можемо побачити на рисунку 2.8.

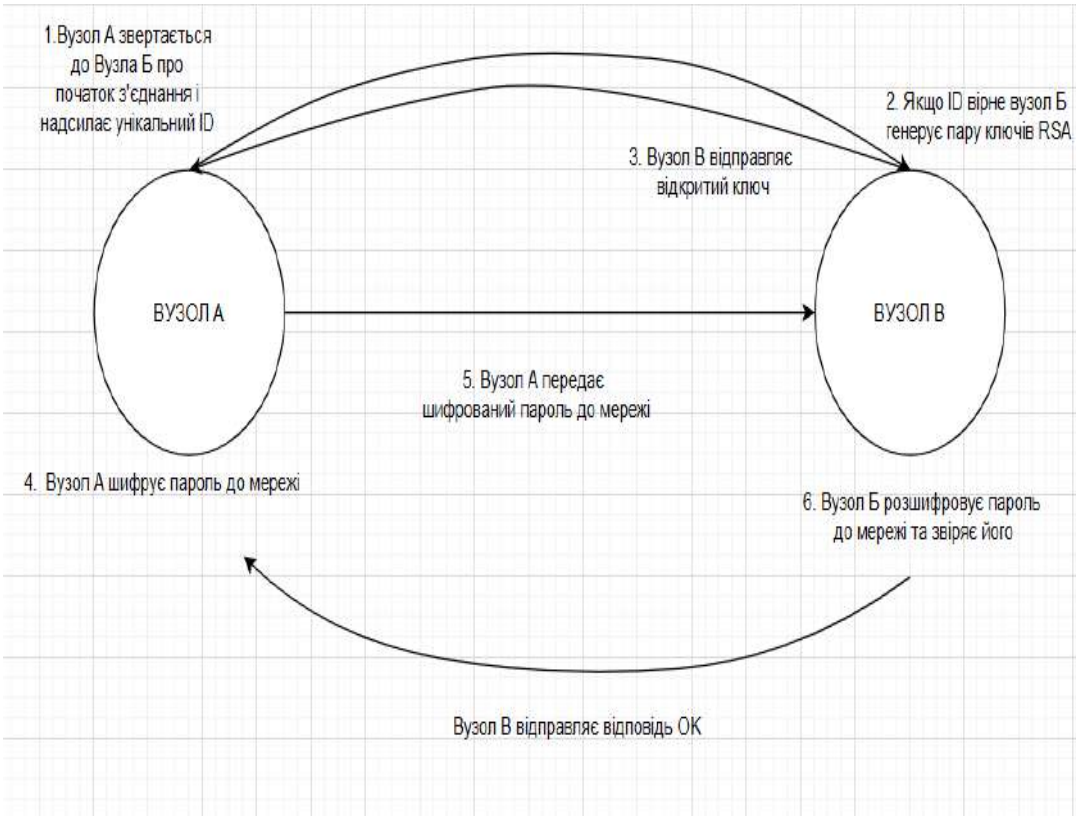


Рисунок 2.8 Загальна схема встановлення безпечного зв’язку

Схема є досить простою у розумінні, та описує процес встановлення безпечного з'єднання між двома вузлами (A і B) у одноранговій мережі за допомогою асиметричної криптографії RSA[27]. Давайте детально розглянемо кожен крок цього процесу. Та складається з наступних кроків:

– початок з’єднання;
– перевірка ID;
– передача відкритого ключа;
– шифрування пароля;
– передача шифрованого пароля;
– розшифрування та верифікація.

Процес встановлення безпечного зв’язку між двома вузлами (A і B) у одноранговій мережі починається з того, що Вузол А звертається до Вузла В з проханням про початок з’єднання. Вузол А надсилає унікальний ідентифікатор (ID), який використовується для автентифікації, його встановлює адміністратор системи при розгортанні. Після отримання запиту, Вузол В перевіряє надісланий ID. Якщо ID вірний, Вузол В генерує пару ключів RSA, яка складається з відкритого і приватного ключів.

Далі, Вузол В відправляє свій відкритий ключ до Вузла А. Цей відкритий ключ буде використовуватись для шифрування повідомлень з паролем до мережі від Вузла А до Вузла В. Вузол А отримує відкритий ключ і використовує його для шифрування пароля. Шифрований пароль передається через мережу до Вузла В.

Коли Вузол В отримує шифрований пароль, він розшифровує його за допомогою свого приватного ключа RSA. Після успішного розшифрування Вузол В перевіряє правильність отриманого пароля. Якщо пароль вірний, Вузол В надсилає відповідь "OK" до Вузла А, що підтверджує успішне встановлення безпечного з’єднання. Тоді обидва вузли можуть безпечно шифрувати та дешифрувати дані використовуючи пароль мережі, за допомогою алгоритму симетричного шифрування AES.

Ця схема забезпечує конфіденційність, оскільки відкрите ключ використовується для шифрування даних, які можуть бути розшифровані лише за допомогою приватного ключа. Крім того, використання унікального ID на початку з'єднання допомагає верифікувати учасників, а зашифровані повідомлення гарантують цілісність даних під час передачі, захищаючи від несанкціонованого доступу або змін.

У кінцевому варіанті архітектура системи матиме вигляд однорангової мережі зі встановленим безпечним з’єднанням. Також потрібно передбачити механізм взаємодії з кінцевим користувачем або іншим мікросервісом. Для організації такої роботи, один з вузлів повинен підтримувати протокол HTTP застосувавши архітектурний стиль проєктування REST API

REST API взаємодіє через HTTP запити, виконуючи основні операції для роботи з ресурсами, такі як створення, оновлення, читання та видалення записів. Існують чотири основні методи, які визначають, яку дію необхідно виконати з ресурсом:

– POST метод використовується для створення нового ресурсу;

– GET метод застосовується для читання або отримання інформації про ресурс;

– PUT метод призначений для оновлення існуючого ресурсу;

– DELETE метод служить для видалення ресурсу.

Таким чином для кожної операції генерації адреси або підписання транзакції у системі вузла буде реалізований відповідна кінцева точка.

Після проведеної роботи, схема системи матиме наступний вигляд(рисунок 2.9):

Після проектування архітектури системи, перейдемо до проектування вузла системи. Кожен вузол у системі має свою архітектуру програмного забезпечення, яка складається з декількох основних компонентів:

– мережевий компонент;

– компонент управління ключами;

– компонент підписання транзакції;

– компонент конфігурації вузла програмного забезпечення.

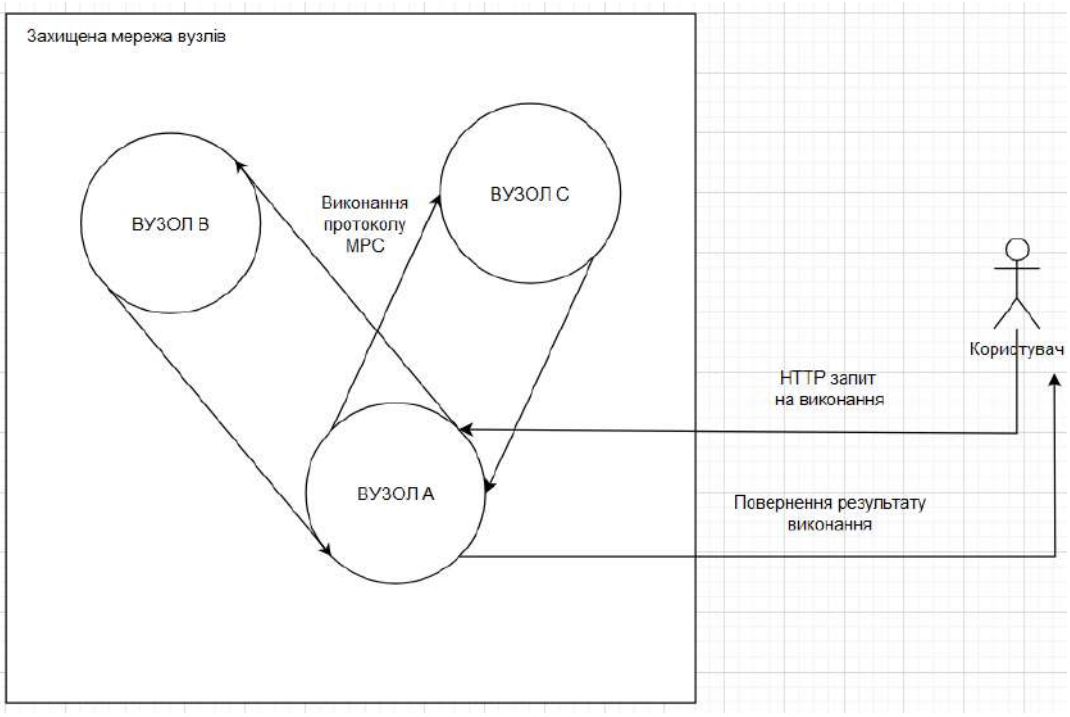


Рисунок 2.9 Схема взаємодії компонентів системи керування криптоактивами

Мережевий компонент відповідає за зв'язок з іншими вузлами, забезпечуючи прийом і відправку повідомлень, таких як запити на підписання або частки ключів. Інтерфейс для ініціалізації з'єднань з вузлами дозволяє вузлу ефективно взаємодіяти в одноранговій мережі. Також цей компонент відповідає за API взаємодію з кінцевим клієнтом.

Компонент управління ключами займається генерацією поліномів та розподілом часток ключів. Цей компонент також шифрує частки ключів перед їх передачею іншим вузлам та розшифровує отримані частки. Це забезпечує безпеку ключової інформації під час її обміну між вузлами мережі. Компонент також включає в себе оновлення ключа та своєчасне виконання протоколу багатосторонніх обрахунків для оновлення частинок секретного ключа відповідно алгоритму.

Компонент підписання відповідає за генерацію часток підпису. Він також виконує верифікацію часток підпису, отриманих від інших вузлів, і комбінує їх для створення загального підпису. Це критично важливий процес, який забезпечує цілісність та авторитетність транзакцій у системі.

Компонент конфігурації налаштовує параметри безпеки, такі як шифрування та генерація ключів, а також параметри мережевого з'єднання, включаючи IP-адреси та порти. Параметри порогової криптографії, зокрема кількість учасників, інтервал оновлення частинок ключа та поріг, також налаштовуються через цей компонент. Це дозволяє вузлу функціонувати в рамках визначених безпекових політик та забезпечує гнучкість системи.

Загальну схему взаємодії компонентів можна побачити на рисунку 2.10.

Рисунок 2.10 Схема взаємодії компонентів архітектури вузла програмного забезпечення системи

Локальне сховище використовується для збереження часток ключів, підписів та системних логів. Шифрування даних забезпечує конфіденційність,

навіть у випадку компрометації фізичного сховища. Сховище буде складатися з таких сутностей:

– конфігурація вузла;
– частки ключів у шифрованому стані;
– часткові підписи транзакцій;
– журнал подій.

Схема локального сховища вузла прийматиме наступний вигляд , що зображено на рисунку 2.11.

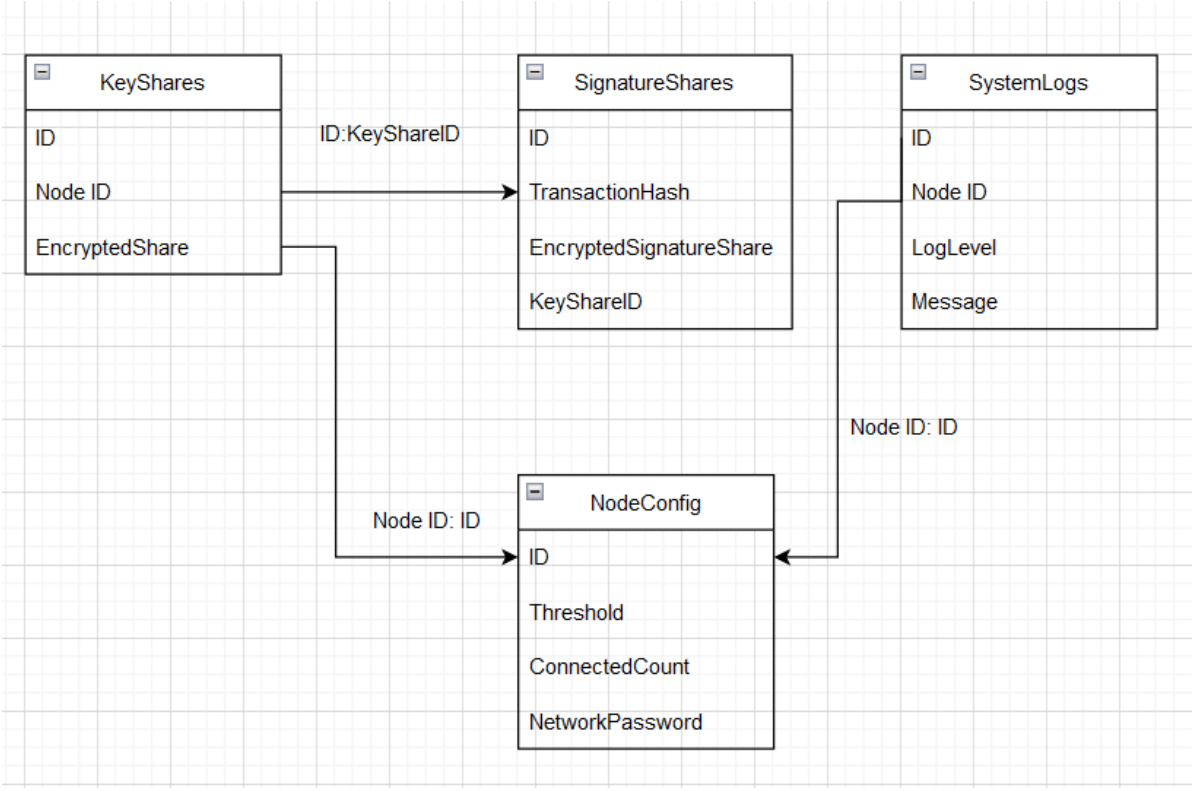


Рисунок 2.11 Схема локального сховища вузла програмного забезпечення системи

Конфігурація вузла буде використовуватися для подальшого масштабування системи.

