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

«До захисту допущено»

Завідувач кафедри
Терещенко В.М. _____
(підпис)

«__»_____20__р.

Дипломна робота
на здобуття ступеня бакалавра

за освітньо-професійною програмою "Інформатика"

спеціальності 122 "Комп'ютерні науки"

на тему:

АТОМАРНИЙ ОБМІН КРИПТОВАЛЮТ

Виконала студентка 4 курсу
Щербіна Марія Сергіївна _____
(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 – 2021

РЕФЕРАТ

Обсяг роботи: 42 сторінки, 8 ілюстрацій, 16 джерел посилань.

АТОМАРНИЙ ОБМІН КРИПТОВАЛЮТ, СМАРТ-КОНТРАКТИ, КРИПТОВАЛЮТИ, БЛОКЧЕЙН-ТЕХНОЛОГІЇ.

Об’єктом роботи є процес організації атомарного обміну криптовалютами. Предметом роботи є програмний засіб для проведення атомарного обміну криптовалютами.

Метою роботи є дослідження криптографічних методів забезпечення атомарності обміну криптовалютами та розробка програмного засобу для виконання атомарного обміну між токенами різних блокчейнів.

Методи розроблення: теоретичне дослідження, розробка програмного продукту. Інструменти розроблення: операційна система - Mac OS Catalina, середовище програмування GoLand 2020, компоненти системи реалізовані мовами загального призначення Golang та JavaScript, а також мовами написання смарт-контрактів Solidity та Bitcoin Script.

Результати роботи: виконано загальний огляд існуючих технологій проведення атомарного обміну криптовалютами, проаналізовано методи роботи з смарт-контрактами в різних блокчейн-мережах, розроблено систему для проведення атомарного обміну між криптовалютами.

Розроблений програмний продукт може застосовуватись для проведення атомарного обміну між криптовалютами у мережах Bitcoin та Ethereum.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ВИЗНАЧЕНЬ

HTLC(Hashed Time-Locked Contract) - смарт-контракт з перевіркою хеш-суми та часовою затримкою;

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

DEX(Decentralized exchange) - децентралізована біржа;

ERC(Ethereum Request for Comments) - запит коментарів платформи Ethereum;

BIP(Bitcoin Improvement Protocol) - протокол покращення Bitcoin;

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

UML(Unified Modeling Language) - система позначень для об’єктно-орієнтованого проектування та аналізу;

EVM(Ethereum Virtual Machine) - віртуальна машина платформи Ethereum;

P2PKH(pay to public key hash) - вид транзакцій, що виконують переказ на задану адресу;

P2SH(pay to script hash) - узагальнені транзакції, що дозволяють використання довільних блокуючих скриптів;

UTXO(Unspent transaction output) - невитрачений вихід транзакції.

ВСТУП

Оцінка сучасного стану об’єкта розробки. З моменту першої публікації з описом застосування блокчейн-технології для розробки однорангової криптовалюти[1] у 2008 році, технологія набула широкої популярності. З того часу, почали з’являтись різні реалізації криптовалют, на основі блокчейну, в тому числі Bitcoin[2], Ethereum[3], Stellar[4], тощо. Це спричинило стрімке зростання попиту на дослідження безпечних методи обміну токенами різних блокчейнів.

Для повноцінного використання електронних токенів в якості цінного активу, зберігаючи переваги розподіленої системи, почали з’являтись перші децентралізовані біржі, що дозволяють проводити атомарні обміни криптовалют. Серед таких, наприклад, біржа Uniswap[5], що дозволяє проводити обмін між токенами стандарту ERC-20. Також, з’являють реалізації міжланцюгових гаманців, з підтримкою атомарного обміну криптовалют, наприклад Liquality[6].

Тим не менш, на сьогоднішній день, більшість програмних продуктів для проведення атомарного обміну криптовалют, дозволяють проводити децентралізований обмін тільки для токенів, що використовують технології платформи Ethereum, через виразність мови написання розумних контрактів. Для токенів, що мають обмежену підтримку смарт-контрактів(наприклад, Bitcoin) підтримка атомарних обмінів не є розповсюдженою.

