КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Завідувач кафедри
Терещенко В.М. _____
(підпис)

«__»_____20__р.

Кваліфікаційна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою "Інформатика"
спеціальності 122 "Комп'ютерні науки"
на тему:

АТОМАРНИЙ ОБМІН КРИПТОВАЛЮТ

Виконала студентка 4 курсу
Іваницька Єлизавета Єгорівна _____
(підпис)

Науковий керівник:
професор, доктор фіз.-мат. наук
Анісімов Анатолій Васильович _____
(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи: 43 сторінки, 18 ілюстрацій, 17 джерел посилань.

АТОМАРНИЙ ОБМІН КРИПТОВАЛЮТ, СМАРТ-КОНТРАКТИ, КРИПТОВАЛЮТА, БЛОКЧЕЙН-ТЕХНОЛОГІЇ, СМАРТ-КОНТРАКТИ З ЧАСОВОЮ ЗАТРИМКОЮ.

Об’єктом роботи є проведення атомарного обміну криптовалюти. Предметом роботи є програмний засіб для виконання атомарного обміну криптовалюти.

Метою роботи є дослідження засобів виконання атомарного обміну криптовалюти та створення програмного забезпечення для проведення атомарного обміну між токенами різних блокчейнів.

Методи розробки: дослідження існуючих підходів, розробка програмного забезпечення. Інструменти розробки: операційна система – Ubuntu 20.04, середовище програмування Visual Studio Code, компоненти системи реалізовані мовою загального призначення Rust, а також мовою написання смарт-контрактів Bitcoin Script.

Результати роботи: виконано загальний огляд існуючих підходів виконання атомарного обміну криптовалюти, досліджено методи роботи зі смарт-контрактами в різних блокчейн-мережах, реалізовано необхідні компоненти для проведення атомарного обміну між криптовалютами.

Запропонований у даній роботі програмний продукт може використовуватися для проведення атомарного обміну між криптовалютою у мережах Bitcoin та NEAR Protocol.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ВИЗНАЧЕНЬ

HTLC(Hashed Time-Locked Contract) – хешований смарт-контракт з часовою затримкою;

P2P network(peer-to-peer network) - однорангова мережа;

POW(Proof-of-work) - доказ роботи, один із механізмів розподіленого консенсусу;

DApps - децентралізовані додатки;

PoS(Proof of Stake) – доказ долі володіння, метод захисту в криптовалюті;

NFT(Non-fungible tokens) – вид криптографічних токенів, кожен екземпляр якого є унікальним;

TPS(Transactions per second) – кількість транзакцій оброблених за секунду;

CLI(Command line interface) – текстовий інтерфейс, де інструкції вводяться лише введенням текстових рядків.

ВСТУП

Оцінка сучасного стану об’єкта розробки. За останні кілька років блокчейн-технології та їх застосування у криптовалюті стають дедалі популярнішими – станом на 2018 рік кількість різних криптовалют була понад 1600, і ця кількість постійно зростає. У зв’язку з цим стрімко зростає попит на дослідження безпечних методів обміну токенами різних блокчейнів.

Для проведення атомарних обмінів криптовалюти почали з’являтися децентралізовані біржі, наприклад, біржа Uniswap [1]. Однак, на сьогоднішній день, програмні продукти для проведення атомарного обміну криптовалют дозволяють проводити децентралізований обмін для обмеженої кількості різних токенів. Наприклад, для токенів, що мають обмежену підтримку смарт-контрактів підтримка атомарних обмінів не є розповсюдженою.

Актуальність роботи та підстави для її виконання. На сьогоднішній день відома велика кількість різних криптовалют, що використовують різні види блокчейну. Обмін між звичайними валютами зазвичай виконується через посередника, що гарантує виконання вимог сторонами угоди, та потребую довіри обох сторін. Тоді як при обміну криптовалюти немає потреби у централізованих установах для виконання операцій, що є значною перевагою над звичайною валютою.

Блокчейн-технології надають можливість проводити безпечні операції з внутрішньою валютою блокчейну, але для гарантування виконання угод, що вносять зміни в кілька різних мереж водночас необхідно використовувати додаткові механізми захисту. Для проведення обміну токенами різних криптовалют, за відсутності третьої сторони, для уникнення ризику втрати коштів, використовуються технології атомарного обміну криптовалют.

Таким чином, застосування даних технологій дозволяє розширити сценарії використання криптовалюти, зберігаючи існуючі переваги, як, наприклад, відсутність потреби у третій особі - гаранти для виконання угод.

Мета й завдання роботи. Метою дипломної роботи є розробка програмного засобу для проведення атомарного обміну між криптовалютами різних блокчейнів.

Для досягнення цієї мети поставлено такі завдання:

- поглибити знання у блокчейн-технологіях;
- дослідити існуючі системи та технології для проведення обміну різних криптовалют;
- розробити та описати алгоритм проведення атомарного обміну криптовалют;
- поглибити знання з написання смарт-контрактів;
- розробити смарт-контракт для реалізації алгоритму атомарного обміну криптовалюти;
- реалізувати необхідні додаткові операції для мережі Bitcoin;
- реалізувати необхідні додаткові операції для мережі NEAR Protocol;
- розробити інтерактивну систему, що дозволятиме двом користувачам обміняти токени різних блокчейнів;
- протестувати розроблене програмне забезпечення, з використанням локальної симуляції мережі;
- навести приклади застосування отриманого програмного забезпечення.

Об'єкт, методи й засоби розроблення. Об’єктом дипломної роботи є процес організації атомарного обміну криптовалют. Предметом роботи є програмний засіб для проведення атомарного обміну криптовалют, який підтримує роботу з токенами мереж NEAR Protocol(NEAR) та Bitcoin(BTC).

Реалізації програмного засобу передувала проектування системи та розробка смарт-контрактів. Основою для розробки стало вивчення існуючих методів та алгоритмів проведення атомарного обміну криптовалют, вивчення особливостей обраних мереж та механізмів розробки та виконання розумних контрактів в загальному та у мережах Bitcoin та NEAR Protocol.

Для розробки смарт контрактів, було використано мови Rust та Bitcoin Script.

Програмне забезпечення було розроблено на операційній системі Ubuntu 20.04. Для створення проекту було використано середовище програмування Visual Code 2022.

Можливі сфери застосування. Розроблений програмний продукт може застосовуватись для проведення атомарного обміну між криптовалютами блокчейнів Bitcoin та NEAR Protocol. Програмне забезпечення є легко розширювальним для підтримки інших блокчейнів, та може стати основою для створення децентралізованої біржі.

РОЗДІЛ 1. БЛОКЧЕЙН-ТЕХНОЛОГІЇ ТА КРИПТОВАЛЮТА

Вперше термін “блокчейн” з’явився як назва повністю реплікованої розподіленої бази даних, реалізованої в системі Bitcoin, через що блокчейн часто зіставляють з реєстром транзакцій у різних криптовалютах. На сьогоднішній день блокчейн застосовується у різних сферах, таких як фінансові операції, ідентифікація користувача чи створення технологій кібербезпеки. Але все ж таки в першу чергу блокчейн-технології актуальні для банківських установ та державних організацій.

1.1. Загальний огляд

Блокчейн являє собою побудований за певним набором правил безперервний послідовний ланцюжок блоків, що містять деяку інформацію. Зв’язок між блоками забезпечується таким чином: кожен блок містить свою особисту хеш-суму та хеш-суму попереднього блоку. Зміна будь-якої інформації в блоці змінить також його хеш-суму. Для відповідності правилам побудови ланцюжка, зміни хеш-суми потрібно буде записати в наступний блок, що спричинить зміни хеш-суми безпосередньо наступного блоку. При цьому попередні блоки залишаться без змін [2].

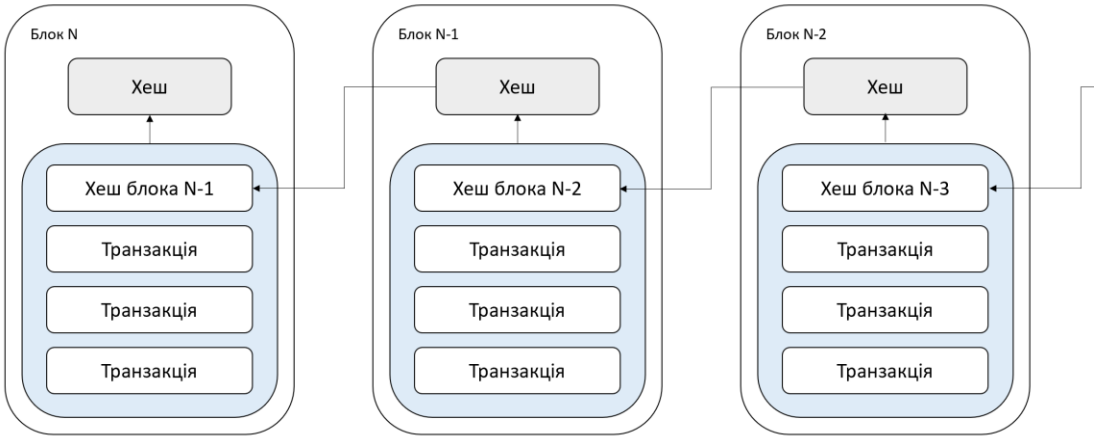

Рисунок 1.1 Структура блокчейна

Кожен такий блок містить хеш-суму з попереднього блоку та корисне навантаження, в якості якого може бути інформація про транзакції. Блокчейн є реєстром записів, що постійно поповнюється.