Кожна частка ключа зберігається з унікальним ідентифікатором. Вона пов'язана з конкретним вузлом, який її зберігає. Частки ключів шифруються перед збереженням для забезпечення конфіденційності.

Кожна частка підпису також має унікальний ідентифікатор і пов'язана з конкретною транзакцією. Як і частки ключів, частки підписів шифруються перед збереженням. Вузол зберігає частки підписів, які потім комбінуються для створення загального підпису.

Журнал подій міститиме записи про всі важливі події, що відбуваються в середині вузла, такі як встановлення з'єднань, обмін повідомленнями і помилки. Ці записи допомагають у моніторингу та діагностиці системи.

2.4 Висновки

Використовуючи проаналізовані в попередніх розділах характеристики та стандарти протоколів багатосторонніх обчислень ми розробляємо протокол для системи конфіденційності та безпеки крипто активів, який буде враховувати усі наявні покращення та недоліки попередніх протоколів. Для покращення протоколу, ми використали принцип інтервального оновлення спільного секрету для ускладнення можливості взлому та надання активного захисту системі.

Після протоколу, нам потрібно було спроектувати архітектуру системи та їх компонентів. Розроблена архітектура одного вузла показує, як основні компоненти взаємодіють між собою, забезпечуючи надійну та безпечну роботу системи. Спроектували схему встановлення безпечного з’єднання вузлів. Описали модель локального сховища, яке забезпечує безпечне зберігання часток ключів, часток підписів та системних логів.

Також було підібрано оптимальний набір інстурментів для розробки та тестування. Було обрано мову програмування Python та веб фреймоврки для розробки веб сокетного з’єднання та API інтерфейсу. Для полегшення тестування та розгортання системи ми використаємо контейнеризацію за допомогою утиліти Docker.

3 РОЗРОБКА СИСТЕМИ КЕРУВАННЯ КРИПТОАКТИВАМИ НА ОСНОВІ ТЕХНОЛОГІЇ БАГАТОСТОРОННІХ ОБЧИСЛЕНЬ

3.1 Реалізація протоколу багатосторонніх обчислень керування криптоактивами

Усю розробку протоколу для керування криптоактивами можна розділити на такі компоненти, як :

– розподілена генерація ключа (Destributed Key Generation);

– оновлення ключа(Refreshing Key);

– створення порогового підпису BLS (Boneh-Lynn-Shacham signature).

Ці компоненти є ключовими у протоколі , адже ними досягається вищий рівень конфіденційності та захищеності системи.

Алгоритм взаємодії та компонентів протоколу можна побачити на рисунку 3.1.

Рисунок 3.1 Алгоритм взаємодії компонентів протоколу

Кожен компонент має своє специфічне завдання та відіграє свою важливу роль. Компоненти можуть бути реалізовані окремими без впливу на інші компоненти. Це спрощує впровадження, тестування та подальше покращення протоколу.

Спершу було розроблено компонент розподіленої генерації ключа, так як він є базою для створення доступу до криптоактивів . Компонент буде виконувати наступні операції:

– генерацію секретного значення у прописаному діапазоні;
– генерацію поліномів на основі секрету;
– об’єднання отриманих частин від інших учасників.

Для генерації секретного значення учасника спершу потрібен буде конфігураційний файл у якому буде прописані тип еліптичної кривої, максимальна кількість учасників та поріг для подальшого відновлення підпису.

Даний конфігураційний файл буде надаватися разом з програмним забезпеченням та автоматично буде підтягуватися під час запуску вузла. Файл має наступний вигляд показаний на лістингу 3.1.

Лістинг 3.1 Структура конфігураційного файлу для протоколу розподіленої генерації ключа

```
# Distributed Key Generation Config
ELLIPTIC_CURVE='secp256k'
MAX_NUMBER_PEERS=3
THRESHOLD=2
```

Для реалізації генерації поліномів ми використаємо алгоритм Шаміра, де потрібно обчислювати поліноми , де їх рівень рівний пороговому значенню яке прописане в конфігураціях.

Демонстрацію реалізації алгоритму генерації поліномів із використанням мови програмування Python можна побачити на лістингу 3.2.

Лістинг 3.2 Функція генерації полінома

```
def generate_polynomial(t, q):
    return [
        random.randint(1, q - 1)
        for _ in range(t)
    ]
```

Тепер нам потрібно реалізувати функцію для генерації спільного ключі згідно алгоритму розглянутого у попередньому розділі. Згідно алгоритму нам потрібно сумувати усі отримані частинки та порахувати їх за модулем порядку еліптичної кривої. Використовуючи вбудовану функцію sum() ми можемо швидко обрахувати суму масиву та потім з отриманої суми обрахувати модуль використавши оператор %, для отримання модуля. Тим самим ми отримуємо спільний секретний ключ, та потім зберігаємо його шифрований вигляд у базу даних вузла.

Після розробки компонента з генерацією ключа, наступним етапом є розробка механізму інтервального оновлення ключа. Для оновлення ключа, нам необхідно генерувати “додаткові” поліноми які додаються до початкових поліномів без зміни їх загальної суми вільних членів. Це дозволяє зберегти публічний ключ незмінним. Для програмної реалізації ми застосуємо функціонал, який був написаний для компонента генерації ключа, що зменшить необхідність у дублюванні коді , що задовільняє принцим DRY(Don’t Repeat Yourself, не повторюй самого себе), а також додамо новий функціонал для додання старих часток і нових, реалізація наведена на лістингу 3.3.

Лістинг 3.3 Програмна реалізація оновлення ключа

```
for i in range(n):
    participant_shares = []
    for i in range(n):
```

```
        # Отримання старої частки з бази даних
        old_share = get_object_or_404(Share, participant_id=i + 1, share_index=j
    + 1)
        # Оновлення частки
        updated_share_value = (old_share.share_value + new_shares[i][j]) % q
    participant_shares.append(updated_share_value)
        # Оновлення частки у базі даних
        old_share.share_value = updated_share_value
        old_share.save()
      updated_shares.append(participant_shares)

    return updated_shares
```

Оновлення секретного ключа без зміни публічного ключа можливе завдяки додаванню додаткових поліномів до існуючих поліномів. Це дозволяє учасникам оновлювати свої секретні частки, зберігаючи той же публічний ключ. Таким чином, система може підтримувати довгострокову безпеку без необхідності зміни публічного ключа.

Заключним компонентом протоколу для системи керування криптоактивами, є реалізація процесу створення валідної підписаної транзакції використовуючи часткові підписи окремих учасників.

Для цього ми реалізуємо алгоритм BLS (Boneh-Lynn-Shacham) підпису що є потужним криптографічним інструментом, що забезпечує короткі та безпечні цифрові підписи з можливістю агрегування. В основі BLS підписів лежать криптографічні властивості парувань на еліптичних кривих. Це особливо корисно в контексті багатосторонніх обчислень, коли підписи від кількох учасників можуть бути об'єднані в один підпис. Отже компонент буде виконувати такі операції:

– обчислення часткового підпису;

– агрегація зібраних підписів.

Для отримання підпису повідомлення нам потрібно реалізувати програмний код , який буде виконувати формулу 3.1.

$$\sigma_i = sk_i * H(m) \quad (3.1)$$

де sk_i приватний ключ *i*-го вузла, H(m) хеш функція з повідомлення m.

Програмний код показано на лістингу 3.4.

Лістинг 3.4 Реалізація процесу створення часткового підпису

```
def hash_and_sign(message, sk):
    #Хешування повідомлення та генерація підпису.
    #Хешування повідомлення у точку на кривій G1
    digest = sha256(message.encode()).digest()
    h = multiply(G1, int.from_bytes(digest, byteorder: 'big') % curve_order)
    # Генерація підпису
    signature = multiply(h, sk)
return signature
```

У даному коді ми користуємося готовими бібліотеками для роботи з еліптичними кривими py_ecc та для хешування повідомлення бібліотеку hashlib. Використання готових бібліотек дозволяє знизити ризики, підвищити ефективність розробки та забезпечити високий рівень безпеки та надійності програмного забезпечення.

Після отримання часткових підписів від учасників нам потрібно їх агрегувати, тобто поєднати підписи в однин валідний підпис. Для цього потрібно обрахувати формулу 3.2.

$$\sigma = \prod_{i=1}^{t} \sigma_i \quad (3.2)$$

де t поріг для складання підпису, σ_i частковий підпис *i*-го вузла.

Програмний код показано на лістингу 3.5.

Лістинг 3.5 Програмний код агрегації часткових підписів

```
def point_addition(P: Point, Q: Point) -> Point:
    if P.x == Q.x and P.y == Q.y:
        m = (3 * P.x * P.x + A) * pow(2 * P.y, -1, q) % q
    else:
        m = (Q.y - P.y) * pow(Q.x - P.x, -1, q) % q
    x_r = (m * m - P.x - Q.x) % q
    y_r = (m * (P.x - x_r) - P.y) % q
    return Point(x_r, y_r)


def aggregate_signatures(signatures: List[Point]) -> Point:
    agg_sig = signatures[0]
    for sig in signatures[1:]:
        agg_sig = point_addition(agg_sig, sig)
    return agg_sig
```

Після реалізації усіх необхідних компонентів у нас є готовий інструмент для впровадження протоколу у роботу розподілених вузлів, які разом працюють як одна система керування криптоактивами.

3.2 Програмна реалізація вузла програмного забезпечення системи керування криптоактивами

Після реалізації основного функціоналу протоколу який буде виконувати необхідні нам операції ми приступимо до реалізації вузлів, які відіграють основну роль у роботі системи, так як виконують обчислення розподілено.

Саме завдяки такій структурі мережі та системи ми досягаємо необхідного рівню захисту.

У нашій системі буде два типи вузлів , а саме:

– генезис вузол;

– службові вузли.

Генезис вузол являється тим самим службовим вузлом , за винятку того , що він буде запущений самий перший і зберігатиме усю необхідну інформацію для вдалого розгортання системи. Тоді адміністратор мережі повинен буде прописати унікальні ідентифікатори вузлів та максимальну кількість з’єднань, так мережа буде ізольованою від сторонніх підключень, та можливість адміністрування та масштабування мережі залишається у довіреної особи, а саме безпосередньо власника цієї мережі.

Службові вузли виконуватимуть роль додаткової обчислювальної сили, які збільшуватимуть захист системи , за рахунок того що зберігатимуть частини закритого ключа у себе , та виконуватимуть протокол згідно алгоритму, для подальшої можливості підписання транзакції.

Так як по структурі генезис та службовий вузол однакові, нам буде вистачати реалізувати одну структуру, а потім вже передавати при запуску необхідні аргументи.

Отже для початку створимо порожній проект використовуючи команду django-admin startproject wallet-peer. У результаті виконання службової команди буде створена папка зі структурою Django проекту наведена на рисунку 3.3.