Актуальність роботи та підстави для її виконання. Станом на сьогодні, в світі існує велика кількість різноманітних криптовалют, що використовують різноманітні блокчейни. Обмін між класичними валютами зазвичай виконується через посередника, що виступає контролюючим органом, гарантує виконання вимог сторонами угоди, та потребує довіри

обох сторін. Однією з основних переваг використання криптовалюти є одноранговість мережі, тобто відсутність потреби в централізованих установах для виконання операцій з валютою.

Блокчейн-технології дозволяють безпечно та без довіри проводити операції з внутрішньою валютою блокчейну, але для виконання угод, що вносять зміни в кілька мереж одночасно необхідно використовувати додаткові механізми захисту. Одним з прикладів таких операцій є обмін криптовалютами. Для того, щоб за відсутності третьої сторони - арбітра угоди, провести обмін токенами різних блокчейнів без ризику втрати коштів, розробляються технології атомарного обміну криптовалютами.

Таким чином, застосування даних технологій дозволяє розширити сценарії використання криптовалюти, додати нові можливості, зберігаючи існуючі переваги, як-то відсутність потреби у гаранті для проведення угод.

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

Для досягнення цієї мети поставлено такі завдання:

- дослідити існуючі системи та технології, що дозволяють проводити обмін криптовалютами;
- поглибити знання у криптографії та блокчейн-технологіях;
- розробити та описати алгоритм проведення атомарного обміну криптовалютами;
- розробити смарт-контракти для реалізації алгоритму атомарного обміну токенами;
- реалізувати необхідні операції для мережі Bitcoin;
- реалізувати необхідні операції для мережі Ethereum;

- розробити інтерактивну систему, що дозволятиме двом користувачам обміняти токени різних блокчейнів;
- протестувати розроблене програмне забезпечення, з використанням локальної симуляції мережі;
- протестувати розроблене програмне забезпечення, з використанням тестових мереж відповідних блокчейнів, що використовують технології реальної мережі;
- навести приклади застосування отриманого програмного забезпечення.

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

Реалізації програмного засобу передувала проектування системи та розробка смарт-контрактів. Основою для розробки стали проведені дослідження існуючих методів та алгоритмів проведення атомарного обміну криптовалютами, вивчення особливостей обраних мереж та механізмів розробки та виконання розумних контрактів в загальному та у мережах Bitcoin та Ethereum.

Для розробки основної частини програмного продукту було використано мову програмування Golang(Go) за фактом існування реалізації вузлів блокчейн мереж Bitcoin(btcd[7]) та Ethereum(geth[8]) даною мовою та бібліотек для розробки користувацького інтерфейсу, а також високої ефективності програм даною мовою через її компільованість. Додатково у проекті використовувалась мова програмування JavaScript для тестування у симуляційній мережі Ethereum за допомогою утиліти Ganache, що є частиною Truffle Suite[9].

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

Програмне забезпечення було розроблено на операційній системі MacOs Catalina. Для створення проекту було використано середовище програмування GoLand 2020.

Можливі сфери застосування. Розроблений програмний продукт може застосовуватись для проведення атомарного обміну між криптовалютами блокчейнів Bitcoin та Ethereum. Програмне забезпечення є легко розширювальним для підтримки інших блокчейнів, та може стати основою для створення децентралізованої біржі. Компоненти розробленої системи, такі як реалізації смарт-контрактів, можуть бути використані для реалізації поза ланцюгових протоколів, наприклад, протоколу платіжних каналів Lightning Network[10].

РОЗДІЛ 1. БЛОКЧЕЙН-ТЕХНОЛОГІЇ

Блокчейн технологія вперше була описана Сатоші Накамото у праці ”Біткойн: однорангова електронна система готівки” [1] у 2008 році, в якості розв’язку проблеми контролю за подвійними витратами токенів без потреби у посередниках. З моменту першої публікації, технологія набула великої популярності, що спричинило бурхливий розвиток даної галузі криптографії та появу великої кількості реалізацій технології, зокрема криптовалюти Bitcoin та Ethereum.