Транзакції в блоках зберігаються у вигляді структури даних, що має назву дерево Меркла. Тобто дерево Меркла виступає підсумком усіх транзакцій у блоці. Кожна транзакція в блоці унікально хешується за допомогою хеш-функцій, таких як MD5, BLAKE2, SHA-1, SHA-256, щоб створити цифровий відбиток усього набору транзакцій. Кожна пара захешованих транзакцій ща раз хешується разом хеш-функцією, це продовжується до тих пір, доки не отримаємо один хеш для всього блоку.

За структурою дерево Меркла є одним з типів бінарного дерева, де хеші транзакційних даних у нижньому рядку називаються «листовими вузлами», проміжні хеші – «гілками», а хеш у верхній частині – «кореневим». Дерево Меркла деколи називають деревом хешів. Кожен окремий блок в блокчейн має один корінь Меркла [3].

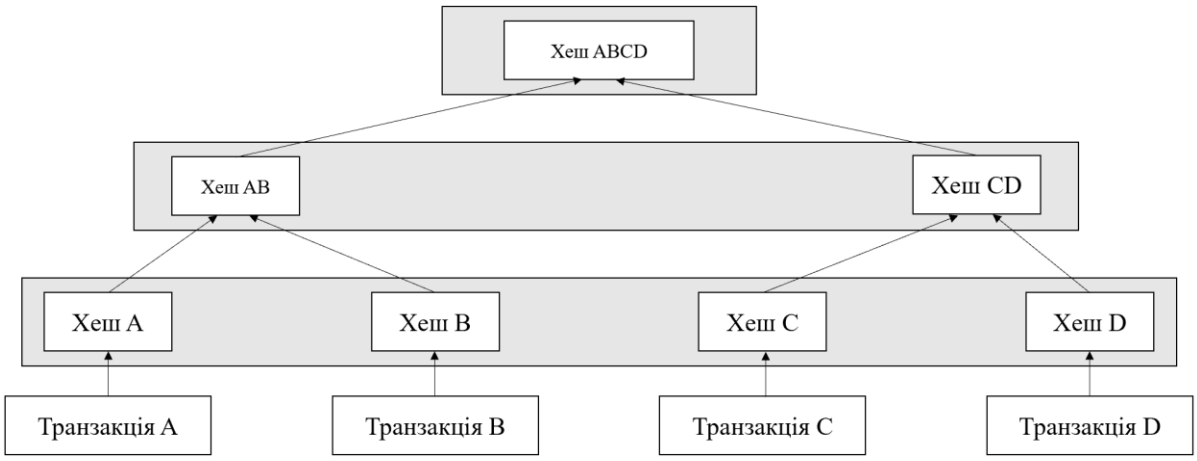

Рисунок 1.2 Дерево Меркла

Кожен такий блок складається із заголовка та тіла. Заголовок блоку містить корінь дерева Меркла, мітку часу, номер серії блоку, ціль складності,

одноразовий код (Nonce) та хеш попереднього блоку. Тіло блоку містить у собі усі транзакції, що підтвердженні всередині блоку.

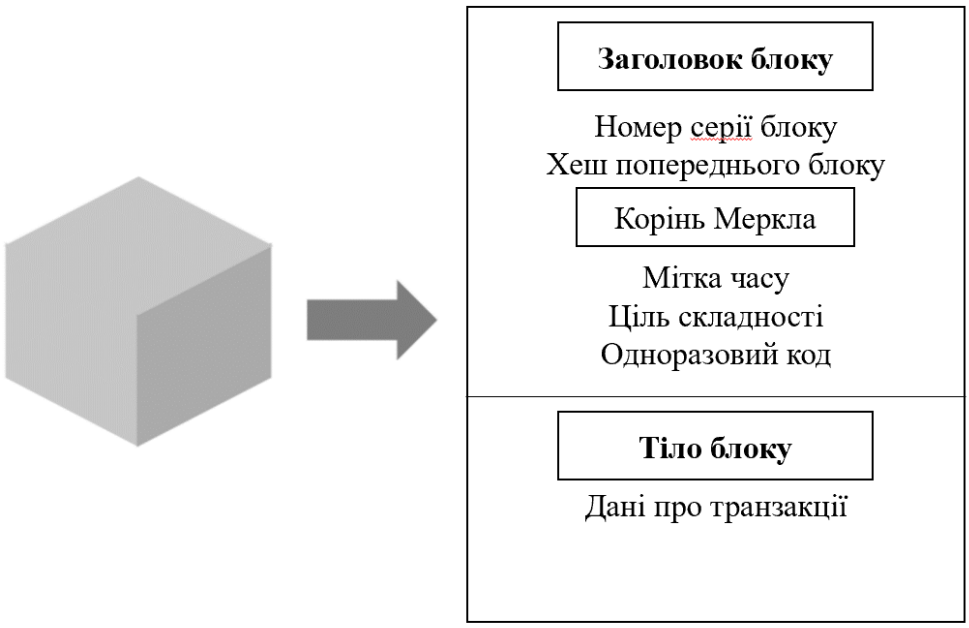

Рисунок 1.3 Структура блоку

Основні переваги дерева Меркла:

1. Оскільки кожен блок має унікальне хеш-значення, обчислене з кореня Меркла та містить хеш попереднього блоку, тоді якщо в деякій транзакції відбулася зміна, то змінюється хеш цієї транзакції. Зміна хешу однієї транзакції призводить до зміни кореня Меркла і таким чином цей блок стає недійсним. Далі це призведе до зміни хешу наступного блоку і т. д., що робить решту блокчейну недійсною. Тому дерево Меркла робить незмінний запис транзакцій у блоці.
2. Блокчейн зазвичай складається з сотень тисяч блоків, і кожен блок може містити до кількох тисяч транзакцій. Отже, пам'ять та обчислювальна потужність є великими проблемами під час перевірки даних. Якби в

блокчейн не було концепції дерева Меркла, то кожен вузол мережі мав би зберігати повну копію кожної окремої транзакції, та під час підтвердження транзакції вузол повинен був порівнювати кожен запис рядок за рядком, що потребує надзвичайно великої потужності комп'ютера. Дерево Меркла вирішує цю проблему. Хешування усіх записів транзакцій відокремлює підтвердження даних від самих даних. Користувачі можуть перевіряти окремі блоки, а також порівнювати транзакції за допомогою порівняння хешів. Таким чином, зменшується обчислювальна потужність та зменшується необхідна кількість пом'яті для перевірки транзакцій.

3. Дерево Меркла запобігає можливість зловмисника змінювати записи транзакцій. Наприклад, якщо хтось спробує підмінити запис у блокчейні, щоб він мав більше криптовалюти, ніж насправді є, такий хибний запис не відповідатиме решті хешів у дереві Меркла.

1.2. Використання блокчейну у криптовалютах

Криптовалюта – це віртуальна (або цифрова) валюта, яка є засобом обміну. Вона подібна до реальної валюти, за виключенням того, що криптовалюта не має жодного фізичного втілення, та використовує криптографію для роботи.

Оскільки криптовалюти працюють незалежно та децентралізовано, без будь-якого централізованого органу, нові одиниці криптовалюти можна додавати лише після виконання певних вимог. Наприклад, у криптовалюті Bitcoin тільки після додавання блоку до блокчейна, майнер отримає винагороду біткоїнами, і це єдиний спосіб генерувати нові біткоїни.

Уявімо собі ситуацію, у якій одна людина має намір відправити гроші іншій людині на її рахунок. З'являються декілька варіантів, чому така операція може бути не успішна, зокрема:

1. У фінансової установи виникнуть технічні проблеми, наприклад, її системи не працюють належним чином.
2. Обліковий запис відправника або отримувача міг бути зламаний, наприклад, могла статися атака з відмовою в обслуговуванні або викрадення особистих даних.
3. Перевищено ліміти переказів для облікового запису відправника чи отримувача.

Тобто, більшість випадків збою у валютних операціях трапляються саме на стороні банківської установи.

Тепер уявімо подібну транзакцію між двома людьми, але замість звичайної валюти вони користуються криптовалютою. З'являється сповіщення з таким запитанням, чи впевнена людина, що вона бажає передати деяку кількість біткоїнів. Якщо так, відбувається обробка: система аутентифікує особу користувача, перевіряє, чи має користувач необхідний баланс для здійснення цієї транзакції тощо. Після цього платіж перераховується, а гроші потрапляють на рахунок одержувача. Все це відбувається за лічені хвилини.