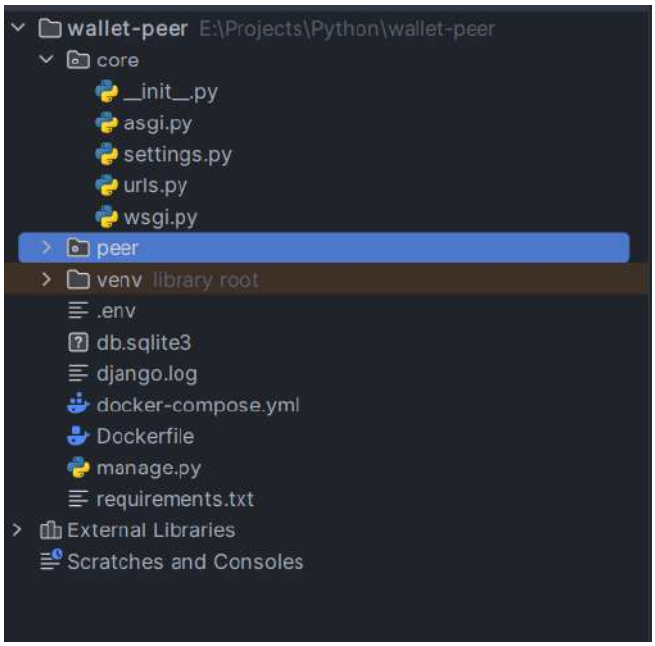


Рисунок 3.3 Структура програмного забезпечення

Далі перейдемо до реалізації кожного з компонентів вузла , які ми розглянули та виділили у попередньому розділ.

Спершу реалізуємо компонент мережевого представлення , а саме налаштуємо бібліотеку Django channels для обробки вхідних запитів на під'єднання інших вузлів , та можливість обробляти вхідні повідомлення. Для цього нам необхідно перейти у файл settings.py та додати команди наведені на лістингу 3.6.

Лістинг 3.6 Налаштування Django Channels

```
ASGI_APPLICATION = 'core.asgi.application'
CHANNEL_LAYERS = {
    'default': {
        'BACKEND': 'channels_redis.core.RedisChannelLayer',
        'CONFIG': {
            "hosts": [('127.0.0.1', 6379)],
        },
    },
}
```

Тут ми вказуємо шлях до нашого асинхронного серверного інтерфейсу та вказуємо Redis в якості сховища ключ-значення , для зберігання сесії з підключеними вузлами. Після цього необхідно налаштувати правильну маршрутизацію пакетів, щоб вузол міг відрізнити протоколи http та ws (websockets), для цього нам необхідно вказати для Django налаштування показані на лістингу 3.7.

Лістинг 3.7 Налаштування маршрутизації пакетів

```
application = ProtocolTypeRouter({
    "http": get_asgi_application(),
    "websocket": AuthMiddlewareStack(
        URLRouter([
            path('ws/', P2PConsumer.as_asgi()),
        ])
    ),
})
```

Далі нам необхідно прописати кінцеві точки через які користувач або третя сторона зможе ініціювати протокол для створення валідної адреси або підписання даних по транзакції. Для цього ми створюємо файл urls.py в папці peer та прописуємо шляхи до представлень так як показано на лістингу 3.6.

Лістинг 3.8 Кінцеві точки для взаємодії клієнта з системою

```
urlpatterns = (
    path(route="create_wallet/",
        view=WalletCreateAPIView.as_view(),
        name="create_wallet"),
    path(route="send_transaction/",
        view=SendTransactionAPIView.as_view(),
```

```
        name="send_transaction"),
    )
```

Після налаштування мережевого компоненту у нас є інтерфейс для клієнтів та інших вузлів нашої системи. Тепер розглянемо компонент локального сховища, з попереднього розділу ми візьмемо структуру бази даних та перенесемо їх у сутності за допомогою парадигми ООП (об’єктно-орієнтоване програмування), яка дозволяє структурувати код у вигляді об’єктів, які поєднують дані та функціонал , пов’язаний з цими даними.

Ця парадигма дозволяє зробити програму модульною, і в подальшому використовувати повторно код модулів в різних частинах програми. Також ООП дозволяє об’єднати дані та методи, які з ними працюють, усередині сутностей. Моделюючи реальний світ даний принцип дозволяє створювати моделі об’єктів реального світу у програмному коді.

Таким чином ми створюємо необхідні моделі бази для вузла, а саме частинки приватних ключів, підписи, інформацію про вузли та журнал подій системи вузла.

Для прикладу наведемо лістинг 3.9 де ми оголошуємо модель журналу подій за допомогою фреймворку Django.

Лістинг 3.9 Модель журналу подій для запису даних

```
class EventLog(models.Model):
    timestamp = models.DateTimeField(auto_now_add=True)
    event_type = models.CharField(max_length=50)
    message = models.TextField()

    def __str__(self):
        return f"{self.timestamp} - {self.event_type}"
```

Маючи усі необхідні сутності даних для вузла ми можемо здійснювати операції над даними та моментально отримувати їх з бази даних , де вони зберігатимуться у шифрованому стані. Для шифрування буде використовуватися готовий функціонал з бібліотеки pycryptodone для виконання алгоритму симетричного шифрування AES. У якості ключа, використовуватиметься вбудована змінна DJANGO_SECRET_KEY , що генерується при кожному новому проекті яка є унікальною та стійкою до атаки перебору значень.

Далі необхідно реалізувати компоненти , які призначенні для виконання протоколу та будуть використовуватися для виконання основних операцій. Для цього ми інтегруємо написаний код протоколу в представлення та функції.

Для цього потрібно написати обробник типів повідомлень , який буде скеровувати програму на необхідний функціонал. Це відбувається шляхом отримання відповідної команди у метод отримання повідомлень на вебсокет, і коли відповідна команда типу “generate_key”, то відбувається звернення до методу протоколу для генерації частинок ключа та після вони розсилаються викликом метода відправки повідомлень через цикл, де вони перебувають у шифрованому стані та адресовані відповідному ідентифікатору вузла, що зберігаються в базі даних вузла.

Для того щоб вузол почав працювати, нам залишається реалізувати компонент конфігурацій, а саме систему зчитування конфігураційних файлів від користувача. Використовуючи бібліотеку django-environ ми зможемо безпечно зчитати конфігураційний файл при запуску вузла.

Структура файлу конфігурацій наведена на лістингу 3.10.

```
# Зчитуємо секретний ключ Django з оточення
SECRET_KEY = 'DJANGO_SECRET_KEY'
# Список можливих вузлів для підключення
NODES = [
```

```
    {"id": "node1", "host": "example.com", "port": 8000},
    {"id": "node2", "host": "example.org", "port": 8000},
    # додайте інші вузли, які можуть бути доступні для підключення
]
# Параметри для порогу відновлення
THRESHOLD = 3  # Приклад, якщо не вказано в середовищі
# Пароль до мережі
NETWORK_PASSWORD = 'NETWORK_PASSWORD'
```

По завершенню розробки ми маємо програмне забезпечення , яким можна запустити в роботу нашу систему керування криптоактивами, де усі операції будуть виконуватися розподілено на різних вузлах. Така схема забезпечує захист системи від атак єдиної точки відмови, та має активний захист використовуючи періодичне оновлення приватних частинок ключа.

3.3 Розгортання та тестування системи керування криптоактивами на основі технології багатосторонніх обчислень

Наступним етапом ,після завершення розробки програмної реалізації протоколу багатосторонніх обчислень для системи та вузла програмного забезпечення системи, є розгортання системи в глобальній мережі, в локальній системі та можливість тестування системи.

Процес розгортання програмного забезпечення є ключовим етапом у життєвому циклі розробки програмних продуктів. Правильне розгортання гарантує, що система керування криптоактивами буде готова до ефективного використання і відповідатиме всім потребам користувачів. Цей процес може включати в себе різноманітні дії та процедури, які забезпечують успішну інсталяцію та налагодження програми.

Один із ключових аспектів розгортання системи для глобальної мережі є вибір хмарних провайдерів для хостингу вузлів системи. Кожен провайдер може мати свої унікальні переваги та можливості, тому важливо обрати того, хто найбільш відповідає вимогам системи керування криптоактивами із використанням технології багатосторонніх обчислень.

Основні вимоги щодо провайдера хостингу:

– безпека;
– продуктивність;
– відмовостійкість;
– масштабованість;
– вартість;

Провайдер хмарних послуг ,повинен надавати найвищий рівень безпеки для збереження криптоактивів та забезпечення захисту від кібератак.

Хостинговий сервер повинен мати високу продуктивність обчислень для швидкої та ефективної роботи вузів системи керування криптоактивами. Сервер повинен бути оснащеним швидкісним інтернетом для швидкої комунікації вузлів системи.

Система повинна бути здатна масштабуватися відповідно до зростаючих потреб обсягу обчислень та зберігання криптоактивів, отже провайдер повинен гарантувати можливість покращення характеристик серверу зі збереженням роботи сервісу.

Вартість послуг провайдера повинна бути конкурентоспроможною та відповідати бюджетним обмеженням проекту.

Зважаючи на ці критерії та рекомендації спільноти DevOps програмістів[39], у якості провайдера хостингу для вузлів системи було обрано Amazon Web Services(AWS), так як він відповідає вимогам безпеки, продуктивності, відмовостійкості, масштабованості та вартості, що були визначені для системи керування криптоактивами.

Для забезпечення надійної та стабільної роботи системи необхідно мінімум три екземпляри сервера для кожного з вузлів системи. Важливо обрати однаковий регіон розгортання у меню провайдера, оскільки це забезпечить швидшу комунікацію між серверами, зменшуючи затримки і підвищуючи продуктивність системи. Розміщення серверів у одному регіоні також сприяє кращій синхронізації даних та оптимізує роботу мережі.

Далі, після вибору провайдера, необхідно створити та налаштувати інстанси програмного забезпечення. Це включає в себе налаштування конфігурацій та встановлення необхідного програмного забезпечення. Меню створення та налаштування характеристів екземпляру серверу наведено на рисунку 3.4.

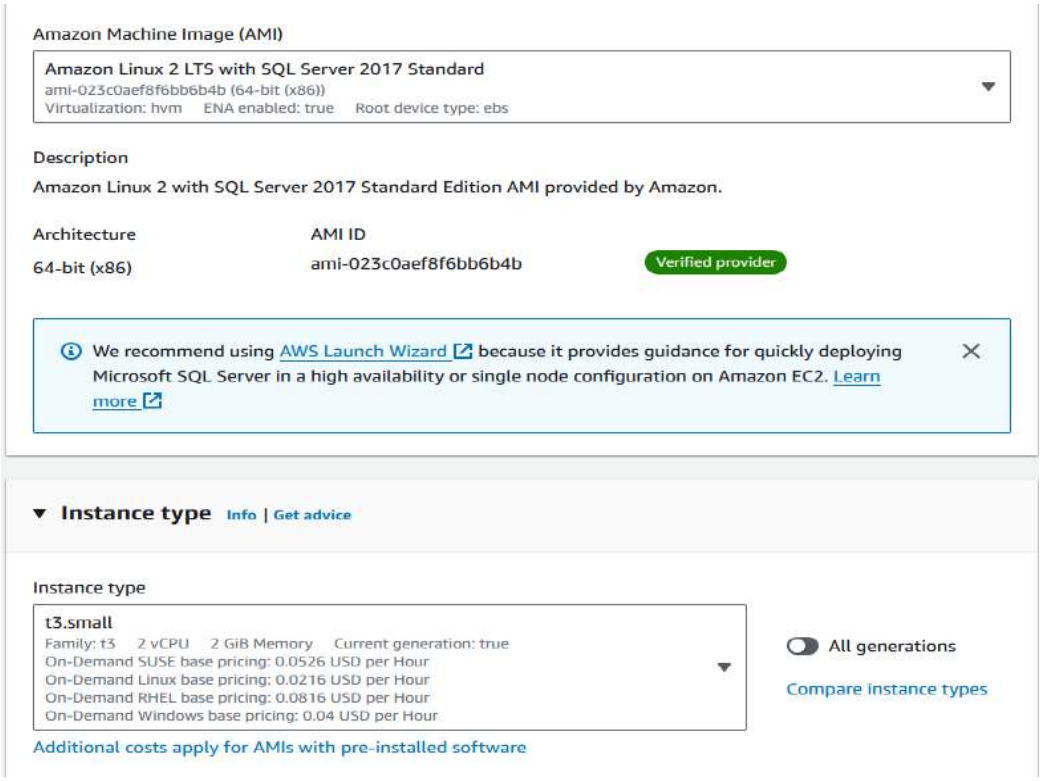


Рисунок 3.4 Меню створення та налаштуванням серверу для вузла системи

Після того як ми обрали конфігурації для серверу , нам надається файл з обліковими даними для доступа через мережевий протокол прикладного рівня Secure Shell (SSH) до екзепляру.