1.1 Загальний огляд

Блокчейн являє собою розподілену базу даних, організовану у вигляді ланцюгу блоків, пов’язаних за допомогою операції хешування. Схема даних показана на рисунку 1.1:

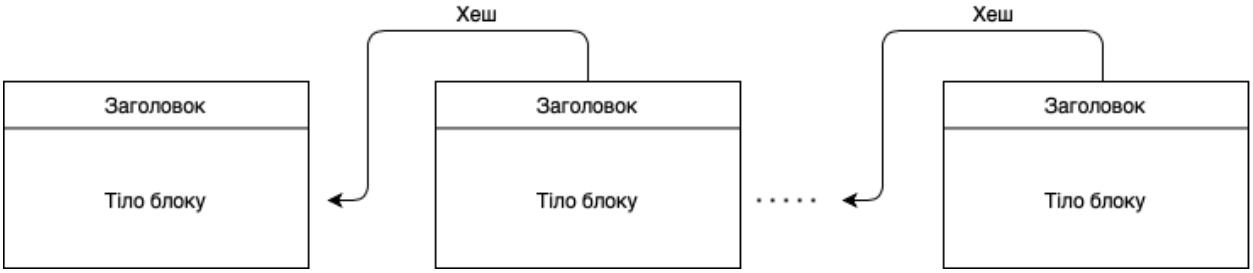

Рисунок 1.1 Структура блокчейна

Кожен блок складається з корисної інформації - тіла блоку, та заголовку, що є службовою частиною для організації сховища даних. В заголовку кожного блоку міститься зокрема хеш попереднього блоку. Така технологія дозволяє використовувати хеш у якості посилання для впорядкування блоків, аналогічно до алгоритмічної структури “зв’язний список”. Основною перевагою даного підходу до організації даних є зростаюча складність модифікації даних, що зберігаються.

Розглянемо наступну ситуацію: нехай зловмисник намагається

підмінити дані блоку номер M, в блокчейні з N блоків. За умови зміни у інформації блоку, зміниться значення хеш-функції даного блоку(блоку M). За будовою блокчейну, так як хеш-сума блоку M записана у блоці M+1, зловмиснику доведеться записати нове значення хеш-суми у заголовок блоку M+1, що вплине на його значення хеш-функції. Так, для підміни даних у блоці на глибині h(h = N - M), необхідно переробити обчислення для h блоків. Для того, щоб було важко додати наступний блок, і тим самим унеможливити атаку на вже записані блоки, використовують спеціальні техніки розподіленого консенсусу. Таким чином, з появою кожного наступного блоку, кількість роботи, яку необхідно проробити для модифікації конкретного блоку, зростає. Це дозволяє сприймати інформацію в блоках, починаючи з певної глибини, як істинну.

Блокчейн використовується в якості розподіленої бази даних в однорангових мережах. Так, кожен вузол мережі зберігає у себе копію усього блокчейну. Будь-яка зміна блокчейну(наприклад, поява нового блоку) поширюється до сусідніх вузлів ініціатором події, які перевіряють коректність нового стану даних та передають подію наступним вузлам, поки кожен учасник мережі не матиме дану зміну у своєму блокчейні. Для того, щоб коректно визначити ініціатора зміни та розв’язувати конфлікти, у випадку якщо декілька вершин спробували поширити зміни до блокчейну, використовують різноманітні механізми досягнення розподіленого консенсусу. Це дозволяє зберігати мережу децентралізованою, адже у класичному випадку підтримка коректності стану даних делегується сертифікованій установі(наприклад, для звичайних валютних операцій - банку).

Найбільш поширеною технікою досягнення консенсусу є PoW(доказ роботи). Так для того, щоб наступний блок міг бути доданим до блокчейну,

необхідно підібрати префікс такий, щоб хеш-сума блоку не перевищувала наперед задане значення(так звана “складність блоку”). Дане значення регулюється в залежності від швидкості генерації префіксів з метою стабілізації частоти появи нових блоків. З властивостей хеш-функції, підбір такого числа є обчислювально складним завданням, яке вимагає великих обчислювальних потужностей. При цьому, за умови знаходження такого значення, перевірка коректності відповіді, є достатньо простим завданням. Таким чином, кожен учасник мережі може перевірити коректність сформованого блоку, але надання відповідного значення одним з вузлів мережі є доказом проробленої роботи з пошуку відповіді(що відображено в назві даного методу). Для проведення атаки з підміною інформації в блоці, необхідно, щоб зловмисник володів більшою частиною обчислювальних ресурсів мережі(так звана “атака 51%”), що в реальності не є можливим.