Таким чином, криптовалюта усуває більшість проблем сучасної банківської системи: немає обмежень на кошти, які можете переказати, рахунки користувачів неможливо зламати, і немає центральної точки збою.

Також одна з значних переваг криптовалюти полягає в тому, що на відмінну від комісії за переказ між банківськими рахунками, вартість транзакцій низька або зовсім нульова. Транзакції можуть здійснюватися у будь-який час, також немає обмежень на покупку чи зняття коштів. Будь-хто може вільно користуватися криптовалютою, на відмінну від відкриття банківського рахунку, для якого потрібні документи та інші довідки.

Міжнародні транзакції з криптовалютою також набагато швидші, ніж банківські перекази. З криптовалютами транзакції займають лише кілька хвилин або навіть секунд, у той час як банківські перекази займають близько півдня.

1.3. Загальні відомості про криптовалюту Bitcoin

Мережа Bitcoin - це однорангова (peer-to-peer) платіжна мережа, яка працює за криптографічним протоколом. Користувачі можуть отримувати та надсилати біткоїни, грошові одиниці, шляхом трансляції повідомлень із цифровим підписом у мережу за допомогою спеціального додатку, що має вигляд гаманця криптовалюти Bitcoin. Транзакції записуються у децентралізовану базу даних, відому як блокчейн, проходячи верифікацію за допомогою системи proof-of-work [4].

Розробник мережі Bitcoin, Сатоші Накамото, заявляв, що розробка та кодування біткоїна почалися в 2007 році. Проект був запущений у 2009 році як програмне забезпечення з відкритим кодом.

Мережа Bitcoin вимагає мінімальної структури для спільного використання транзакцій. Достатньо спеціальної децентралізованої мережі волонтерів, що представляють собою так званні вузли, вони можуть залишати мережу та знову приєднуватися до неї за бажанням. Після повторного підключення вузол завантажує та перевіряє нові блоки з інших вузлів, щоб мати повну свою локальну копію блокчейна.

Найбільш поширеною верифікацією блоку є proof-of-work (доказ роботи). Для того, щоб наступний блок можна було додати до блокчейну, необхідно підібрати такий префікс, щоб хеш-сума блоку не перевищувала наперед задане значення(так звана “складність блоку”). Дане значення змінюється в

залежності від швидкості генерації префіксів для стабілізації частоти додавання нових блоків до блокчейну.

Proof-of-work являє собою пошук такого значення, яке при хешуванні, наприклад за допомогою SHA-256, починається з певної кількості нульових бітів. Середня необхідна кількість операцій для пошуку такого значення є експоненційною відносно кількості необхідних нульових бітів.

Для мережі Bitcoin proof-of-work реалізовано наступним чином: у блоці поступово збільшується одноразове число (nonce), доки не буде знайдено значення, яке задовольняє умову - необхідну кількість нульових бітів у хеш-сумі. Після верифікації блоку, його не можна змінити без повторної перевірки. Оскільки наступні блоки об'єднуються в спільний ланцюжок, робота по зміні певного блоку вимагатиме повторного виконання всіх блоків після нього [5].

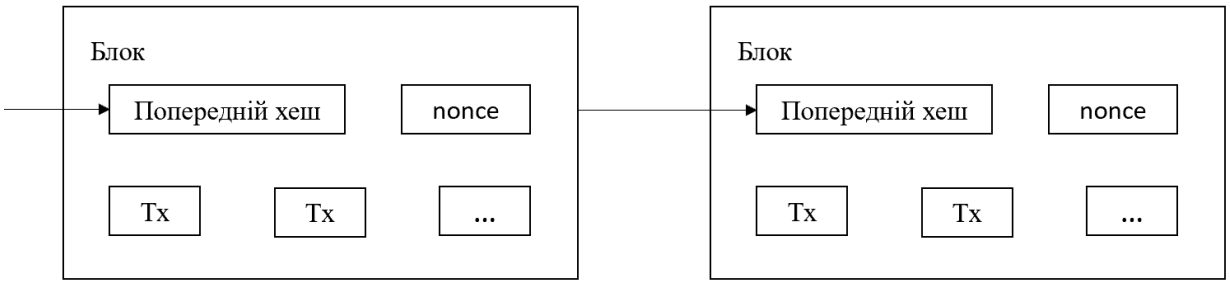

Рисунок 1.4 Структура блоку з nonce

Наведемо основні принципи роботи мережі Bitcoin:

1. Нові транзакції передаються на всі вузли
2. Кожен вузол компонує нові транзакції у блок
3. Кожен вузол працює над пошуком nonce для proof-of-work для свого блоку
4. Коли вузол знаходить потрібне значення, він передає блок усім вузлам
5. Вузли приймають блок, тільки якщо всі транзакції в ньому валідні

6. Вузли виражають своє прийняття блоку, працюючи над створенням наступного блоку в ланцюжку, використовуючи хеш прийнятого блоку як попередній хеш

Завжди вважається правильним найдовший ланцюжок у вузлах, вузли будуть продовжувати працювати над його розширенням [5].

Розглянемо випадки можливих потенційних атак на мережу Bitcoin та функції, які захищають її від таких атак [6].

Несанкціоновані витрати. Перешкоджання несанкціонованих витрат досягається використанням криптографії із публічним та приватним ключем. Наприклад, коли Аліса надсилає біткоїн Бобу, Боб стає новим власником біткоїна. Єва, спостерігаючи за транзакцією, хоче витратити щойно отриманий Бобом біткоїн, але вона не може це зробити, оскільки не знає приватного ключа Боба для підписання транзакції.

Подвійні витрати. Особливою проблемою, яку має вирішувати система інтернет-платежів, є подвійні витрати, тобто коли користувач відправляє одну і ту ж монету двом або більше різним одержувачам. Приклад такої проблеми: якби Єва відправила біткоїн Алісі, а пізніше надіслала той самий біткоїн Бобу. Мережа біткоїн захищає від подвійних витрат, записуючи всі перекази біткоїнів у реєстр (блокчейн), який є доступним для перегляду усім користувачам, і гарантує, що всі передані біткоїни не були витрачені раніше.

Стан гонитви. Припустимо, що Єва пропонує заплатити Алісі біткоїн в обмін на товари і підпише відповідну транзакцію, все одно є ймовірність того, що вона також створить іншу транзакцію, одночасно відправивши той самий біткоїн Бобу. За правилами мережа приймає тільки одну з транзакцій. Це називається атакою перегонів, оскільки існує гонка, яка транзакція буде прийнята першою. Аліса ж у свою чергу може зменшити ризик такої атаки,

поставивши умову, що вона не доставлятиме товари, до тих пір коли платіж Єви Алісі не з’явиться в блокчейні.

Модифікація історії. Кожен наступний блок, який додається до блокчейну, починаючи з блоку, що містить певну транзакцію, називається підтвердженням цієї транзакції. В ідеалі продавці та служби, які отримують оплату в біткоїнах, мають почекати хоча б одного підтвердження, яке буде розповсюджено по мережі, перш ніж вважати, що платіж проведено успішно. Чим більше підтверджень чекає продавець, тим складніше зловмиснику успішно скасувати транзакцію в блокчейні, якщо звісно ж зловмисник не контролює більше половини загальної потужності мережі, але в цьому випадку це вже називається атакою 51%.

1.4. Смарт-контракти в мережі Bitcoin

Смарт-контракт – це комп’ютерна програма або протокол транзакції, що призначений для автоматичного виконання, контролю або документування юридично релевантних подій і дій відповідно до умов контракту або угоди. Основною ціллю смарт-контрактів є зменшення потреби в посередниках, зменшення витрат на арбітраж і примусове виконання, а також зменшення шкідливих і випадкових винятків [7].

Смарт-контракт у термінах криптовалюти представляє собою цифрову угода, яка автоматично виконується на основі попередньо визначених критеріїв. Наприклад, критерієм смарт-контракту може бути таке, що біткоїн повинен автоматично надсилатися від одного користувача до іншого після певної затримки.

Смарт-контракти можуть як дуже простими, та включати в себе один простий критерій, так і надзвичайно складними і включати в себе багато різних критеріїв.

Мережа Bitcoin підтримує широке різноманіття смарт-контрактів, використовуючи потужну мову сценаріїв, що має назву Script. Script надає можливість користувачам встановлювати певні критерії витрачання своїх біткоїнів, а транзакції з біткоїнами блокують певну кількість біткоїнів потрібних для цих сценаріїв. Користувач повинен задовольняти цим критеріям, щоб витратити біткоїн, заблокований у сценарії. Таким чином, всі транзакції з біткоїнами являються по суті смарт-контрактами.

Критерії витрат біткоїна називаються сценарієм scriptPubKey або сценарієм блокування. Дані та сценарій, що задовольняють заданим критеріям, називаються ScriptSig або ScriptWitness, це залежить від того, використовується вхідний код SegWit чи ні.

Мова сценаріїв Script виявилася корисною для мережі Bitcoin, оскільки вона не є повною за Тьюрингом, тобто не допускає логічних циклів, а отже не можливо створити нескінчений цикл. Ця функція захищає мережу Bitcoin від атак відмови в обслуговуванні (Denial of Service attacks), які можуть уразити інші мережі криптовалют.