Отримавши доступ до системи, нам необхідно встановити необхідні залежності для коректної роботи системи, а саме інтерпретатор для мови програмування Python та системи контейнеризації Docker, для легкого запуску сервісів з якими взаємодіє вузол системи такі як база даних Postgres та системи кешування Redis.

Після цього важливо забезпечити безпеку системи через налагодження захищеного зв'язку, наприклад, за допомогою OpenVPN сервера[40] та налаштування клієнтів на інших вузлах. Це дозволяє забезпечити безпечний канал зв'язку між різними частинами системи. У якості сервера оберемо один із вузлів системи та оголосимо необхідні конфігурації , а саме потрібно вказати тип пристрою (у нашому випадку універсальний тунельний пристрій), IP-адреси вузла-сервера та вузла-клієнтів приватної мережі та шлях до файлу ключа, який використовується для шифрування та розшифрування трафіку між сервером та клієнтом.

Налаштування маршрутизації також є важливим етапом. Створення DNS-зон для кожного вузла та використання сервісів DNS, як AWS Route 53, дозволяє ефективно розподіляти трафік між різними частинами системи, забезпечуючи при цьому стабільність та високу доступність. Для початку необхідно створення DNS-зон для кожного компонента системи, відбувається це через консоль провайдера де вказується тип операції та доменне ім'я для вузла системи. DNS-зона дозволяє ідентифікувати вузли системи за їхніми доменними іменами. Після цього необхідно налаштувати DNS-сервери для правильної обробки запитів до цих зон, використовуючи послуги провайдера з реєструванням доменних імен на відповідних власних серверах. Схема розгорнутої системи має вигляд показаний на рисунку 3.5.

Після чого потрібно запустити вузли з відповідними налаштуваннями системи для валідної роботи підписів, для цього потрібно вказати потрібний вид еліптичної кривої відповідно до типу криптоактиву, який буде зберігатися системою вузлів.

Для демонстрації роботи та тестування системи керування криптоактивами, було обрано провести локальне розгортання використовуючи Docker та Docker Compose для створення локального середовища, яке імітує роботу глобальної системи. Нам потрібно створити в директорії проекту два файли: DockerFile та docker-compose.yml.

Dockerfile містить перелік команд, які визначають, що відбувається в середовищі контейнера.

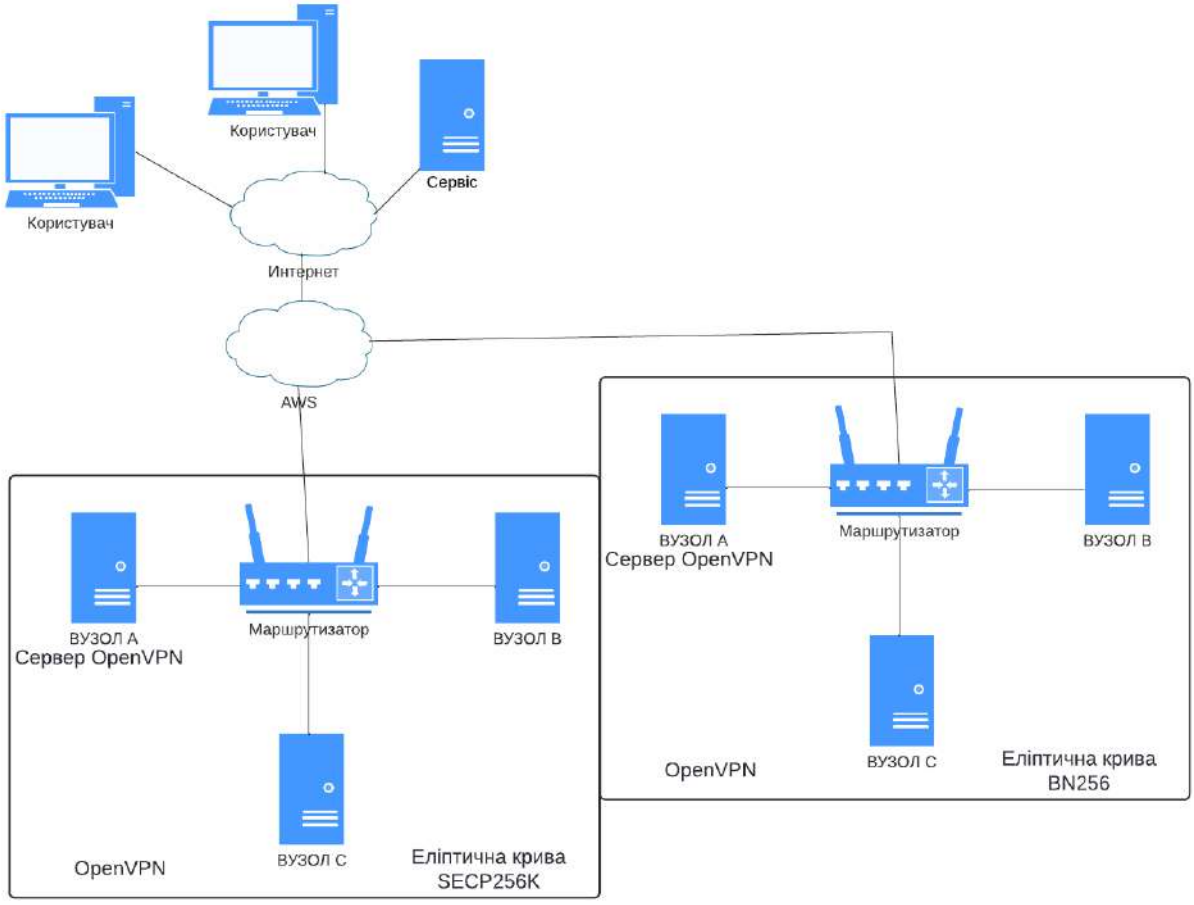


Рисунок 3.5 Схема розгорнутої системи керування криптоактивами для різних типів еліптичних кривих

Файл docker-compose.yaml є конфігураційним файлом для Docker Compose, інструменту, що спрощує опис та запуск багатоконтейнерних додатків. У ньому ми описуємо сервіси які використовуватиме розгорнутий

локально вузол системи, а саме базу даних , систему Redis, сервіси для вебсокетного з'єднання та для запитів http. Для кожного із сервісів відповідно прописуємо порти на яких вони будуть працювати, а також правила зібрання компонентів та шляхи до файлів, де docker зможе зберігати необхідні файли для легшого повторного запуску. Структура файлу для вузла системи показана на малюнку 3.6.

Для початку тестування необхідно запустити один генезис вузол системи та два службових вузли системи, отримати порти на яких вони запущені та прописати у відповідний конфігураційний файл для генезис вузла, для того щоб він міг надавати інформацію під'єднаним вузлам системи про інші компоненти структури програмного забезпечення. Для цього ми заходимо у корінь програми та прописуємо команду docker-compose up -d –build, яка запустить збірку проекту докером , де він підкачає усі необхідні залежності.

```yaml
services:
  django_wsgi:
    container_name: django_wsgi
    build: .
    command: [ "gunicorn", "core.wsgi:application", "--bind", "0.0.0.0:9000" ]
    volumes:
      - .:/code
    ports:
      - "9000:9000"

  django_asgi:
    container_name: django_asgi
    build: .
    # command: python manage.py runserver 0.0.0.0:8000
    command: daphne -b 0.0.0.0 -p 8000 core.asgi:application
    volumes:
      - .:/code
    ports:
      - "8000:8000"
```

Рисунок 3.6 Структура файлі docker-compose.yml для вузла системи керування криптоактивами

Після запуску програмного вузла, у логах докер файлу будуть наступі інформаційні повідомлення показані на рисунку 3.7.

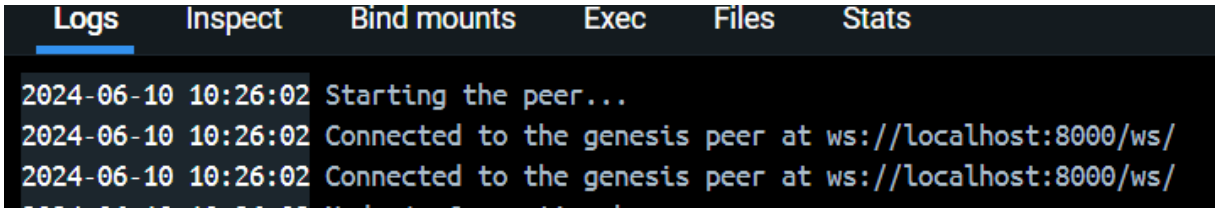


Рисунок 3.7 Логи запуска вузла програмного забезпечення сиситеми

Тепер нам необхідно скористатися програмною Postman[41] для виклику кінцевої точки для генерації Ethereum адреси. Після виклику POST методу отримаємо відповідь ,що показана на рисунку 3.8. У логах вузла це виглядає наступним чином показним на рисунку 3.9.

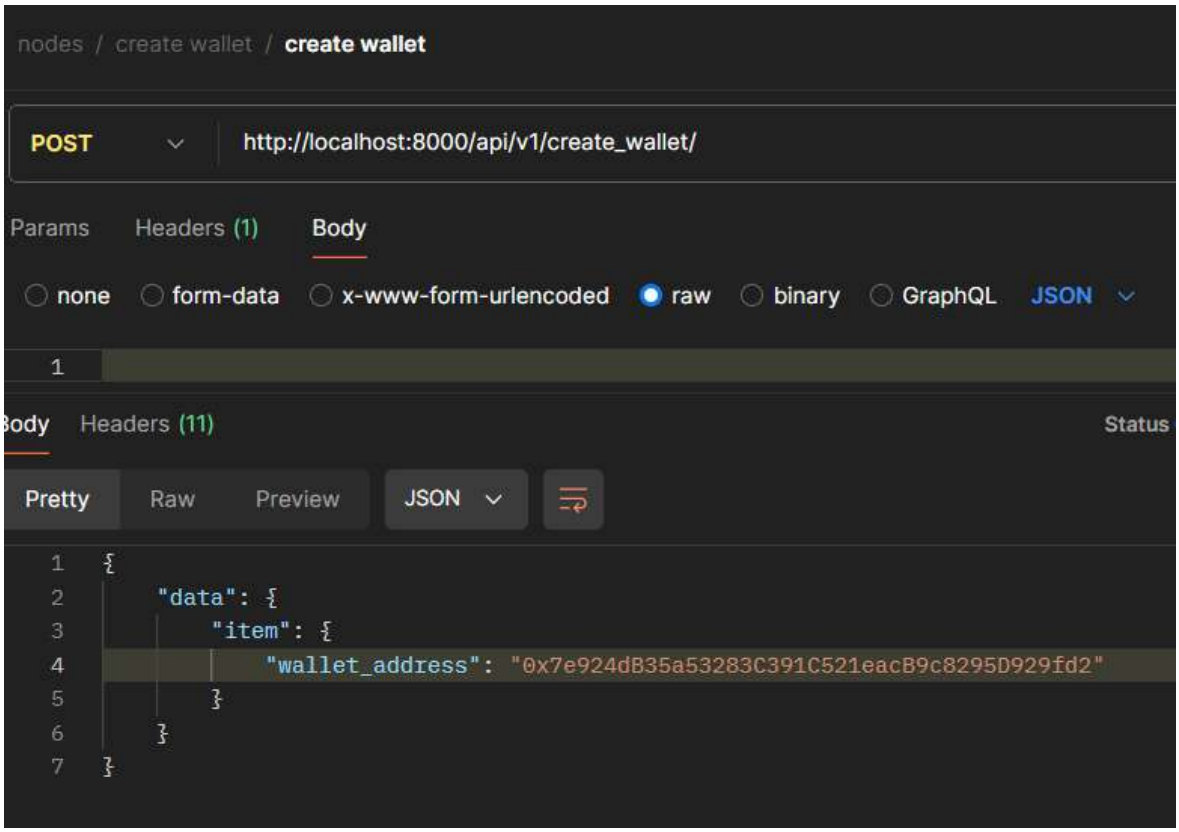


Рисунок 3.8 Створення адреси для Ethereum мережі використовуючи вузол системи керування криптоактивами

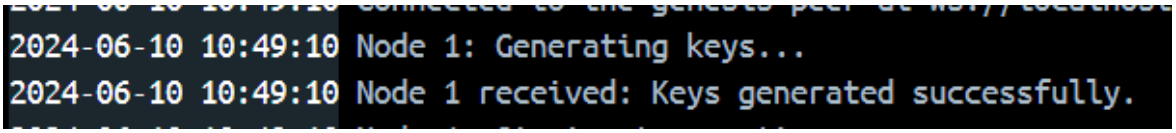


Рисунок 3.9 Логи вузла системи керування криптоактивами

Після створення адреси , нам необхідно перевірити валідність створення підписаної адреси, для цього ми скористаємося тестовою мережею Sepolia, де зможемо здійснити транзакцію без витрачення реальних коштів, для отримання тестових криптоактивів що відповідає цифровому долару tUSDT можна скористатися , так званими, «кранами».