1.2 Переваги і недоліки в сфері валютних операцій

У випадку класичних електронних операцій з валютою, більшість зобов’язань з забезпечення коректності балансів бере на себе банк або інша фінансова інституція. Такий підхід має ряд недоліків, які пов’язані з централізованістю даного підходу. Серед найбільш значущих:

1. Кожна транзакція відбувається через банк, відповідно графік виконання операцій залежить виключно від можливостей банку.
2. Відсутність анонімності.
3. Банк може накласти вето на проведення транзакції.

Для того, щоб позбутись описаних недоліків було зроблено чимало спроб розробити децентралізовану електронну валюту. Перші електронні валюти дозволяли однорангове використання, але вимагали посередництво

банку для верифікації коректності токену та транзакції(наприклад, що токен не було витрачено двічі).

Важливою роботою було електронна валюта Шаума, яка була описана у [11]. Вона дозволяла виконувати операції однорангово, гарантувала приватність через використання сліпого підпису, але все одно потребувала взаємодії з банком для створення токену чи отримання грошового еквіваленту.

Поява блокчейн-технологій повністю вирішила проблеми централізованого керування. Усі учасники мережі, за допомогою механізмів розподіленого консенсусу, виконують функції банку, як контролюючого органу.

Використання криптовалют на основі блокчейн-технологій забезпечують

1. Приватність
2. Децентралізованість транзакцій, тобто відсутність потреби у посереднику для виконання довільних операцій
3. Високу швидкість проведення платежів, незалежно від географічного положення відправника та отримувача

Серед недоліків можна визначити:

1. Об’єм ресурсів, що потрібні для того, щоб бути повноцінним учасником мережі(потрібно зберігати копію всього блокчейну)
2. Функціонування мережі(в тому числі час підтвердження транзакцій) залежить від кількості активних учасників.

1.3 Мережа Bitcoin

Мережа Bitcoin являє собою першу реалізацію криптовалюти, що використовувала блокчейн технологію. Кожен блок складається з заголовку,

що містить службову інформацію та тіла, що складається зі списку транзакцій.

Для спрощення верифікації конкретних транзакцій, заголовок блоку містить хеш кореня дерева Меркле, що побудоване на списку транзакцій блоку.

Цифровий підпис. Для підписання транзакцій використовується асиметрична криптографія з використанням еліптичних кривих, зокрема використовується еліптична крива secp256k1 та алгоритм ECDSA. Дана крива визначена на скінченному полі і дозволяє ефективно проводити обчислення.

Адреси. Для виконання операцій, кожен учасник мережі створює аккаунт, який може мати необмежену кількість пар ключів. Ідентифікатором у мережі Біткойн, що дозволяє створювати транзакції слугує адреса. Адреса визначається з публічного ключу, за допомогою застосування хеш-функції RIPEMD-160. Так, адреса має розмір 20 байт і найчастіше записується за допомогою кодування base58(для запобігання використання нерозрізнювальних літер) у вигляді 34 символів латинського алфавіту.

Розумні контракти. Для контролю за правом розпоряджатись токенами, використовується концепція блокуючих та розблокуючих скриптів мовою Bitcoin Script. Мова складається з невеликої кількості операційних кодів та не є Тьюринг-повною, адже у ній відсутні цикли. Інтерпретатор даної мови використовує стекову модель.

Транзакції. Структуру транзакції задає набір входів транзакції, набір виходів транзакції, та певна службова інформація. Транзакції бувають різних видів і класифікуються в залежності від умов, що потрібно виконати, аби витрати кошти, що переказуються. Так, класичною транзакцією є P2PKH - тобто переказ токенів на задану адресу. Для реалізації більш складних технік

контролю доступу, використовуються транзакції типу P2SH, що дозволяють використання довільних блокуючих скриптів.