Одним з критеріїв виконання транзакції може бути блокування витрати біткоїна за часом (time locked), тобто вони доступні лише через певний час. Як приклад, сценарій може вимагати три підписи, щоб витратити біткоїн до настання певного часу, після чого потрібен лише один підпис. Це надає можливість робить альтернативні варіанти виконання транзакцій.

1.5. Загальні відомості про NEAR Protocol

NEAR Protocol – це блокчейн, що використовує новітню технологію шардингу, що має назву Nightshade для досягнення масштабованості. NEAR Protocol був запущений в 2020 як децентралізована хмарна інфраструктура для розміщення децентралізованих додатків (DApps). Порівнюючи з іншими блокчейнами NEAR більш простий у використанні, більш дешевий та

швидкий. Також кожен користувач може зробити свій внесок у його розвиток, оскільки NEAR має відкритий вихідний код.

NEAR працює з використанням смарт-контрактів та алгоритму Proof of Stake (PoS), який відповідає за безпеку мережі.

Платформа NEAR пропонує користувачу широкий спектр інструментів та мов програмування, а також набір смарт-контрактів з кросчейн-функціями, за допомогою яких розробники можуть створювати DApps. На платформі діє спрощений процес реєстрації, також використовуються зрозумілі імена акаунтів замість криптографічних адрес гаманців.

Для можливості конкурувати з іншими блокчейнами з підтримкою смарт-контрактів, NEAR реалізує певні функції для підвищення продуктивності. Головною технологією NEAR є Nightshade, яка використовує шардинг для ефективнішої обробки даних. Шардинг відповідає за розподіл завданнь з обробки транзакцій між безліччю нод-валідаторів. Таким чином, кожна нода відповідає за обробку лише частини транзакцій мережі, що дозволяє збільшити кількість транзакцій оброблених за секунду (TPS). Теоретично Nightshade дозволяє NEAR обробляти мільйони транзакцій за одну секунду, при цьому не знижуючи продуктивність [8].

Також є можливість взаємодії NEAR з Ethereum через Rainbow Bridge – надійний міст, який дозволяє користувачам передавати активи, такі як токени ERC20 та NFT, між мережами Ethereum та NEAR. Також навіть дозволяється взаємодіяти зі смарт-контрактами та DApps по обидва боки, використовуючи той самий Rainbow Bridge.

Отже, NEAR особливий тим, що він неймовірно швидкий, здатний обробляти мільйони транзакцій за секунду (tps) і майже миттєво досягає завершення транзакції завдяки одно-секундному створенню блоку.

Ще одна особливість NEAR полягає у тому, що для його використання не обов'язково мати знання про блокчейн. У NEAR представлена дуже зручна система облікових записів та ключів доступу до них. Звичайні користувачі зможуть отримати доступ до DApps, побудованих на NEAR, пройшовши звичний процес реєстрації, тому користувачі можуть навіть не здогадуватися, що мають справу з блокчейном.

1.6 Смарт-контракти в NEAR Protocol

Веб-додатки NEAR мають дві частини у вигляді Front-end та Back-end [9]:

1. Смарт-контракти : серверна частина програми, яка запускає код і зберігає дані в блокчейні. Смарт-контракти керують зберіганням та модифікацією даних у ланцюжку.
2. Інтерфейс для взаємодії за смарт-контрактами : подібно до хмарного API, надається можливість взаємодіяти зі смарт-контрактами, використовуючи near-api-js та доступний код програми.

Головним аспектом API платформи NEAR є інтерфейс JSON-RPC. Near-api-js представляє інтерфейс RPC та зручні функції та записує примітиви NEAR у формі об'єктів JavaScript. Near-api-js використовується як основний інтерфейс для взаємодії з NEAR Protocol під час написання JavaScript (на стороні клієнта чи сервера).

Розумні контракти для NEAR повинні бути скомпільовані в WebAssembly. Розробникам доступна можливість використовувати одну з двох технологій для написання та складання смарт-контрактів на платформі NEAR [10]:

1. Rust: потужна мова програмування, використовується для написання цінних смарт-контрактів, більше підходить для фінансових додатків
2. AssemblyScript: це група помічників, які допомагають смарт-контрактам виглядати так само, як TypeScript, під час компіляції їх у WebAssembly. На даний момент не рекомендується використовувати

при розробці фінансових децентралізованих додатків через новизну мови програмування.

РОЗДІЛ 2. ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ

2.1. Хешовані смарт-контракти з часовою затримкою

Хешовані смарт-контракти з часовою затримкою (HTLC) - це клас платежів, з використанням часових блокуваннь та хеш-блокуваннь, тобто від одержувача платежу вимагається підтвердження отримання платежу до встановленого терміну, тобто створення криптографічного підтвердження платежу, інакше платіж не буде проведено і кошти повернуться відправнику[11].

Термін Hashed Time Lock Contract (HTLC) у криптовалюті відноситься до спеціальної функції, що використовується для створення розумних контрактів, які можуть змінювати канали платежів. Загалом функція HTLC використовується для здійснення транзакції між двома користувачами, обмеженої часом. Практично це означає, що одержувач транзакції HTLC повинен підтвердити платіж, надаючи криптографічне підтвердження протягом визначеного терміну. Якщо одержувач не виконає поставлені умови платежу за певний період часу, кошти будуть повернуті початковому відправнику.

HTLC відрізняється від стандартних криптовалютних операцій таким чином:

1. Hashlock – це захешована або криптографічно зашифрована версія відкритого ключа, згенерована особою, яка ініціює транзакцію, пов’язаний закритий ключ використовується для розблокування вихідного хешу.

   У HTLC ініціатор генерує та хешує ключ. Хеш зберігається в попередньому зображенні, яке розкривається під час остаточної транзакції. HTLC запрограмовані на закінчення терміну дії після

закінчення певного періоду або створення певної кількості блоків, створюючи відомі дату і час завершення [12].

2. Ще одним суттєвим елементом HTLC є блокування часу. Використовуються два різних блокування часу для встановлення часових обмежень для транзакцій, створених за допомогою HTLC, Перший – CheckLockTimeVerify (CLTV) – використовує часову базу для блокування, тобто часові обмеження жорстко закодовані.

   Другий – CheckSequenceVerify (CSV) – не залежить від часу, натомість використовується кількість згенерованих блоків для визначення, коли завершити транзакцію[12].

Одним з найпопулярніших випадків використання хешованих контрактів із блокуванням часу є Bitcoin Lightning Network. З впровадженням HTLC у платіжні канали можна передавати кошти від користувача до користувача через взаємопов'язані платіжні канали, при цьому не вимагаючи жодного рівня довіри. Такий процес називають мережевою маршрутизацією. Для прикладу, HTLC надає можливість Алісі надсилати свої кошти Бобу через інших учасників мережі (наприклад, Керол), Керол гарантовано не зможе перехопити кошти, завдяки використанню функції хеш-блокування та блокування часу.

2.2. Реалізація HTLC для мережі Bitcoin

У мережі Bitcoin хешовані смарт-контракти з часовою затримкою реалізуються наступним чином: як основний механізм, для імплементації смарт-контрактів, використовуються так званні блокуючі скрипти. Вони описують умови, які має втиконати отримувач у відмикаючому скрипті, тобто має довести право розпоряджатися заблокованими токенами.

Існує певний набір кодів операцій мови програмування Bitcoin Script. Наведемо деякі з них, що використовуються для реалізації смарт-контрактів, відповідно до документації [13]:

- OP_IF - OP_ELSE - OP_ENDIF – для розгалуження виконання скрипту;
- OP_SIZE - для перегляду значення верхівки стеку та додавання до стеку довжини зчитаного значення;
- OP_EQUALVERIFY – для вилучення двох верхніх значень зі стеку та порівняння їх. Виникає помилка у разі не співпадіння значень;
- OP_SHA256 – для вилучення верхнього значення зі стеку і додавання значення його хеш-суми, використовуючи хеш-функцію SHA256;
- OP_DUP – для додання до стеку копії верхнього значення зі стеку;
- OP_HASH160 – подібно до OP_SHA256, але використовується хеш-функція RIPEMD160 замість SHA256;
- OP_CHECKLOCKTIMEVERIFY – дістає значення з вершини стеку, яке вважається мінімальною позначкою часу, починаючи з якої транзакція вважається коректною. Викидається помилка в тому випадку, коли значення nLockTime в транзакції не є більшим ніж значення, отримане зі стеку;
- OP_DROP – для вилучення верхнього значення зі стеку;
- OP_CHECKSIG – інтерпретує два верхні значення стеку як публічний ключ та цифровий підпис. До стеку додається 1 за умови: значення підпису, отримане зі стеку, являється коректним підписом для даної хеш-функції, зробленим з використанням приватного ключа, що відповідає публічному ключу, отриманому зі стеку. Інакше до стеку додається 0.