Отримавши тестові криптоактиви, нам необхідно викликати кінцеві точки для підписання даних транзакції. Для цього ми знову заходимо в Postman та викликаємо необхідний POST метод. Результат виконання показаний на рисунку 3.10.

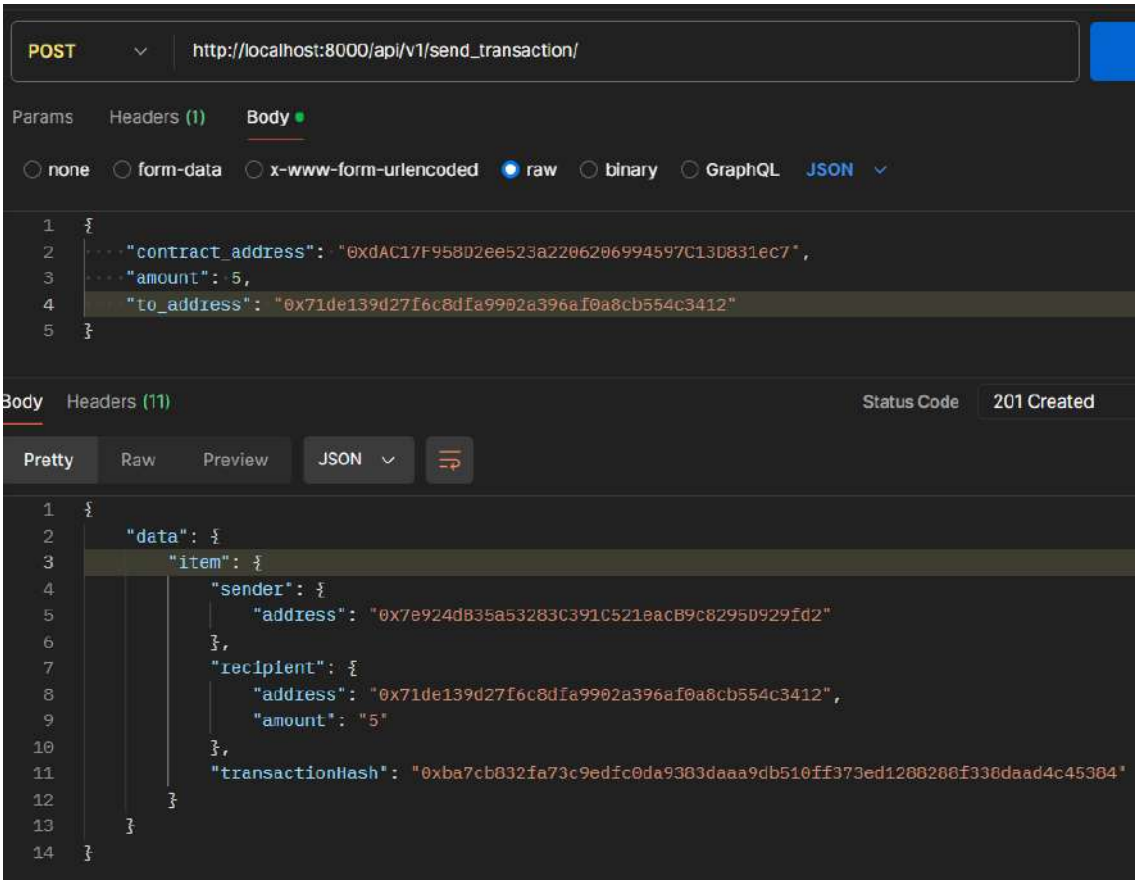


Рисунок 3.10 Підписання транзакції Ethereum використовуючи вузол програмного забезпечення системи керування криптоактивами

Щоб переконатися , що транзакція дійсно виконалася скористаємося веб сервісом Etherscan, що надає прозору інформацію про стан мережі. Для цього у пошукову строку введемо хеш отриманої транзакції. Інформація від Etherscan показана на рисунку 3.11.

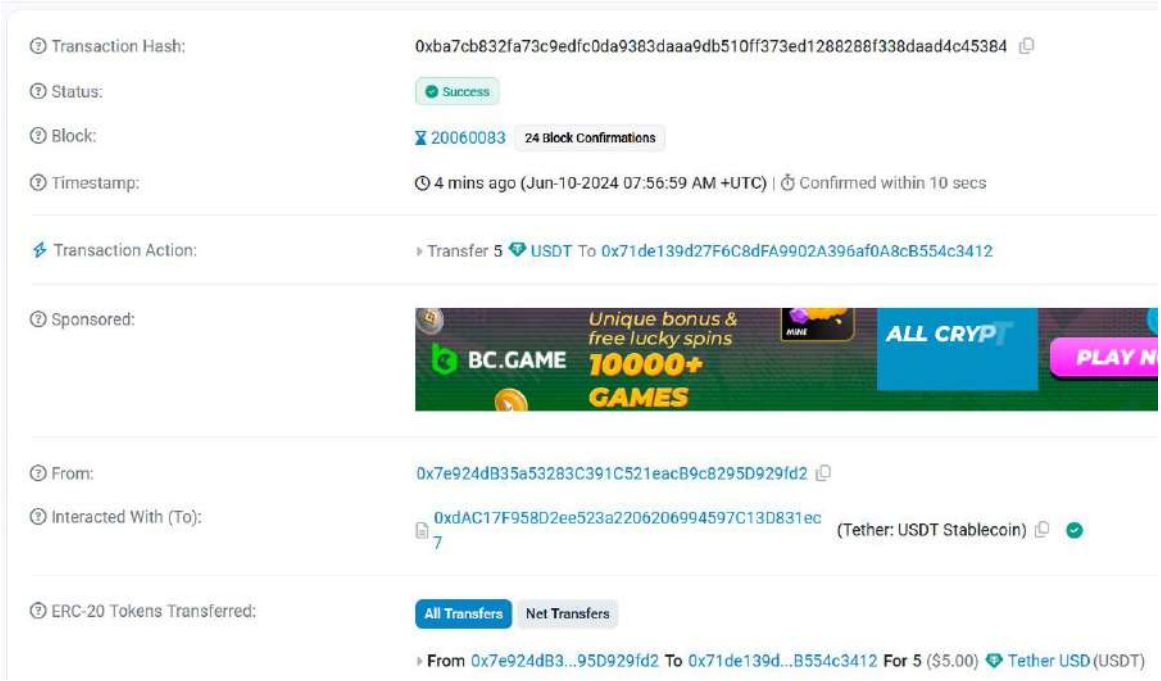


Рисунок 3.11 Інформація по хешу транзакції від сервісу Etherscan

3.4 Висновки

Розгортання системи керування криптоактивами на основі багатосторонніх обчислень включає кілька важливих етапів: вибір хмарних провайдерів, створення та налаштування інстансів, контейнеризація додатків, налаштування захищеного зв’язку та маршрутизації.

Локальне тестування системи дозволяє перевірити коректність її роботи перед глобальним розгортанням, що знижує ризики та забезпечує надійність і безпеку управління криптоактивами.

Таким чином, ми створили систему для керування криптоактивами, яка використовує технології багатосторонніх обчислень. Ця система дозволяє генерувати ключі та підписувати транзакції , виконуючи операції між різними вузлами. Це дозволяє підвищити безпеку активів , адже приватний ключ перебуває у розподіленому стані.

ВИСНОВКИ

У рамках кваліфікаційної роботи було проведено дослідження предметної області та розробка системи для надійного зберігання криптоактивів, зокрема приватних ключів до гаманців Ethereum мережі, з використанням технології багатосторонніх обчислень (MPC), яка може бути використана у різних криптопроектах для безпечного переказу криптоактивів.

На основі аналізу характеристик та стандартів протоколів багатосторонніх обчислень був розроблений протокол, який враховує наявні покращення та недоліки попередніх протоколів. Для покращення безпеки, у протоколі було впроваджено принцип інтервального оновлення спільного секрету.

Архітектура системи передбачає взаємодію основних компонентів для забезпечення надійної та безпечної роботи. Було спроектовано схему встановлення безпечного з'єднання вузлів та описано модель локального сховища, яке забезпечує безпечне зберігання часток ключів, підписів та системних логів.

Для розробки та тестування системи було обрано мову програмування Python та веб фреймворки для створення веб сокетного з'єднання та API інтерфейсу. Система була контейнеризована за допомогою Docker для полегшення тестування та розгортання.

Розгортання системи включало вибір хмарного провайдера, створення та налаштування екземплярів серверу, налаштування захищеного зв'язку через OpenVPN та маршрутизації. Локальне тестування системи забезпечило перевірку її коректності перед глобальним розгортанням.

Таким чином, розроблена система дозволяє безпечно керувати криптоактивами, здійснювати генерацію ключів та підписання транзакцій за допомогою технології багатосторонніх обчислень, підвищуючи рівень безпеки та конфіденційності криптоактивів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. What Makes a Blockchain Secure? Академія Binance : веб-сайт. URL: https://academy.binance.com/en/articles/what-makes-a-blockchain-secure (дата звернення 01.05.2024).
2. Що таке блокчейн і як він працює? Академія Binance : веб-сайт. URL: https://academy.binance.com/uk/articles/what-is-blockchain-and-how-does-it-work (дата звернення 01.05.2024).
3. Що таке транзакції у блокчейні? Incrypted : веб-сайт. URL: https://incrypted.com/tranzakcii-v-blokcheyn/ (дата звернення 01.05.2024).
4. Що таке ECDSA і чому він Важливий для Цифрових Підписів. JestJournal : веб-сайт. URL: https://jestjournal.com.ua/crypto/shcho-take-ecdsa-i-chomu-vin-vazhlyvyy-dlia-tsyfrovykh-pidpysiv/ (дата звернення 03.05.2024).
5. Що таке криптовалютний гаманець? Академія Binance : веб-сайт. URL: https://academy.binance.com/uk/articles/crypto-wallet-types-explained (дата звернення 05.05.2024).
6. Де зберігати криптовалюту: вимоги до криптогаманців. Голос Конотопа : веб-сайт. URL: https://konotop.in.ua/de-zberigati-kriptovalyutu-vimogi-do-kriptogamanciv/ (дата звернення 05.05.2024).
7. Definition: blockchain. ComputerLanguage : веб-сайт. URL: https://www.computerlanguage.com/results.php?definition=blockchain.
8. What Is a Merkle Tree & What Is Its Role in Blockchain? Learn ByBit : веб-сайт. URL: https://learn.bybit.com/blockchain/what-is-merkle-tree/ (дата звернення 06.05.2024).
9. Блокчейн. Wikipedia : веб-сайт. URL: https://uk.wikipedia.org/wiki/%D0%91%D0%BB%D0%BE%D0%BA%D1%87%D0%B5%D0%B9%D0%BD.
10. Proof of Work (PoW) Process flow. ResearchGate : веб-сайт. URL: https://www.researchgate.net/figure/Proof-of-Work-PoW-Process-flow-Latif-et-al-2021_fig2_374870812.

11. Why is so Much Important to have Digital Signatures? Medium : веб-сайт. URL: https://saurabh57788.medium.com/why-is-so-much-important-to-have-digital-signatures-8583abec63d1.

12. Blockchain vs. Traditional Database: What Should Be a Startup’s Choice. AppInventiv : веб-сайт. URL: https://appinventiv.com/blog/traditional-database-vs-blockchain/.

13. Які бувають гаманці для криптовалют? Блог ByBit : веб-сайт. URL: https://blog.whitebit.com/uk/types-of-crypto-wallets/.

14. Кастодіальні та некастодіальні гаманці: у чому різниця? Академія Binance : веб-сайт. URL: https://academy.binance.com/uk/articles/custodial-vs-non-custodial-wallets-what-s-the-difference (дата звернення 07.05.2024).

15. An overview of Multi-Party Computation (MPC), Threshold Signatures (TSS) and MPC-TSS wallets. Medium : веб-сайт. URL: https://mmasmoudi.medium.com/an-overview-of-multi-party-computation-mpc-threshold-signatures-tss-and-mpc-tss-wallets-4253adacd1b2 (дата звернення 10.05.2024).

16. Secure multi-party computation. Wikipedia : веб-сайт. URL: https://en.wikipedia.org/wiki/Secure_multi-party_computation (дата звернення 10.05.2024).

17. What is Multi-Party Computation (MPC)? Certik : веб-сайт. URL: https://www.certik.com/resources/blog/SlB0bOnxC10TxOgdcYNbJ-what-is-multi-party-computation-mpc.