Баланс. Баланс кожної адреси не зберігається у блокчейні, а перераховується за запитом з публічної інформації. Кількість токенів, що належить певній адресі визначається сумарним об'ємом невитрачених виходів транзакцій(UTXO), у яких дана адреса вказана в якості отримувача.

1.4 Мережа Ethereum

Мережа Ethereum з'явилась пізніше, ніж мережа Біткойн, і має багато спільного з нею. Основною відмінністю є те, що Ethereum є не просто імплементацією криптовалюти, а ще й платформою для створення довільних децентралізованих додатків. Для досягнення даної мети, розробники Ethereum додали підтримку розумних контрактів, як об'єктів першого класу.

Адреси. У мережі існує два різних типи адрес: адреси, що належать користувачам, як у Біткойні, та адреси, на яких працюють смарт-контракти. Дані адреси підтримують усі ті ж можливості, що і адреси користувачів - вони можуть виступати отримувачами у транзакції, можуть ініціювати транзакцію, тощо. Крім цього, до даних адрес можна робити запити, що дозволяють виконання функцій контракту.

Розумні контракти. Для розробки розумних контрактів для платформи Ethereum використовується чимало мов програмування, зокрема найбільш поширена - мова Solidity. Дана мова є Тьюринг-повною. Середовищем виконання розумних контрактів даної платформи є віртуальна машина EVM.

РОЗДІЛ 2. ТЕОРЕТИЧНЕ ОБҐРУНТУВАННЯ

2.1 Хешовані смарт-контракти з часовою затримкою

Для розпоряджання токенами в блокчейн, тобто ініціації транзакції, використовуються механізм смарт-контрактів. В межах моделі, контракти можна розглядати, як набір умов, що повинні бути виконаними, для того щоб створити транзакцію, що витрачає дані токени і яка буде визнана рештою мережі, як коректна. Так як при додаванні нової інформації в блокчейн, транзакції верифікуються автоматично, умови розумних контрактів задаються у вигляді вихідного коду на спеціалізованих мовах програмування і перевіряються вузлами мережі.

Для розробки алгоритму атомарного обміну криптовалютами використаємо хешовані смарт-контракти з часовою затримкою.

2.1.1 Загальний огляд та вимоги

Хешовані смарт-контракти з часовою затримкою - це контракт, що блокує токени та надає два і тільки два можливих способи розблокування:

1. Отримувачем, за умови надання значення s такого, що
   $has\square(s) = \square s$, де hs - значення, що вказано у контракті
2. Відправником, за умови, що з моменту публікації контракту, пройшов проміжок часу, що вказано у контракті.

Хешовані смарт-контракти з часовою затримкою параметризуються набором значень:

1. Ідентифікатор(адреса) відправника
2. Ідентифікатор(адреса) отримувача
3. Кількість токенів
4. Цільове значення хеш-функції
5. Довжина секретного значення

6. Величина часової затримки

Ідея використання хешованих смарт-контрактів з часовою затримкою описана, наприклад, у роботі [12].

2.1.2 Реалізація для мережі Bitcoin

Розглянемо реалізацію хешованих смарт контрактів з часовою затримкою для мережі, що працює за протоколом Bitcoin. Як було згадано у розділі 1, у даній мережі в якості механізму імплементації розумних контрактів використовуються блокуючі скрипти, які описують умови, що має задовольнити отримувач у відмикаючому скрипті, щоб довести право розпоряджатись даними токенами. Стандарт імплементації блокуючого скрипта описано у протоколі покращення Біткойну №199(BIP-199[13]). З моменту публікації 27 березня 2017 року, до скрипту додалась вимога додатково перевіряти довжину секретного значення, а не лише значення хеш-суми. Так при створенні контракту, ініціатор транзакції вказує довжину секретного значення.

Розглянемо визначення кодів операцій мови програмування Bitcoin Script, що використовуються для реалізації даного смарт контракту, згідно з документацією[14].

OP_IF, OP_ELSE, OP_ENDIF використовуються для задання розгалуження виконання скрипту.

OP_IF переглядає значення вершини стеку даних, інтерпретує його як логічне значення та на основі результату, дозволяє чи не дозволяє виконання наступного блоку кодів.