- OP_TRUE, OP_FALSE – до стеку додається значення, що інтерпретується як логічна істина або хиба відповідно.

Наведемо приклад блокуючого скрипта контракту:

1. OP_IF
2. OP_SIZE
3. <довжина секретного значення>
4. OP_EQUALVERIFY
5. OP_SHA256
6. <хеш секретного значення>
7. OP_EQUALVERIFY
8. OP_DUP
9. OP_HASH160
10. <адреса отримувача>
11. OP_ELSE
12. <позначка часу>
13. OP_CHECKLOCKTIMEVERIFY
14. OP_DROP
15. OP_DUP
16. OP_HASH160
17. <адреса відправника>
18. OP_ENDIF
19. OP_EQUALVERIFY
20. OP_CHECKSIG

На першому кроці у наведеному вище скрипті виконується одна з гілок розгалуження, в залежності від значення даних у вершині стеку. Відповідно перевірка з використання образу хешу, що вказаний в скрипті, виконується у разі інтерпретації значення як істини. Інакше ж, при інтерпретації даних як хиба, виконується перевірка часового блоку. Але не залежно від розгалуження обов’язково виконується перевірка електронно-цифрового підпису транзакції. Перевірка поверне істинне значення лише у випадку, коли для створення підпису було використано приватний ключ, відповідний адресі отримувача, при розблокуванні за допомогою образу хешу. Відповідно, при розблокуванні після завершення часової затримки, необхідно використати приватний ключ, відповідний адресі відправника, для розблокування контракту.

Існують лише два ймовірні випадки розблокування коштів:

1. Кошти розблоковує одержувач з використанням секретного значення. У такому випадку розблоковуючий скрипт має наступний вигляд:
   1) <підпис>
   2) <публічний ключ>
   3) <кандидат секретного значення>
   4) OP_TRUE
2. Кошти розблоковує відправник, що можливо лише після завершення часового блокування. У такому випадку розблоковуючий скрипт має наступний вигляд:
   1) <підпис>
   2) <публічний ключ>
   3) OP_FALSE

Можемо зробити наступний висновок: отримувач має змогу розблокувати кошти лише при умові вказання правильного секретного значення та володіння приватним ключем, відповідним адресі отримувача. Отже, приведена реалізація задовольняє стандартам хешованих смарт-контрактів з часовою затримкою, описаних у розділі 1.

2.3. Реалізація HTLC для мережі NEAR

Хешовані смарт-контракти з часовою затримкою для мережі NEAR Protocol на практиці реалізуються за допомогою опису необхідного функціоналу, з використанням Rust API.

На відмінну від мережі Bitcoin, немає можливості безпосередньо створити блок з транзакцією, яка містить блокуючий скрипт, для блокування токену NEAR. Запропонуємо натомість використати інший механізм для реалізації хешованого смарт-контракту з часовою затримкою.

Використаємо додатково створений акаунт, у якому будуть знаходитися заблоковані токени та описані умови, при виконанні яких ці токені або відправляться далі отримувачу, або повернуться відправнику у разі не виконання поставлених умов. У цьому акаунті буде знаходитися список унікальних ідентифікаторів створених смарт-контрактів та відповідних їм HTLC структур, а також набір додаткових функцій, які відповідають за перевірку виконання умов та розпоряджаються заблокованими токенами.

Структура HTLC має наступний вигляд:

- creation-time – час створення смарт-контракту;
- lifetime – скільки часу контракт є дійсним, по його закінченню кошти повертаються відправнику;
- secret-hash – хеш секретної фрази для розблокування;
- amount – яка кількість токенів приймає участь у контракті;
- sender-address – адреса відправника;
- recipient-address – адреса отримувача;
- completed – чи виконаний успішно контракт;
- middle-acc – акаунт, на якому тимчасово зберігається контракт та токени, доки не будуть виконані умови або не закінчиться час життя контракту.

2.4. Теоретичні засади атомарного обміну криптовалюти

Атомарним називається обмін однієї криптовалюти на іншу без участі третіх осіб. Такий обмін або виконується успішно та нерозривно, або не виконується взагалі. Атомарний обмін можна здійснити, навіть якщо користувачі не довіряють один одному. Також гарантується, що один учасник не втратить монети, навіть при умові, що другий хоче його обдурити.

В загальному випадку обміну криптовалютами може брати участь довільна кількість учасників. Структура обміну задається за допомогою орієнтованого

зваженого графу D(V, E), де множина вершин V - учасники обміну, а множина ребр E – фактичне бажання обміну криптовалюти [14].

Обов’язково атомарний обмін має гарантувати наступне:

- якщо всі учасники обміну слідують протоколу, всі обміни відбудуться;
- якщо існує коаліція, яка спеціально порушує правила протоколу, то учасники, що чинили чесно, не отримують збитків;
- не існує раціональної причини не слідувати існуючому протоколу.

Найрозповсюдженим випадком обміну криптовалюти є обмін між двома учасниками. Для розгляду даного випадку, означимо спочатку визначення атомарності обміну між двома учасниками.

Атомарний протокол обміну для двох учасників має гарантувати:

- Якщо обидва учасники виконують протокол, обидва обміни відбудуться.
- При порушенні протоколу одним учасником, інший не втратить свої кошти.
- Жодному з учасників невигідно порушення протоколу.

Наведемо приклад протоколу, який дозволяє досягти атомарності у обміні криптовалютами між двома учасниками.

Припустимо, що Аліса має 1 токен блокчейну виду 1, а Боб - 1 токен блокчейну виду 2.

1. На першому кроці Аліса генерує секретне значення s і обчислює значення обраної хеш-функції hash(s) = hs, тримаючи в секреті значення s.
2. Далі Аліса публікує в блокчейні виду 1 хешований смарт-контракт з часовою затримкою. За умов смарт-контракту, Боб отримає 1 токен блокчейну виду 1, якщо він назве таке значення s’, що hash(s’) = hs.

Інакше у випадку, коли Боб не називає таке значення, то через зазначений у смарт-контракті час t1, Аліса отримає свій токен назад.

3. Наступним кроком Боб має перевірити контракт, який був опублікований Алісою на його коректність.
4. Далі Боб повинен опублікувати подібний контракт на блокчейні виду 2. Указавши в умовах контракту, що Аліса отримає 1 токен блокчейну виду 2, при виконанні умови, що вона назве таке значення s'', що hash(s'') = hs, де hs Боб дізнається з контракту на блокчейні виду 1. Інакше, за умови, що Аліса не називає правильне значення, то через час t2, Боб отримає свій токен назад.
5. Використавши значення s у контракті Боба, Аліса отримає токен блокчейну виду 2 та озвучує значення s Бобу.
6. У свою чергу Боб використовуючи значення s, що йому розповіла Аліса, розблокує контракт Аліси та отримає токен блокчейну виду 1.

На рисунку 2.1 схематично зображено алгоритм обміну токенів двох видів криптовалют.