18. Threshold Signatures. Medium : веб-сайт. URL: https://scryptplatform.medium.com/threshold-signatures-a0eff03dc29c.

19. Building A New Digital World: Threshold Signing and Key Distribution Generation. Medium : веб-сайт. URL: https://medium.com/asecuritysite-when-bob-met-alice/building-a-new-digital-world-threshold-signing-and-key-distribution-generation-a1235390b6aa.

20. Threshold Signatures Explained. Академія Binance : веб-сайт. URL: https://academy.binance.com/uk/articles/threshold-signatures-explained (дата звернення).

21. Multisig vs. (SSS vs. TSS). Typefully : веб-сайт. URL: https://typefully.com/tomkowalczyk/JAfJzji (дата звернення 12.05.2024).

22. Як працює хешування в блокчені? Plisio : веб-сайт. URL: https://plisio.net/uk/blog/how-does-hashing-in-blockchain-work (дата звернення).

23. WebSocket. Wikipedia : веб-сайт. URL: https://uk.wikipedia.org/wiki/WebSocket (дата звернення 13.05.2024).

24. G.Tillem, O.Burundukov. Threshold Signatures using Secure Multiparty Computation. *ING DLT Team*. 2020. (дата звернення 13.05.2024).

25. Proactive Secret Share for Eigen Secret Recovery. EigenLab. 2021. URL: https://www.pdau.edu.ua/sites/default/files/node/4518/pravylaoformlennyaspyskuvykorystanyhdzherel.pdf (дата звернення 14.05.2024).

26. Andreas M. Antonopulos (2017) Mastering Bitcoin: Programming theOpen Blockchain. Second Edition. (дата звернення);

27. Daniel A. D., Emalda R. S. WSN Security: An Asymmetric Encryption and Hash Function based Approach. International Journal of Engineering Research and. 2021. T. V5, № 02.

28. Методи і алгоритми захисту інформаційних ресурсів комп'ютерних систем: навчальний посібник / В. М. Джулій, Ю. П. Кльоц, І. В. Муляр, В. М. Чешун. Хмельницький: ХмНУ, 2020. 196 с;

29. Don Johnson, Alfred Menezes, Scott Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security. Volume 1. 2001. С. 36-63.

30. Majid Mumtaz & Luo Ping. Forty years of attacks on the RSA cryptosystem:Abriefsurvey, JournalofDiscreteMathematicalSciences and Cryptograph. 2019. С. 9-29. URL: https://doi.org/10.1080/09720529.2018.1564201.

31. Abhishek Mishra & Ashutosh Gupta .Multisecretsharingscheme using iterative method, Journal of Information and Optimization Sciences. 2018. С. 631-641, URL: https://doi.org/10.1080/02522667.2017.1385161.

32. Feihu Xu, Xiongfeng Ma, Qiang Zhang, Hoi-Kwong Lo, and Jian-Wei Pan. Secure quantum key distribution with realistic devices. Rev. Mod. Phys. 2020. 96 ст.;

33. Jens Groth and Victor Shoup. Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506. 2022. URL: https://eprint.iacr.org/2022/506.

34. Peer-to-peer. Wikipedia : веб-сайт. URL: https://uk.m.wikipedia.org/wiki/Peer-to-peer (дата звернення 11.05.2024);

35. What is MPC (Multi-Party Computation)? Fireblocks : веб-сайт. URL: https://www.fireblocks.com/what-is-mpc (дата звернення 15.05.2024);

36. Django ORM і QuerySets. DjangoGirls : веб-сайт. URL: https://tutorial.djangogirls.org/uk/django_orm/ (дата звернення17.05.2024);

37. PostgreSQL Configuration : Best Practices for Performance and Security – Berkley, United States: aPress, 2020. – 226 с. (дата звернення 20.05.2024).

38. How To Set Up Django with Postgres, Nginx, and Daphne (Django Channels) on Ubuntu 20.04. Medium : веб-сайт. URL: https://okbaboularaoui.medium.com/how-to-set-up-django-with-postgres-nginx-and-daphne-django-channels-on-ubuntu-20-04-b0d24dcc7da9.

39. Топ 5 хмарних провайдерів: плюси та мінуси. Itedu : веб-сайт. URL: https://itedu.center/ua/blog/sysadministration/top-5-cloud-providers-advantages-and-disadvantages/ (дата звернення 22.05.2024).

40. Reference manual for OpenVPN 2.4. OpenVPN : веб-сайт. URL: https://openvpn.net/community-resources/reference-manual-for-openvpn-2-4/ (дата звернення).

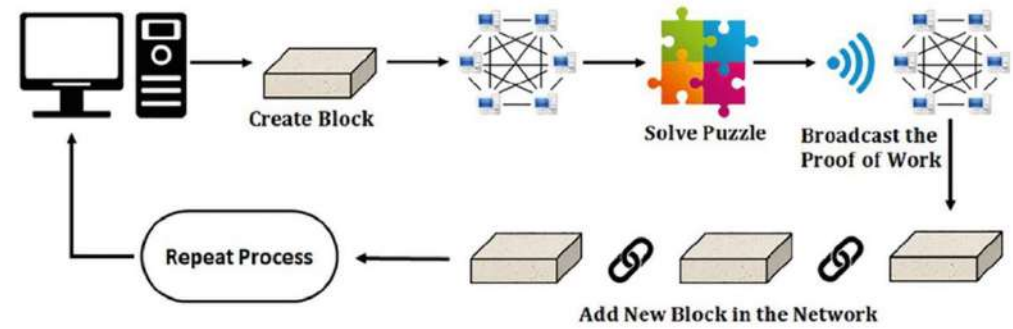
41. Postman documentation overview. Postman : веб-сайт. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення 25.05.2024).

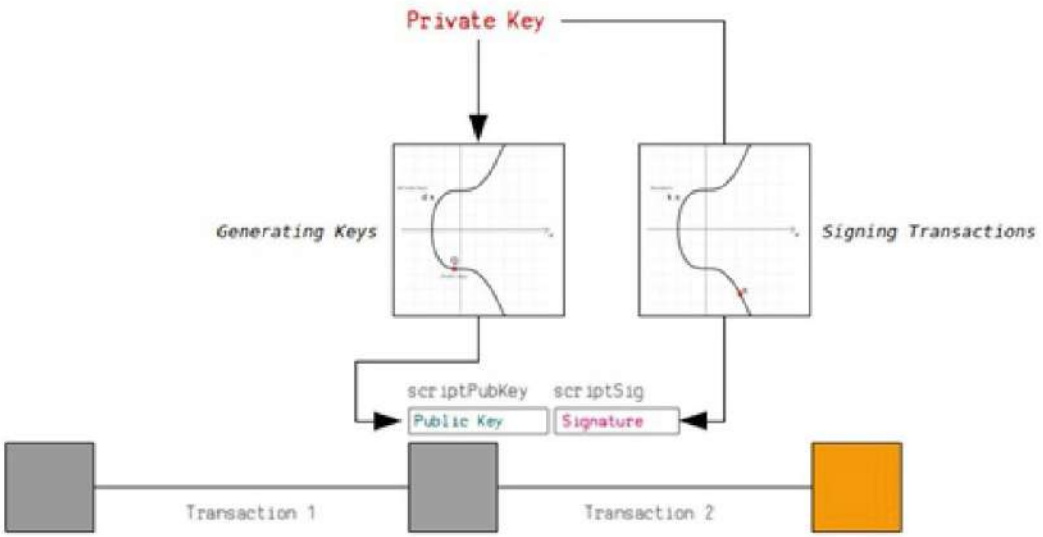
42. Docker : Complete Guide To Docker For Beginners And Intermediates, 2020. – 140 с. (дата звернення 01.06.2024).

ДОДАТОК А

(Обов’язковий)

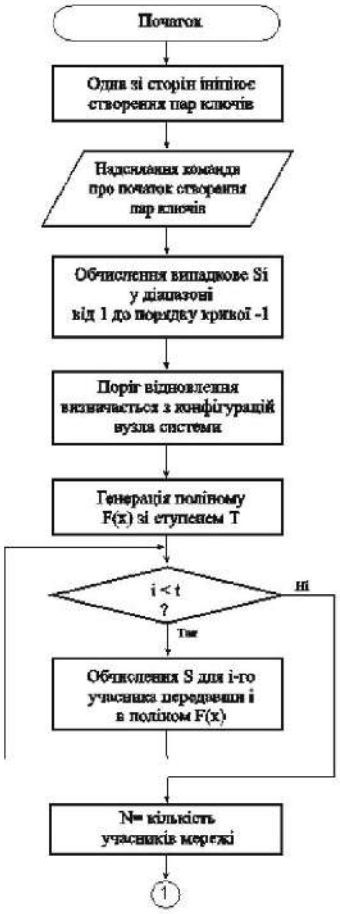
Копії графічної частини

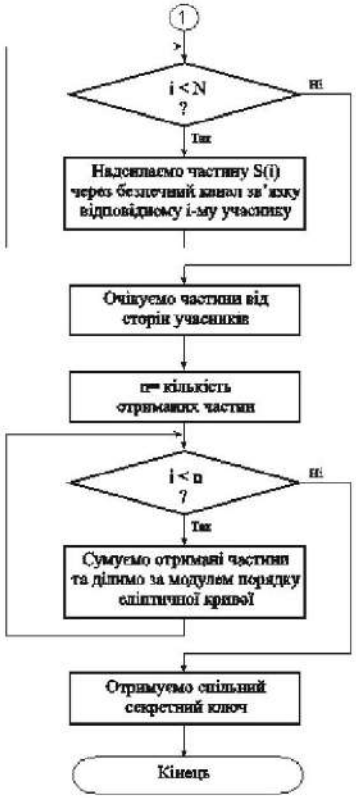


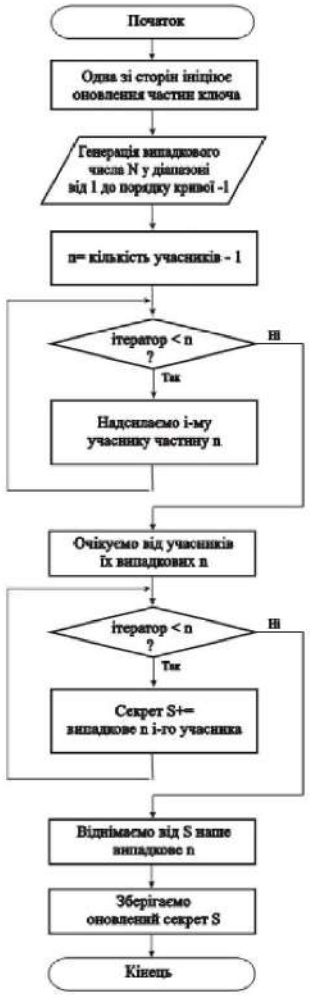


					КвРКБ.200116.20.01.16 ПЗ		
Зм	Арк.	№ докум.	Підпис	Дата	Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation Принципи роботи блокчейн технології	Літ	Маса / Масштаб
Розроб.		Фінок Є.В.				У	
Перевір.		Джулій В.М.				Аркуш	Аркушів 1
Н.контр.		Мостовий С.Б.					
Т.контр.						ХНУ, КБ-20-1	
Затверд.		Кльоц Ю.П.					