OP_ELSE дозволяє виконання наступного блоку операцій, якщо попередній OP_IF отримав хибне значення перевірки і не був виконаним.

OP_ENDIF використовується для завершення блоку розгалуження та повернення до послідовного виконання.

OP_SIZE переглядає значення верхівки стеку, не вилучаючи його, та додає до стеку довжину зчитаного значення.

OP_SHA256 вилучає значення з вершини стеку і додає значення його хеш-суми, з використанням хеш-функції SHA256.

OP_EQUALVERIFY вилучає два значення з вершини стеку та порівнює їх. Якщо значення відрізняються, виконання скрипту завершується з помилкою.

OP_DUP додає до стеку копію значення, що знаходиться у його вершині.

OP_HASH160 вилучає значення з вершини стеку і додає значення його хеш-суми, з використанням хеш-функції RIPEMD160.

OP_CHECKLOCKTIMEVERIFY - вилучає значення з вершини стеку, інтерпретуючи його, як мінімальну позначку часу, починаючи з якої транзакція вважається коректною. Якщо значення nLockTime в транзакції не є більшим ніж значення, отримане зі стеку, виконання скрипту закінчується помилкою.

OP_DROP вилучає значення з вершини стеку.

OP_CHECKSIG - вилучає два значення з вершини стеку, інтерпретуючи їх як публічний ключ та цифровий підпис. Для транзакції, включаючи входи, виходи та скрипт. Якщо значення підпису, отримане зі стеку, є коректним підписом для даної хеш-функції, зробленим з використанням приватного ключа, що відповідає публічному ключу, отриманому зі стеку, до стеку додається значення 1. Якщо дана умова не виконується до стеку додається значення 0.

OP_TRUE - додає до стеку значення, що інтерпретується як логічна істина.

Остаточно, блокуючий скрипт контракту має вигляд:

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

Розглянемо детально наведений скрипт. Так на кроці 1, в залежності від значення на вершині стеку даних, виконується одна з гілок розгалуження. Якщо це значення інтерпретується як істина, виконується перевірка з використанням образу хешу, вказаного в скрипті. Якщо ж це значення інтерпретується як фальш, виконується перевірка часового блоку. В обох випадках, останнім кроком виконується перевірка електронно-цифрового підпису транзакції. У випадку розблокування за допомогою образу хешу, перевірка поверне істинне значення лише, якщо для створення підпису було використано приватний ключ, що відповідає адресі отримувача(вказаній у скрипті на позиції 10). Аналогічно, за умови розблокування після завершення дії часової затримки, для розблокування контракту необхідно використовувати приватний ключ, що відповідає адресі відправника(вказаній у скрипті на позиції 17).

Розглянемо два можливі випадки розблокування коштів.

1. Одержувачем за допомогою секретного значення. Відповідний розблоковуючий скрипт має вигляд:

1. <підпис>
2. <публічний ключ>
3. <кандидат секретного значення>
4. OP_TRUE

2. Відправником, за умови завершення часової затримки. Відповідний розблоковуючий скрипт має вигляд:

<підпис>
<публічний ключ>
OP_FALSE

Розглянемо виконання смарт контракту для першої ситуації. На першому кроці виконується конкатенація блокуючого скрипту з розблокуючим, і отриманий скрипт інтерпретується за допомогою стекової моделі. Так, результатом конкатенації є скрипт:

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

20. OP_HASH160
21. <адреса відправника>
22. OP_ENDIF
23. OP_EQUALVERIFY
24. OP_CHECKSIG

На підготовчому етапі, стек даних є порожнім.

Крок 1: Значення <підпис> складається на стек даних. Стан інтерпретатора:

Стек: <підпис>

Кроки 2-4 виконуються аналогічно. Стан інтерпретатора:

Стек: <підпис> <публічний ключ> <кандидат секретного значення> 1

Крок 5: З верхівки стеку вилучається значення 1, виконується позитивна гілка розгалуження(кроки 6-14).

Стек: <підпис> <публічний ключ> <кандидат секретного значення>

Крок 6: Переглядається значення у вершині стеку - <кандидат секретного значення>. До стеку записується довжина отриманого значення.