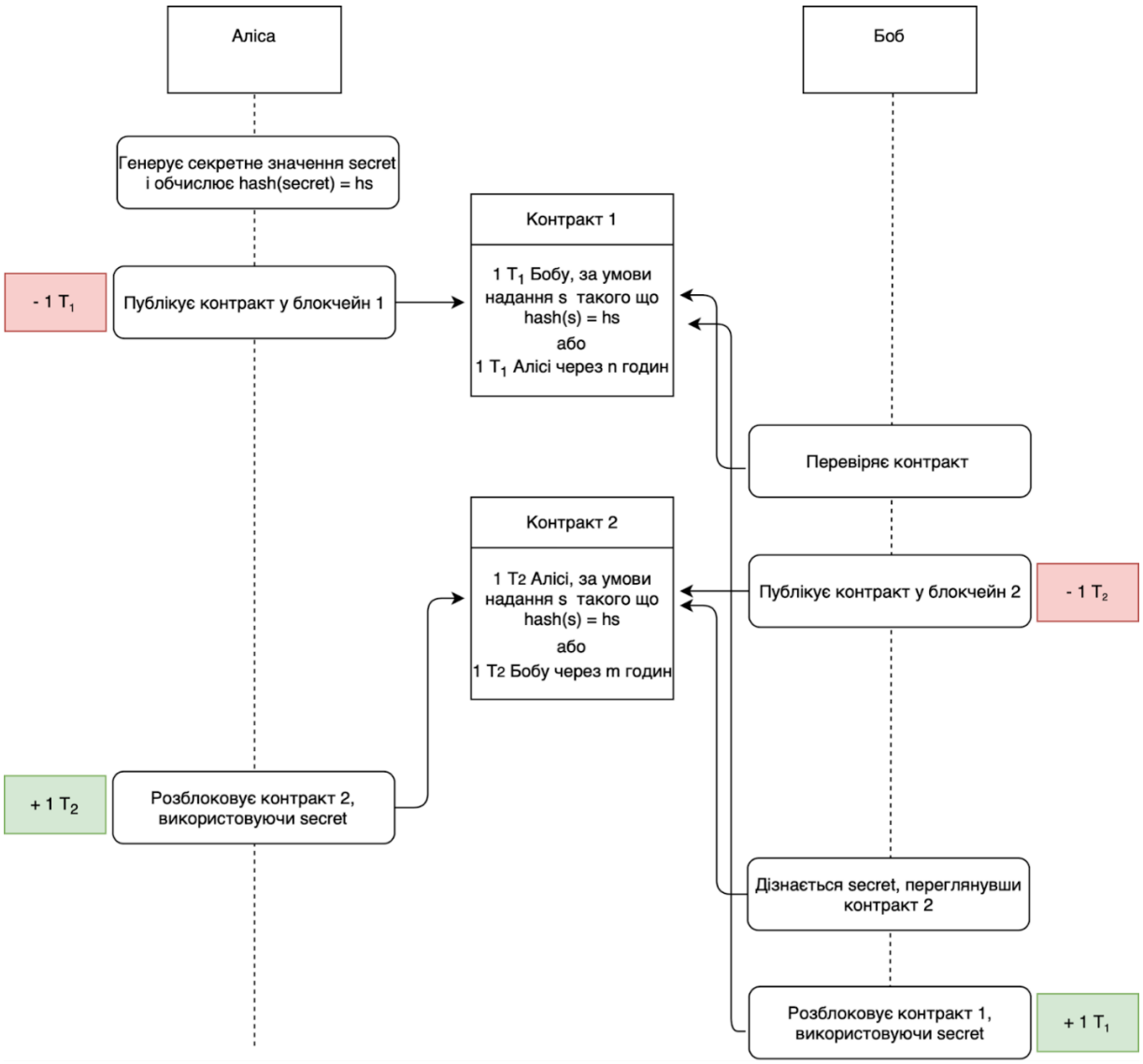

Рисунок 2.1 Схематичне зображення атомарного обміну криптовалюти

РОЗДІЛ 3. РОЗРОБКА СИСТЕМИ ДЛЯ ВИКОНАННЯ АТОМАРНОГО ОБМІНУ МІЖ ДВОМА УЧАСНИКАМИ

3.1. Вибір інструментів реалізації

Для реалізації смарт-контракту з часовою затримкою, необхідного для атомарного обміну, у мережі NEAR Protocol обрано мову Rust, оскільки офіційний відкритий код самої мережі також реалізовано засобами мови програмування Rust.

Локальна версія мережі розгортається за допомогою спеціального модуля Kurtosis NEAR. Kurtosis надає розробникам зручні інструменти для розробки персоналізованих середовищ, оптимізованих для створення прототипів і end-to-end тестування в просторі блокчейну [15].

Модуль Kurtosis NEAR розгортає на локальній машині Docker контейнери, використовуючи скрипт для завантаження середовища, що містять всі необхідні інструменти для розробника. Також доступний інтерфейс управління гаманця [16].

Для мережі Bitcoin необхідний смарт-контракт з часовою затримкою реалізовано засобами мови програмування Golang. Такий вибір було зроблено у зв’язку з тим, що офіційна реалізація вузлів мережі написана мовою Golang. Це дає змогу використовувати готові реалізації структури транзакцій, необхідні константи тощо. Також є доступ до середовища виконання смарт-контрактів, тобто є можливість налаштування автоматизованого тестування реалізації смарт-контрактів.

Для тестування локально створеного смарт-контракту використано симуляційну мережу Bitcoin, локальний вузол btcd у режимі simnet.

3.2. Реалізація HTLC для атомарного обміну у мережі NEAR

Основним вузлом, на якому виконується смарт-контракт з часовою затримкою, виступає створений додатково гаманець у мережі NEAR Protocol. На цьому гаманці скомпільовано код, що містить в собі об’єкта так званого менеджера, який містить створені контракти та низку функцій, таких як:

- new(self, lifetime, secret, recipient) – створює новий HTLC протокол з заданими параметрами та додає його до списку існуючих;
- check_valid_time(self, contract_id) – повертає True, якщо ще не вийшов час дії контракту з заданим contract_id, інакше False;
- check_claimer(self, contract_id) – перевіряє на відповідність адресу гаманця отримувача, та адресу користувача, що зініціював запит на відкриття заблокованого смарт-контракту;
- check-secret(self, contract_id, secret) – перевіряє коректність секретної фрази для відкриття заблокованого смарт-контракту;
- claim(self, contract_id, secret) – за допомогою цієї функції користувач може розблокувати смарт-контракт та отримати свої кошти;
- refund(self, contract_id) – користувач, що ініціював HTLC протокол може отримати свої кошти назад, у разі не виконання умов опонентом за визначений час.

Користувачі можуть спілкуватися зі згаданим вище акаунтом за допомогою команд NEAR CLI, передаючи параметри викликаної функції у форматі JSON. Також варто зазначити, що тимчасово заблоковані кошти ініціатора смарт-контракту з часовою затримкою знаходяться на акаунті разом зі створеним HTLC протоколом. Далі їх або отримує опонент, правильно назвавши секретну фразу, або ініціатор повертає кошти назад, по закінченню часу існування контракту.

3.3. Реалізація HTLC для атомарного обміну у мережі Bitcoin

Для реалізації смарт-контракту з часовою затримкою, необхідного для проведення атомарного обміну, у мережі Bitcoin використано офіційну реалізацію вузлів мережі btcd там запити на мові Script. Btcd дозволяє використовувати структури транзакцій, стандарти кодування даних для мережі Bitcoin [17].

Структура для відображення параметрів смарт-контракту наступна:

- From – ініціалізатор смарт-контракту, тобто відправник коштів.
- To – отримувач коштів, у разі виконання вимог смарт-контракту.
- Amount – кількість коштів, що приймають участь у контракті.
- TimeLock – час, який існує смарт-контракт.
- SecretHash – хеш секретної фрази.

Також реалізовано необхідні функції, для створення та управління смарт-контрактом. Наведемо деякі з них:

- CreateHTLC – створення нового смарт-контракту.
- RedeemHTLC – відкриття смарт-контракту, використавши секретну фразу
- RefundHTLC – повернення коштів, у разі не виконання опонентом умов смарт-контракту.

Користувач спілкується зі створеною програмою за допомогою спеціальних запитів CLI, задаючи необхідні параметри.

3.4. Тестування проведення атомарного обміну на симуляційних мережах

Опишемо приклад проведення атомарного обміну між двома користувачами, використовуючи локальні симуляційні мережі Bitcoin та NEAR Protocol.

Перед початком виконання атомарного обміну, переглянемо стани рахунків двох користувачів, нехай це буде Аліса та Боб, у мережах.

За допомогою запитів до мережі Bitcoin можемо дізнатися стан рахунків Аліси та Боба:

```
liza@LAPTOP-BDLMGH3E:~/Diploma$ btcctl --simnet --rpcuser=admin --rpcpass=admin --wallet getbalance bob
2
liza@LAPTOP-BDLMGH3E:~/Diploma$ btcctl --simnet --rpcuser=admin --rpcpass=admin --wallet getbalance alice
0
```

Рисунок 3.1 Стан рахунків учасників обміну, до початку проведення атомарного обміну

Подібним чином дізнаємося стан рахунків учасників обміну у мережі NEAR Protocol:

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near state alice.test.near
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Account alice.test.near
{ amount: '69915600090433020000000000',
  block_hash: 'GAUdS9uWxzmKjhGCqA1up5zBZre5LVHe5c3RR1MH84zv',
  block_height: 68684,
  code_hash: '11111111111111111111111111111111',
  locked: '0',
  storage_paid_at: 0,
  storage_usage: 182,
  formattedAmount: '69.91560009043302' }
```

Рисунок 3.2 Стан рахунку Аліси у мережі NEAR Protocol до початку обміну

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near state bob.test.near
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Account bob.test.near
{ amount: '99973293842185694000000000',
  block_hash: '4fQLMEEU1x3hEe3i84WAhCV88i2JCjQPtTRWtUXmK3j7',
  block_height: 68769,
  code_hash: '11111111111111111111111111111111',
  locked: '0',
  storage_paid_at: 0,
  storage_usage: 182,
  formattedAmount: '99.973293842185694' }
```

Рисунок 3.3 Стан рахунку Боба у мережі NEAR Protocol до початку обміну

Далі розпочинаємо процес атомарного обміну. Нехай Боб хоче обміняти свої 2 Bitcoin на 20 NEAR у Аліси. Розпишемо атомарний обмін покроково:

1. Нехай першим ініціює протокол Боб. Він придумує секретну фразу, та за допомогою створеної функції у мережі NEAR Protocol дізнається хеш секретної фрази.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ echo -n secretphrase | sha256sum
ec81d47e5650c9b41215fc39d556b55b20b85e5c1ebcc315f706986a2b389c56  -
```

Рисунок 3.4 Отримання хеша секретної фрази

2. Тепер Боб, використовуючи отриманий хеш, створює контракт у мережі Bitcoin, у якому зазначує скільки та кому будуть переведені токени, у разі успішного виконання контракту.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/ACCS_$ export BOB_BTC_ADDRESS=SNgJ8WD7aJcM3sEVWCm2f9tD52TRKSwYF6
liza@LAPTOP-BDLMGH3E:~/Diploma/ACCS_$ export ALICE_BTC_ADDRESS=ShRZcm7kQYFUuhV55rPpgjAaCvxeFPuTjp
liza@LAPTOP-BDLMGH3E:~/Diploma/ACCS_$ ./ACCS create-htlc --blockchain BTC --from $BOB_BTC_ADDRESS --to $ALICE_BTC_ADDRESS --amount 2.0
 --hash ec81d47e5650c9b41215fc39d556b55b20b85e5c1ebcc315f706986a2b389c56 --hash-len 12 --time 3600
Please enter password for unlocking wallet
Successfully deployed HTLC. Transaction ID: edcb1a847490cbb5c52bbcbe090fd2c065fa2dd1ce2313fc166bcbbd0018c6a1
```

Рисунок 3.5 Створення смарт-контракту у мережі Bitcoin

3. Далі Боб повідомляє Алісі з яким хешом було створено контракт. Аліса у свою чергу створює смарт-контракт у мережі NEAR Protocol, підписавши його тим самим хешом.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near call htlc.test.near new '{"lifetime": 3600000000000, "secret":"ec81
d47e5650c9b41215fc39d556b55b20b85e5c1ebcc315f706986a2b389c56", "recipient":"bob.test.near"}' --accountId alice.test.near --deposit 20
Scheduling a call: htlc.test.near.new({"lifetime": 3600000000000, "secret":"ec81d47e5650c9b41215fc39d556b55b20b85e5c1ebcc315f706986a2b
389c56", "recipient":"bob.test.near"}) with attached 20 NEAR
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Doing account.functionCall()
Transaction Id 3GzBuRhJcnqLRvYsg58YuahnHNf6ktsrKCAFjXArXKQc
To see the transaction in the transaction explorer, please open this url in your browser
http://127.0.0.1:52909/transactions/3GzBuRhJcnqLRvYsg58YuahnHNf6ktsrKCAFjXArXKQc
'dddd64e997895d4333ebc132f721eb7d90f42eb9461c60ccef9c45cfecd91a18'
```

Рисунок 3.6 Створення смарт-контракту у мережі NEAR Protocol

Також перейшовши по посиланню, яке можна побачити на Рисунку 3.6, є можливість переглянути деталі транзакції. Оскільки контракти зберігаються у додатково створеному акаунті, тимчасово заблоковані кошти також зберігаються на цьому акаунті. На Рисунку 3.7 бачимо усі деталі транзакції з акаунту Аліси на акаунт, де зберігається створений нею контракт разом із заблокованими коштами. Тобто наразі ніхто не має доступу до цих коштів, окрім root акаунту. Але вважаємо, що він вміє лише відповідати на запити користувачів та зберігати в собі створені контракти.

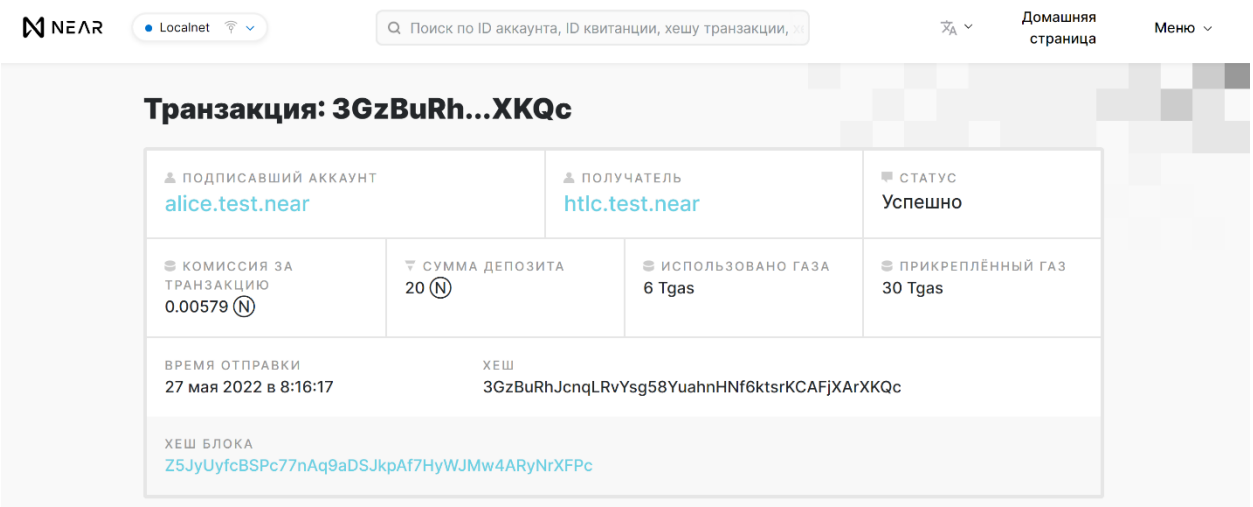

Рисунок 3.7 Деталі транзакції заблокованих тимчасово коштів за акаунту Аліси на root акаунт

4. Після того, як створені обидва контракти, Боб розблоковує смарт-контракт у NEAR Protocol, використавши відому йому секретну фразу.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near call htlc.test.near claim '{"contract_id":"dddd64e997895d4333ebc132
f721eb7d90f42eb9461c60ccef9c45cfecd91a18", "secret":"secretphrase"}' --accountId bob.test.near
Scheduling a call: htlc.test.near.claim({"contract_id":"dddd64e997895d4333ebc132f721eb7d90f42eb9461c60ccef9c45cfecd91a18", "secret":"s
ecretphrase"})
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Doing account.functionCall()
Transaction Id BXfiAtyLbbwSS9dtfSpgfaB4TfXyAuBhWNQ6HjevHGHX
To see the transaction in the transaction explorer, please open this url in your browser
http://127.0.0.1:52909/transactions/BXfiAtyLbbwSS9dtfSpgfaB4TfXyAuBhWNQ6HjevHGHX
''
```

Рисунок 3.8 Розблокування смарт-контракту у мережі NEAR Protocol

Можемо тепер переглянути стан акаунтів Боба та Аліси та переконатися, що він отримав обіцяні кошти від Аліси.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near state bob.test.near
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Account bob.test.near
{ amount: '119967527079713257000000000',
  block_hash: 'ACe1fziWqRz1y9QSQouc6LnoQwER9cVyqakY4tAZewJk',
  block_height: 69899,
  code_hash: '11111111111111111111111111111111',
  locked: '0',
  storage_paid_at: 0,
  storage_usage: 182,
  formattedAmount: '119.967527079713257' }
```

Рисунок 3.9 Стан акаунту Боба у мережі NEAR Protocol

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near state alice.test.near
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Account alice.test.near
{ amount: '49909807720339352000000000',
  block_hash: 'DB5fiq4Q59gCQ2tgDr6CHQRzYmQE7W8FxpdhL7JHwMmi',
  block_height: 69950,
  code_hash: '11111111111111111111111111111111',
  locked: '0',
  storage_paid_at: 0,
  storage_usage: 182,
  formattedAmount: '49.909807720339352' }
```

Рисунок 3.10 Стан акаунту Аліси у мережі NEAR Protocol

5. Після розблокування Бобом смарт-контракту секретною фразою, Аліса може дізнатися цю фразу, зробивши наступний запит:

```
liza@LAPTOP-BDLMGH3E:~/Diploma/near-protocol/rust-htlc$ local_near call htlc.test.near get_secret '{"contract_id":"dddd64e997895d4333e
bc132f721eb7d90f42eb9461c60ccef9c45cfecd91a18"}' --accountId alice.test.near
Scheduling a call: htlc.test.near.get_secret({"contract_id":"dddd64e997895d4333ebc132f721eb7d90f42eb9461c60ccef9c45cfecd91a18"})
Loaded master account test.near key from /home/liza/.neartosis/2022-05-26T18.49.33/validator-key.json with public key = ed25519:JBqAup
5y5GqxVjqg9ytzRrSbQ4eKEgDNMC4w6WqAUwPa
Doing account.functionCall()
Transaction Id 7MAL9ZzTLNm59StAz1nVmbU69jviUca1ZsiAsFvucVFJ
To see the transaction in the transaction explorer, please open this url in your browser
http://127.0.0.1:52909/transactions/7MAL9ZzTLNm59StAz1nVmbU69jviUca1ZsiAsFvucVFJ
'secretphrase'
```

Рисунок 3.11 Отримання секретної фрази у мережі NEAR Protocol

6. Наступним кроком, Аліса розблоковує смарт-контракт у мережі Bitcoin та отримує обіцяні кошти.

```
liza@LAPTOP-BDLMGH3E:~/Diploma/ACCS_$ ./ACCS redeem-htlc --blockchain BTC --to $ALICE_BTC_ADDRESS --txid edcb1a847490cbb5c52bbcbe090fd
2c065fa2dd1ce2313fc166bcbbd0018c6a1 --secret secretphrase
Please enter password for unlocking wallet
Successfully deployed transaction to redeem HTLC. Transaction ID: ac748c97f916e7e610486b3f9a649f9a9f684ed34fec73fdce0f9a91367e2051
```

Рисунок 3.12 Розблокування смарт-контракту у мережі Bitcoin

Можемо тепер переглянути стан акаунтів Боба та Аліси та переконатися, що вона отримала обіцяні кошти від Боба.

```
liza@LAPTOP-BDLMGH3E:~/Diploma$ btcctl --simnet --rpcuser=admin --rpcpass=admin --wallet getbalance bob
0
liza@LAPTOP-BDLMGH3E:~/Diploma$ btcctl --simnet --rpcuser=admin --rpcpass=admin --wallet getbalance alice
1.99998996
```

Рисунок 3.13 Стан акаунтів Боба та Аліси у мережі Bitcoin

Атомарний обмін між токенами Bitcoin та NEAR Protocol проведено успішно, кожен отримав обіцяні кошти. Також бачимо, що Аліса гарантовано знатиме секретну фразу, після використання її Бобом, тобто Боб не зможе забрати кошти Аліси собі і при цьому надурити Алісу.