КвРКБ.200116.20.01.18 ПЗ

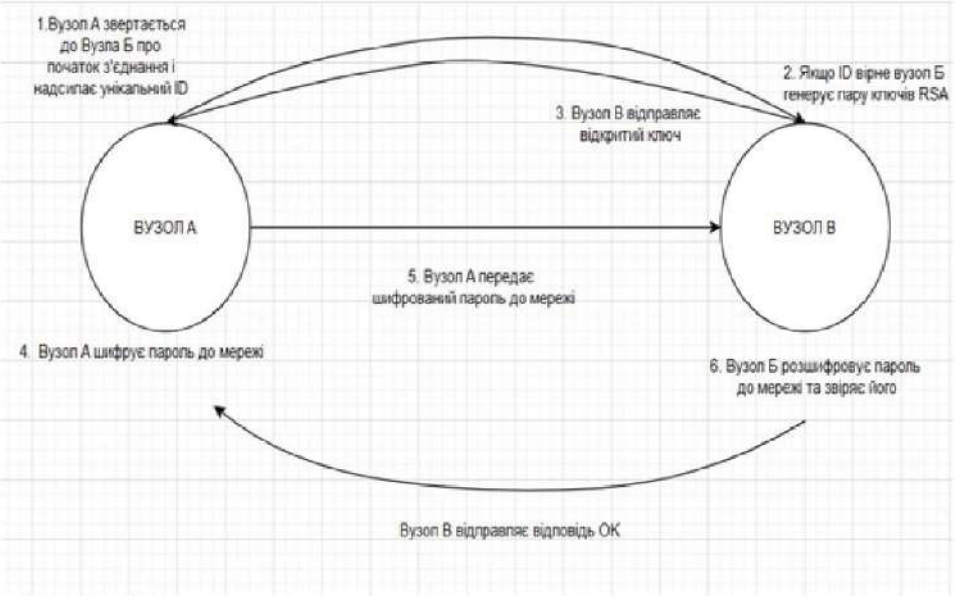


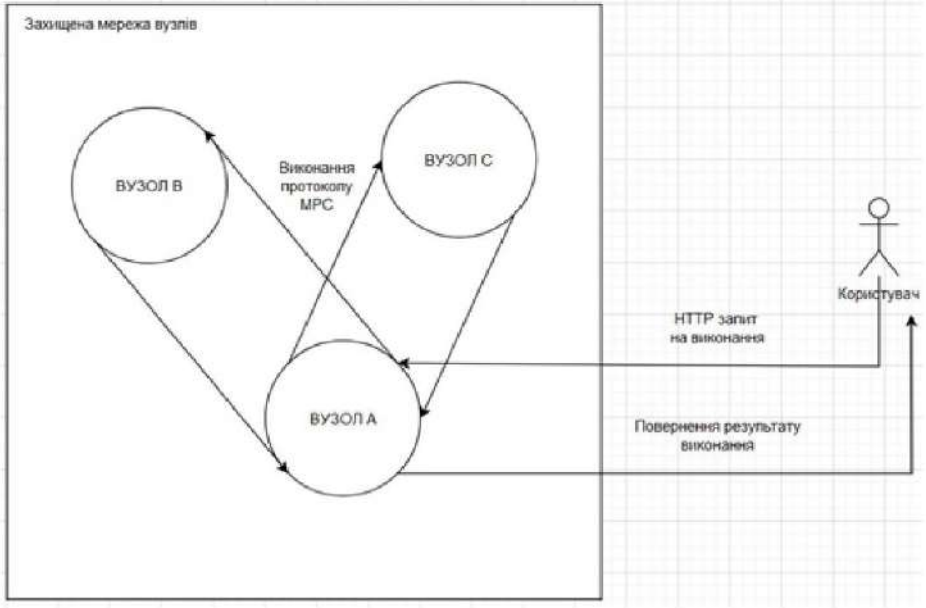




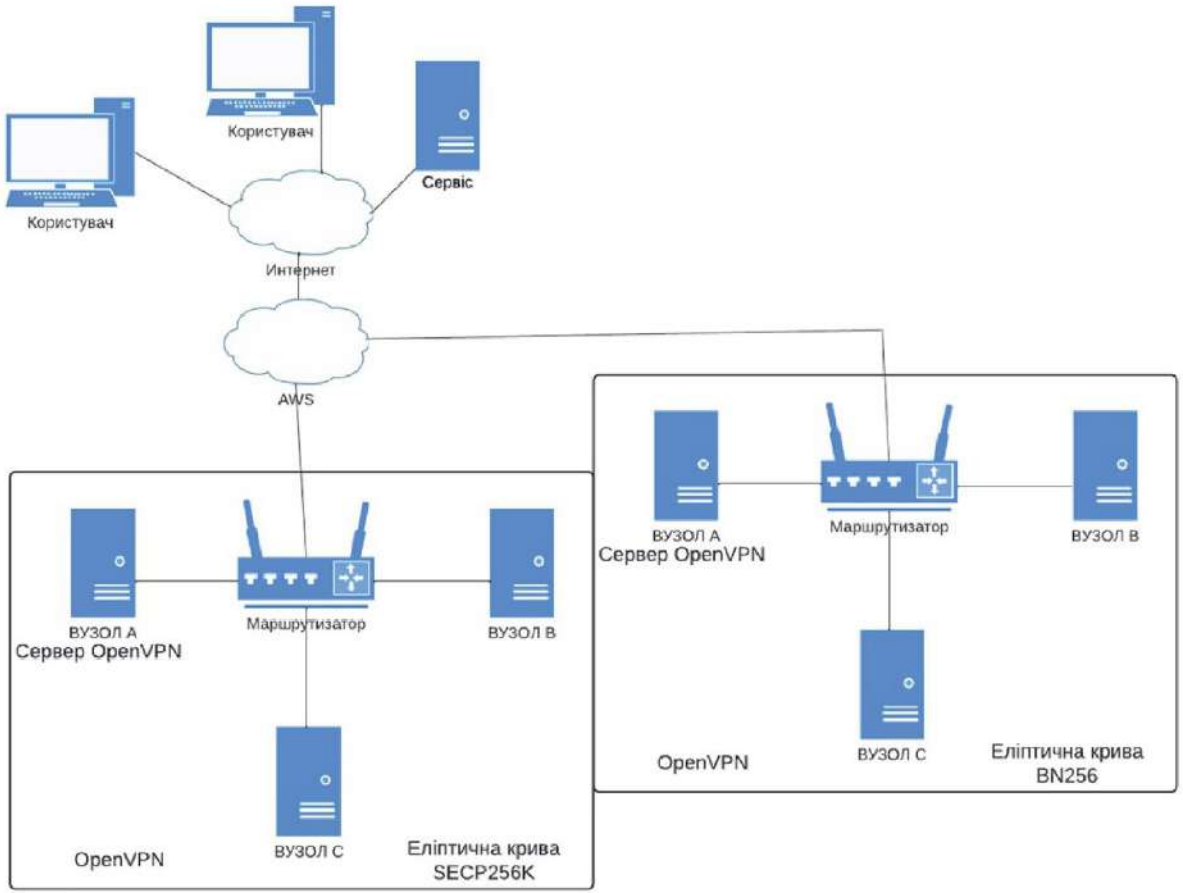
					КвРКБ.200116.20.01.16 ПЗ		
Зм.	Арк.	№ докум.	Підпис	Дата	Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation	Літ.	Маса	Масштаб
Розроб.		Філюк Є.В.			Алгоритм роботи протоколу багатосторонніх обчислень системи	У		
Перевір.		Джулій В.М.				Аркуш	Аркушів 1	
Н.контр.		Мостовий С.В				ХНУ, КБ-20-1		
Т.контр.								
Затверд.		Кльоц Ю.П.						

КвРКБ.200116.20.01.18 ПЗ





					КвРКБ.200116.20.01.16 ПЗ			
Зм	Арк.	№ докум.	Підпис	Дата	Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation	Літ	Маса	Масштаб
Розроб.		Філюк Є.В.			Схема взаємодії вузлів програмного забезпечення системи	У		
Перевір.		Джулій В.М.				Аркуш	Аркушів	1
Н.контр.		Мостовий С.В.				ХНУ, КБ-20-1		
Т.контр.								
Затверд.		Кльоц Ю.П.						



					КвРКБ.200116.20.01.16 ПЗ
Зм.	Арк.	№ докум.	Підпис	Дата	Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation. Розгорнута системи керування криптоактивами на основі технології багатосторонніх обчислень.
Розроб.		Філюк Є.В.			
Перевір.		Джулій В.М.			
Н.контр.		Мостовий С.В.			
Т.контр.					
Затверд.		Кльоц Ю.П.			

Літ	Маса	Масштаб
У		
Аркуш	Аркушів 1	
ХНУ, КБ-20-1		

ДОДАТОК Б

(Обов’язковий)

Програмний код вузла програмного забезпечення системи

Клас для виконання усіх криптографічних операцій:

```
from typing import List, Tuple
from random import SystemRandom
from hashlib import sha256
from django.conf import settings

# Ініціалізація параметрів кривої з налаштувань
A = settings.ELLIPTIC_CURVE_PARAMS['A']
B = settings.ELLIPTIC_CURVE_PARAMS['B']
q = settings.ELLIPTIC_CURVE_PARAMS['q']
G = settings.ELLIPTIC_CURVE_PARAMS['G']

class Point:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y

class CryptographyHelper:
    def __init__(self, t: int, n: int):
        self.t = t
        self.n = n
        self.random = SystemRandom()

    def generate_keys(self) -> Tuple[int, Point]:
        private_key = self.random.getrandbits(256)
        public_key = self.scalar_mult(private_key, Point(*G))
        return private_key, public_key

    def scalar_mult(self, k: int, P: Point) -> Point:
        N = P
        R = None
        while k:
            if k & 1:
                R = self.point_addition(R, N) if R else N
            N = self.point_addition(N, N)
            k >>= 1
        return R

    def point_addition(self, P: Point, Q: Point) -> Point:
        if P.x == Q.x and P.y == Q.y:
```

```
            m = (3 * P.x * P.x + A) * pow(2 * P.y, -1, q) % q
        else:
            m = (Q.y - P.y) * pow(Q.x - P.x, -1, q) % q
        x_r = (m * m - P.x - Q.x) % q
        y_r = (m * (P.x - x_r) - P.y) % q
        return Point(x_r, y_r)

    def aggregate_signatures(self, signatures: List[Point]) -> Point:
        agg_sig = signatures[0]
        for sig in signatures[1:]:
            agg_sig = self.point_addition(agg_sig, sig)
        return agg_sig

    def sign_message(self, private_key: int, message: str) -> Point:
        message_hash = int(sha256(message.encode()).hexdigest(), 16)
        signature = self.scalar_mult(private_key, Point(message_hash, message_hash))
        return signature

    def split_secret(self, secret: int) -> List[Tuple[int, int]]:
        coefficients = [secret] + [self.random.randint(0, q - 1) for _ in range(self.t - 1)]
        shares = [(i, self._polynomial(i, coefficients)) for i in range(1, self.n + 1)]
        return shares

    def _polynomial(self, x: int, coefficients: List[int]) -> int:
        return sum([coeff * (x ** i) for i, coeff in enumerate(coefficients)]) % q

    def reconstruct_secret(self, shares: List[Tuple[int, int]]) -> int:
        def _lagrange_interpolation(x: int, shares: List[Tuple[int, int]]) -> int:
            total = 0
            for i, (xi, yi) in enumerate(shares):
                li = 1
                for j, (xj, _) in enumerate(shares):
                    if i != j:
                        li *= (x - xj) * pow(xi - xj, -1, q) % q
                total += yi * li % q
            return total % q

        secret = _lagrange_interpolation(0, shares)
        return secret
```

Код класу який відповідає за запуск вузла та під’єднання до мережі вузлів:

```
logging.basicConfig(level=logging.INFO, format='[%(levelname)s] %(message)s')

class Command(BaseCommand):
    help = 'Connects to the peers and logs messages'

    def handle(self, *args, **options):
        # Register signal handler for Ctrl+C
        signal.signal(signal.SIGINT, self.signal_handler)
        self.stdout.write(self.style.SUCCESS("Starting the peer..."))
        logging.info("Starting the peer...")

        # Connect to peers concurrently
        asyncio.get_event_loop().run_until_complete(self.connect_to_peers())

    async def connect_to_peers(self):
        uri = settings.WEBSOCKET_SERVER_URI  # Update with your WebSocket server URL
        network_password = settings.NETWORK_PASSWORD

        async with websockets.connect(uri) as websocket:
            # Надсилання паролю мережі
            await websocket.send(json.dumps({"action": "authenticate", "password": network_password}))
            response = await websocket.recv()
            response_data = json.loads(response)

            if response_data.get("status") == "ok":
                self.stdout.write(self.style.SUCCESS("Successfully authenticated with the genesis peer"))
                logging.info("Successfully authenticated with the genesis peer")

                # Отримання списку наявних вузлів
                await websocket.send(json.dumps({"action": "get_nodes"}))
                nodes_response = await websocket.recv()
                nodes_data = json.loads(nodes_response)

                if nodes_data.get("status") == "ok":
                    nodes = nodes_data.get("nodes", [])
                    self.store_nodes_in_db(nodes)
                    self.stdout.write(self.style.SUCCESS(f"Received and stored {len(nodes)} nodes"))
                    logging.info(f"Received and stored {len(nodes)} nodes")
```

```
                # Імітація криптографічних операцій
                helper = CryptographyHelper(t=settings.THRESHOLD,
n=settings.NODES)

                # Імітація генерації ключів
                private_key, public_key = helper.generate_keys()
                logging.info(f"Node 1: Generated keys. Public Key: ({public_key.x},
{public_key.y})")

                # Імітація розбивки секрету
                shares = helper.split_secret(private_key)
                logging.info(f"Node 1: Split secret into shares: {shares}")

                # Імітація підписання повідомлення
                message = "Test transaction data"
                signature = helper.sign_message(private_key, message)
                logging.info(f"Node 1: Signed message. Signature: ({signature.x},
{signature.y})")

                # Імітація агрегації підписів
                signatures = [signature for _ in range(settings.THRESHOLD)]
                aggregated_signature = helper.aggregate_signatures(signatures)
                logging.info(f"Node 1: Aggregated signatures. Aggregated Signature:
({aggregated_signature.x}, {aggregated_signature.y})")

                # Продовження прослуховування нових команд
                await self.listen_for_commands(websocket)

        else:
            self.stdout.write(self.style.ERROR("Failed to authenticate with the genesis
peer"))
            logging.error("Failed to authenticate with the genesis peer")

    async def listen_for_commands(self, websocket):
        while True:
            try:
                message = await websocket.recv()
                command = json.loads(message)
                logging.info(f"Received command: {command}")

                # Обробка отриманих команд
                if command['action'] == 'new_transaction':
                    self.handle_new_transaction(command)
                elif command['action'] == 'new_signature':
                    self.handle_new_signature(command)
                # Додати інші обробники команд за необхідності
```

```
            except websockets.ConnectionClosed:
                logging.warning("Connection closed by the peer")
                break

    def handle_new_transaction(self, command):
        transaction_data = command['data']
        logging.info(f"Processing new transaction: {transaction_data}")
        # Логіка для обробки нової транзакції

    def handle_new_signature(self, command):
        signature_data = command['data']
        logging.info(f"Processing new signature: {signature_data}")
        # Логіка для обробки нової підпису

    def store_nodes_in_db(self, nodes):
        Node.objects.all().delete()  # Очищення попередніх записів
        for node in nodes:
            Node.objects.create(address=node['address'], public_key=node['public_key'])
        logging.info(f"Stored {len(nodes)} nodes in the database")

    def signal_handler(self, sig, frame):
        self.stdout.write(self.style.WARNING("Ctrl+C detected. Exiting..."))
        logging.warning("Ctrl+C detected. Exiting...")
        exit(0)
```