Стек: <підпис> <публічний ключ> <кандидат секретного значення> <довжина кандидату секретного значення>

Крок 7: Значення <довжина секретного значення> складається на стек даних.

Стек: <підпис> <публічний ключ> <кандидат секретного значення> <довжина секретного значення><довжина секретного значення>

Крок 8: З вершини стеку вилучаються два значення - <довжина кандидату секретного значення> та <довжина секретного значення>, та порівнюються між собою. За умови, що дані значення співпадають виконання продовжується.

Стек: <підпис> <публічний ключ> <кандидат секретного значення>

Крок 9: З верхівки стеку вилучається значення <кандидат секретного значення> і на стек додається значення хеш-функції SHA 256, застосоване до вилученого значення.

Стек: <підпис> <публічний ключ> <хеш кандидату секретного значення>

Крок 10: Значення <хеш секретного значення> складається на стек даних.

Стек: <підпис> <публічний ключ> <хеш кандидату секретного значення> <хеш секретного значення>

Крок 11: З вершини стеку вилучаються два значення - <хеш кандидату секретного значення> та <хеш секретного значення>, та порівнюються між собою. За умови, що дані значення співпадають виконання продовжується.

Стек: <підпис> <публічний ключ>

Крок 12: Значення <публічний ключ> подвоюється на вершині стеку

Стек: <підпис> <публічний ключ> <публічний ключ>

Крок 13: З верхівки стеку вилучається значення <публічний ключ>, до зчитаного значення застосовується хеш-функція RIPEMD-160, та отримане значення(яке згідно структури мережі блокчейн є адресою, що відповідає даному публічному ключу) додається до стеку.

Стек: <підпис> <публічний ключ> <адреса>

Крок 14: Значення <адреса отримувача> складається на стек даних.

Стек: <підпис> <публічний ключ> <адреса> <адреса отримувача>

Кроки 15-22 не змінюють стан інтерпретатора.

Крок 23: З вершини стеку вилучаються два значення - <адреса> та <адреса отримувача>, та порівнюються між собою. За умови, що дані значення співпадають виконання продовжується.

Стек: <підпис> <публічний ключ>

Крок 24: З вершини стеку вилучаються два значення - <підпис> та <публічний ключ>. Для транзакції обчислюється значення хешу, виконується верифікація цифрового підпису. З криптографічних властивостей електронно-цифрового підпису, виконання скрипту закінчується коректно за виконання наступних умов:

1. Вихідна транзакція не була модифікована з моменту підписання.

2. Для створення підпису було використано приватний ключ, який відповідає вказаному публічному ключу.

Отже, за умови використання всіх правильних даних, інтерпретатор успішно завершує виконання скрипту, і транзакція вважається розблокованою.

Розглянемо можливі атаки на даний контракт:

1. Неправильна структура розблокуючого скрипту.

2. Неправильне секретне значення. За умови використання неправильного секретного значення, можливі два варіанти:

2а. Довжина використаного значення відрізняється від вказаної у блокуючому скрипті. Тоді, виконання скрипту завершується з помилкою на кроці 8.

2б. Довжини використаного та істинного секретного значення співпадають. Виконання скрипту завершується з помилкою на етапі перевірки значень хеш-функції(на кроці 11).

3. Неправильне значення публічного ключу. У даному випадку виконання скрипту завершується з помилкою на етапі перевірки значень адреси отримувача, вказаної ініціатором транзакції у блокуючому скрипті та значення адреси отриманої шляхом застосування хеш-функції RIPEMD-160 до вказаного публічного ключу(на кроці 23).

4. Неправильне значення підпису. Тут можливі наступні варіанти:

4а. Підписано неправильно сформовану, модифіковану, чи підмінену транзакцію.

4б. Для створення підпису використано неправильний ключ.

В усіх випадках, невідповідність виявляться на кроці 24.

Отже, отримувач може розблокувати кошти лише за умови вказання правильного секретного значення та за умови володіння приватним ключем, що відповідає адресі отримувача, вказаній у контракті, що означає що дана реалізація задовольняє вимогам, поставленим до хешованих смарт-контрактів з часовою затримкою, сформованим у розділі 1.