Отже, усі вимоги до атомарності обміну між токенами різних криптовалют виконані.

ВИСНОВКИ

В даній кваліфікаційній роботі було виконано поставлені завдання, а саме:

- поглиблено знання з блокчейн-технологій, ознайомилися з існуючими криптовалютами;
- досліджено відомі на сьогоднішній день технології, за допомогою яких можна проводити обмін токенами різних блокчейнів атомарно;
- розроблено та описали алгоритм проведення атомарного обміну криптовалюти;
- розроблено хешовані смарт-контракти з часовою затримкою для мереж Bitcoin та NEAR Protocol;
- розроблено зручні функції, що дозволяє провести атомарний обмін між криптовалютами, спілкуючись з симуляційними мережами запитами;
- протестувано створене програмне забезпечення, використовуючи мережі у тестовому та симуляційному режимах;

Результатом кваліфікаційної роботи є програмне забезпечення для проведення атомарного обміну криптовалютами між токенами мереж Bitcoin та NEAR Protocol.

ДОДАТОК А
Лістинг коду смарт-контракту для мережі NEAR Protocol

```
//! Rust implementation of Hashed Time-Locked Contract on NEAR blockchain
//!
use near_sdk::borsh::{self, BorshDeserialize, BorshSerialize};
use near_sdk::{env, near_bindgen};
use near_sdk::{AccountId, Promise, Timestamp, Duration, CryptoHash, Balance};
use std::collections::HashMap;
use std::fmt;
use hex;

near_sdk::setup_alloc!();

#[derive(Default, BorshDeserialize, BorshSerialize)]
pub struct HTLC {
   creation_time: Timestamp,
   lifetime: Duration,
   secret_hash: CryptoHash,
   amount: Balance,
   sender_address: AccountId,
   recipient_address: AccountId,
   completed: bool,
   middle_acc: AccountId,
}

impl fmt::Display for HTLC {
   fn fmt(&self, f: &mut fmt::Formatter) -> fmt::Result {
      write!(f, "Creation time: {}, Lifetime: {}, Amount: {}, Sender: {},
               Recipient: {}", self.creation_time, self.lifetime,
               self.amount, self.sender_address, self.recipient_address)
   }
}

#[near_bindgen]
#[derive(Default, BorshDeserialize, BorshSerialize)]
pub struct HTLCManager {
   contracts: HashMap<String, HTLC>
}
```

```
#[near_bindgen]
impl HTLCManager {
   #[init]
   pub fn init() -> Self {
      let res = Self::default();
      return res;
}

#[payable]
pub fn new(&mut self, lifetime: Duration, secret: String, recipient: AccountId) ->
String {
      let mut res = HTLC::default();
      res.creation_time = env::block_timestamp();
      res.lifetime = lifetime;
      res.secret_hash = match env::sha256(secret.as_bytes()).try_into() {
         Ok(x) => x,
         Err(_) => panic!("expected len 32")
      };
      res.amount = env::attached_deposit();
      res.sender_address = env::signer_account_id();
      res.recipient_address = recipient;
      res.completed = false;
      res.middle_acc = env::current_account_id();
      Promise::new(env::current_account_id())
         .transfer(res.amount);
      let contract_id = hex::encode(env::sha256(res.to_string().as_bytes()));
      self.contracts.insert(contract_id.clone(), res);
      return contract_id;
}

#[private]
pub fn check_valid_time(&self, contract_id: &String) -> bool {
      let current_time = env::block_timestamp();
      return self.contracts[contract_id].creation_time +
self.contracts[contract_id].lifetime > current_time;
}

#[private]
pub fn check_claimer(&self, contract_id: &String) -> bool {
```

```
        return env::signer_account_id() ==
self.contracts[contract_id].recipient_address;
}

#[private]
pub fn check_secret(&self, contract_id: &String, secret: &String) -> bool {
        return env::sha256(secret.as_bytes()) ==
self.contracts[contract_id].secret_hash;
}

pub fn get_timestamp(&self, contract_id: &String) -> String {
        return format!("{} {}", env::block_timestamp(),
self.contracts[contract_id].creation_time);
}

#[payable]
pub fn claim(&mut self, contract_id: &String, secret: &String) {
        let contract = match self.contracts.get(contract_id) {
            Some(x) => x,
            None => env::panic(b"ERROR: No contract was found for given contract
id")
        };
        if contract.completed {
            env::panic(b"ERROR: Contract already fulfilled!");
        }
        if !self.check_valid_time(contract_id) {
            env::panic(b"ERROR: Contract expired!");
        }
        if !self.check_claimer(contract_id) {
            env::panic(format!("Recipient id: {}, current id: {}",
                        contract.recipient_address,
                        env::signer_account_id()).as_bytes());
        }
        if !self.check_secret(contract_id, secret) {
            env::panic(b"ERROR: Provided unlocker secret is incorrect!");
        }
        Promise::new(contract.recipient_address.clone())
            .transfer(contract.amount);

        self.contracts.get_mut(contract_id).unwrap().completed = true;
```

```
}

#[payable]
pub fn refund(&mut self, contract_id: &String) {
        let contract = match self.contracts.get(contract_id) {
            Some(x) => x,
            None => env::panic(b"ERROR: No contract was found for given contract
id")
        };
        if contract.completed {
            env::panic(b"ERROR: This contract was already completed")
        }
        if self.check_valid_time(contract_id) {
            env::panic(b"ERROR: This contract is not yet available for refund")
        }
        if env::signer_account_id() != contract.sender_address {
            env::panic(b"ERROR: Cannot verify identity of account requesting a
refund")
        }

        Promise::new(contract.sender_address.clone())
            .transfer(contract.amount);
        self.contracts.get_mut(contract_id).unwrap().completed = true;
    }
}
```

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Децентралізована біржа Uniswap [Електронний ресурс] – Режим доступу до ресурсу: https://app.uniswap.org/#/swap.
2. Визначення блокчейну та його структура [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Blockchain.
3. Дерево Меркла. Застосування дерева Меркла у Blockchain [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/techskill-brew/merkle-tree-in-blockchain-part-5-blockchain-basics-4e25b61179a2.
4. Мережа Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bitcoin_network.
5. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Електронний ресурс] / Satoshi Nakamoto – Режим доступу до ресурсу: https://bitcoin.org/bitcoin.pdf
6. Bitcoin. A Primer for Policymakers [Електронний ресурс] / Jerry Brito – Режим доступу до ресурсу: https://www.mercatus.org/system/files/Brito_BitcoinPrimer.pdf
7. Визначення смарт-контрактів у криптовалюті [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Smart_contract
8. Мережа NEAR Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://academy.binance.com/ru/articles/what-is-near-protocol-near
9. Смарт-контракти у мережі NEAR Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://www.leewayhertz.com/create-smart-contracts-on-near-protocol/

10. Офіційна документація створення смарт-контрактів у мережі NEAR Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://docs.near.org/docs/develop/contracts/overview#
11. Смарт-контракти з часовою затримкою у мережі Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts
12. Смарт-контракти з часовою затримкою [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/terms/h/hashed-timelock-contract.asp
13. Опис мови програмування Bitcoin Script [Електронний ресурс] – Режим доступу до ресурсу: https://en.bitcoin.it/wiki/Script
14. Herlihy M. Atomic Cross-Chain Swaps [Електронний ресурс] / Maurice Herlihy – Режим доступу до ресурсу: https://arxiv.org/pdf/1801.09515.pdf
15. Платформа розробки Kurtosis [Електронний ресурс] – Режим доступу до ресурсу: https://www.kurtosistech.com/
16. Офіційна документація розгортання симуляційної мережі NEAR Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://docs.near.org/docs/tools/kurtosis-localnet
17. Реалізація вузла Bitcoin мовою програмування Go [Електронний ресурс] – Режим доступу до ресурсу: https://pkg.go.dev/github.com/btcsuite/btcd.