Код обробника вхідних повідомлень від інших вузлів:

```
logging.basicConfig(level=logging.INFO, format='[%(levelname)s] %(message)s')

class NodeConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.node_group_name = 'nodes_group'

        # Join room group
        await self.channel_layer.group_add(
            self.node_group_name,
            self.channel_name
        )

        await self.accept()
        logging.info(f"Node connected: {self.channel_name}")

    async def disconnect(self, close_code):
        # Leave room group
        await self.channel_layer.group_discard(
```

```
            self.node_group_name,
            self.channel_name
        )
        logging.info(f"Node disconnected: {self.channel_name}")

    async def receive(self, text_data):
        text_data_json = json.loads(text_data)
        action = text_data_json.get('action')

        if action == 'authenticate':
            await self.authenticate(text_data_json)
        elif action == 'get_nodes':
            await self.send_nodes_list()
        elif action == 'new_transaction':
            await self.handle_new_transaction(text_data_json)
        elif action == 'new_signature':
            await self.handle_new_signature(text_data_json)

    async def authenticate(self, data):
        password = data.get('password')
        if password == settings.NETWORK_PASSWORD:
            await self.send(text_data=json.dumps({"status": "ok"}))
            logging.info(f"Node authenticated: {self.channel_name}")
        else:
            await self.send(text_data=json.dumps({"status": "error", "message":
"Authentication failed"}))
            logging.warning(f"Node authentication failed: {self.channel_name}")

    async def send_nodes_list(self):
        nodes = await database_sync_to_async(Node.objects.all)()
        nodes_list = [{"address": node.address, "public_key": node.public_key} for
node in nodes]
        await self.send(text_data=json.dumps({"status": "ok", "nodes": nodes_list}))
        logging.info(f"Sent nodes list: {nodes_list}")

    async def handle_new_transaction(self, data):
        transaction_data = data.get('data')
        logging.info(f"Processing new transaction: {transaction_data}")
        # Логіка для обробки нової транзакції
        await self.channel_layer.group_send(
            self.node_group_name,
            {
                'type': 'broadcast_message',
                'message': json.dumps({"action": "new_transaction", "data":
transaction_data})
            }
        )
```

```
    async def handle_new_signature(self, data):
        signature_data = data.get('data')
        logging.info(f"Processing new signature: {signature_data}")
        # Логіка для обробки нової підпису
        await self.channel_layer.group_send(
            self.node_group_name,
            {
                'type': 'broadcast_message',
                'message': json.dumps({"action": "new_signature", "data": signature_data})
            }
        )

    async def broadcast_message(self, event):
        message = event['message']
        await self.send(text_data=message)
```

Код оголошення моделей для бази даних:

```
from django.db import models

class Node(models.Model):
    address = models.CharField(max_length=255, unique=True)
    public_key = models.CharField(max_length=255)

    def __str__(self):
        return self.address

class KeyShare(models.Model):
    node = models.ForeignKey(Node, on_delete=models.CASCADE)
    share = models.TextField()  # збереження частки ключа у вигляді тексту

    def __str__(self):
        return f"Share for node {self.node.address}"

class LocalNodeInfo(models.Model):
    address = models.CharField(max_length=255, unique=True)
    private_key = models.TextField()
    public_key = models.CharField(max_length=255)

    def __str__(self):
        return self.address

class SystemLog(models.Model):
    timestamp = models.DateTimeField(auto_now_add=True)
```

```
    log_level = models.CharField(max_length=50)
    message = models.TextField()

    def __str__(self):
        return f"{self.timestamp} - {self.log_level} - {self.message}"
```

Код методу взаємодії з Ethereum вузлом:

```
@staticmethod
    def send_signed_transaction(private_key, to_address, value, gas, gas_price, nonce,
data=""):
        web3 = Web3(Web3.HTTPProvider(settings.ETHEREUM_NODE_URL))

        # Створення та підписання транзакції
        transaction = {
            'to': to_address,
            'value': web3.toWei(value, 'ether'),
            'gas': gas,
            'gasPrice': web3.toWei(gas_price, 'gwei'),
            'nonce': nonce,
            'data': data,
        }

        signed_txn = CryptographyHelper.sign_message(transaction, private_key)
        tx_hash = web3.eth.sendRawTransaction(signed_txn.rawTransaction)

        return web3.toHex(tx_hash)
```

Код API інтерфейсу:

```
# nodes/views.py
class PublicKeyView(APIView):
    def get(self, request):
        try:
            local_node_info = LocalNodeInfo.objects.first()
            if not local_node_info:
                return Response({"error": "Local node information not found."},
status=status.HTTP_404_NOT_FOUND)

            return Response({"public_key": local_node_info.public_key},
status=status.HTTP_200_OK)
        except Exception as e:
            return Response({"error": str(e)},
```

```
status=status.HTTP_500_INTERNAL_SERVER_ERROR)


class SignTransactionView(APIView):
    def post(self, request):
        try:
            local_node_info = LocalNodeInfo.objects.first()
            if not local_node_info:
                return Response({"error": "Local node information not found."},
status=status.HTTP_404_NOT_FOUND)

            private_key = local_node_info.private_key
            to_address = request.data.get('to_address')
            value = request.data.get('value')
            gas = request.data.get('gas', 21000)  # стандартний ліміт газу для простих
транзакцій
            gas_price = request.data.get('gas_price')
            nonce = request.data.get('nonce')
            data = request.data.get('data', '')

            if not to_address or not value or not gas_price or nonce is None:
                return Response({"error": "Missing required transaction parameters."},
                                status=status.HTTP_400_BAD_REQUEST)

            txn_hash = CryptographyHelper.send_signed_transaction(private_key,
to_address, value, gas, gas_price, nonce,
                                                  data)

            return Response({"transaction_hash": txn_hash},
status=status.HTTP_200_OK)
        except Exception as e:
            return Response({"error": str(e)},
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Філюка Євгена Вікторовича
ПІБ здобувача вищої освіти

Студент ФІТ, 4 курсу, групи КБ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

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

06.06.2024
дата

підпис

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Філюк Євген Вікторович

Тема Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 4; кількість сторінок записки 64.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена система конфіденційності та безпеки криптоактивів на основі технології Multi-Part Computation. Ця система має активний захист від витоку приватного ключа, що є розподіленим між учасниками системи. У процесі проєктування були розроблені такі компоненти: розподілена генерація пар ключів приватного та публічного, агрегація частково підписаних хешів транзакції, система логування подій на вузлі прогармного забезпечення системи. Крім того, надані рекомендації щодо глобального розгортання системи в мережі та локального тестування системи.

2. Висновок про відповідність кваліфікаційної роботи завданню.У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині. Було дотримано усіх вимог щодо виконання

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У кваліфікаційній роботі була виконана низка завдань, таких як наведення загальної характеристики задачі, визначення проблематики, розглянуто предмет та методи дослідження. Виконанні задачі посприяли створенню рішення системи керування криптоактивів на основі багатосторонніх обчислень. У першому розділі проведено аналіз предметної області, розглянуто ріхні рішення щодо керування криптоактивами та способами їх захисту. У другому розділі, на основі результатів аналізу та визначеного завдання, було спроєктовано протокол багатосторонніх обчислень та архітектуру взаємодії вузлів програмного забезпечення системи. У третьому розділі описані практичні аспекти розробки, реалізації компонентів системи та протоколу захисту криптоактивів. Також наведено опис глобального розгортання в мережі та локального тестування системи з виконанням дійсної криптографічної транзакції у мережі Ethereum. Система рохроблена з використанням актуальних технологій та засобів.

4. Позитивні сторони роботи Розроблена система керування криптоактивів на основі багатосторонніх обчислень є акутальною та цікавою темою, що відповідає потребами блокчейн індустрії та безпечних переказів активів. Тематика роботи має потенціал для подальшого розвитку та вдосконалення системи.

5. Негативні сторони роботи У системі не передбачено резервне копіювання розподілених ключів на випадок втрати доступу до вузлів системи, що є надзвичайно актуальним в сучасних умовах, тому за відсутності більшості частин ключа, доступ до активів буде втрачено. Для системи не вистачає графічного інтерфейсу та документації, тому користувачі з недостатнім досвідом не зможуть швидко та легко скористатися системою.

6. Оцінка графічного оформлення та пояснювальної записки роботи. Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. У цілому графічне оформлення є чітким та якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження У переліку використаних джерел наявні посилання на популярні ресурси, такі як Вікіпедія. Такі джерела не рекомендовано використовувати при написанні кваліфікаційних робіт

9. Оцінка кваліфікаційної роботи Ураховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Бойко Юлій Миколайович,
доктор технічних наук, професор кафедри ТМІТ

«19» січня 2024. (підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система конфіденційності та безпеки крриптоактивів на сонові технології Multi-Party Computation

Автор: Філюк Євген Вікторович

Спеціальність: 125 – Кібербезпека

Освітня програма: освітньо-професійна

Науковий керівник: Джулій Володимир Миколайович, канд. техн. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 94,49%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про систему забезпечення академічної доброчесності у ХНУ (https://khmnu.edu.ua/wp-content/uploads/normatyvni-dokumenty/polozhennya/pro-systemu-zabezpechennya-akademichnoyi-dobrochesnosti.pdf, Додаток В) кваліфікаційна робота, виконана за , освітньо-професійною програмою, кількісні показники рівня унікальності тексту у відсотках до загального обсягу матеріалу в якій складає 75-100 %, визнається роботою з високою унікальністю тексту: «Текст вважається унікальним і не потребує додаткових дій щодо запобігання неправомірним запозиченням».

Керівник роботи _____ Володимир ДЖУЛІЙ

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

Fri Jun 14 20:50:17 EEST 2024, Мостовий Сергій Володимирович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

	Документ		Сумарний збіг по Базі Даних	
ID: 130619 Назва: Система конфіденційності та безпеки криптоактивів на основі технології Multi-Party Computation Додано в БД: 2024-06-14 Автора: Філюк Є.В. Керівники: Джулій В.М, Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	64490	1001	962 (1%)	14 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра кібербезпеки

Дата перевірки:
14.06.2024 22:25:11 EEST

Дата звіту:
14.06.2024 22:29:27 EEST

ID перевірки:
1016361808

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100008300

Назва документа: Філюк Кб-20-1 на плагіат

Кількість сторінок: 60 **Кількість слів: 9909** **Кількість символів: 78917** **Розмір файлу: 3.11 MB** ID **файлу: 1016166831**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.51% Схожість

Найбільша схожість: 1.11% з Інтернет-джерелом (https://academy.binance.com/uk/articles/what-is-blockchain-and-how.

5.15% Джерела з Інтернету	252	**Сторінка** 62
0.88% Джерела з Бібліотеки	49	**Сторінка** 63

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0% Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4
Підозріле форматування	12 сторінок