Розглянемо виконання смарт контракту для даної ситуації.

Так, результатом конкатенації є скрипт:

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

Крок 1: Значення <підпис> складається на стек даних. Стан інтерпретатора:

Стек: <підпис>

Кроки 2-3 виконуються аналогічно. Стан інтерпретатора:

Стек: <підпис> <публічний ключ> 0

Крок 4: З верхівки стеку вилучається значення 0, виконується негативна гілка розгалуження(кроки 15-20).

Стек: <підпис> <публічний ключ>

Крок 15: Значення <позначка часу> складається на стек даних. Стан інтерпретатора:

Стек: <підпис> <публічний ключ> <позначка часу>

Крок 16: З верхівки стеку вилучається значення <позначка часу> і порівнюється з позначкою часу, що відповідає мінімальному моменту активації транзакції. Якщо час перевірки транзакції менший за вилучене зі стеку значення, виконання скрипта завершується з помилкою. Інакше, до стеку додається значення 1.

Стек: <підпис> <публічний ключ> 1

Крок 17: З верхівки стеку вилучається значення 1.

Стек: <підпис> <публічний ключ> 1

Крок 18: Значення <публічний ключ> подвоюється на вершині стеку

Стек: <підпис> <публічний ключ> <публічний ключ>

Крок 19: З верхівки стеку вилучається значення <публічний ключ>, до зчитаного значення застосовується хеш-функція RIPEMD-160, та отримане значення(яке згідно структури мережі блокчейн є адресою, що відповідає даному публічному ключу) додається до стеку.

Стек: <підпис> <публічний ключ> <адреса>

Крок 20: Значення <адреса відправника> складається на стек даних.

Стек: <підпис> <публічний ключ> <адреса> <адреса відправника>

Крок 23: З вершини стеку вилучаються два значення - <адреса> та <адреса відправника>, та порівнюються між собою. За умови, що дані значення співпадають виконання продовжується.

Стек: <підпис> <публічний ключ>

Крок 24: З вершини стеку вилучаються два значення - <підпис> та <публічний ключ>. Для транзакції обчислюється значення хешу, виконується верифікація цифрового підпису, з аналогічними властивостями, як і при виконанні іншого розблокуючого сценарію.

Спроба передчасно виконати контракт завершується аварійним виходом на кроці 16, спроба зняти кошти з використанням неправильної адреси - на кроці 23, при модифікації транзакції, чи спробі використати чужий публічний ключ - на кроці 24.

Отже, наведена реалізація відповідає вимогам хешованого смарт-контракту з часовими затримками.

2.1.3 Реалізація для мережі Ethereum

Для мережі Ethereum, смарт-контракти реалізуються за допомогою мови програмування Solidity. У мережі Ethereum, розумні контракти визначаються за допомогою бінарного коду(результату компіляції), та інтерфейсу контракту.

Інтерфейс складається з модифікуючих функцій(сеттери) та функцій що не змінюють стан контракту, а лише зчитують його інформацію(геттери). Два типи методів розрізняються за шляхом виконання. Геттери виконуються миттєво, так як не потребують зміни стану контракту, а отже не потребують запису до блокчейну. Виконання функцій сеттерів є відкладеним, адже через зміну стану контракту, виклик функції та новий стан контракту мають бути опублікованими на блокчейні, а отже - мають бути включеними до транзакції, та пройти шлях включення до блоку та додавання нового блоку до блокчейну.

Сформуємо інтерфейс, який має задовольняти смарт контракт в Ethereum, щоб відповідати вимогам означення хешованих смарт-контактів з часовою затримкою.

Для забезпечення двох можливих способів розблокування контракту достатньо створити інтерфейс контракту, що містить лише два модифікуючі методи - redeem для розблокування з використанням секретного значення та refund для повернення коштів ініціатору транзакції.

Лістинг коду реалізації смарт-контракту для платформи Ethereum наведено у додатку 1.

2.2 Атомарний обмін криптовалютами

Визначимо поняття атомарності для обміну криптовалютами. В загальній постановці завдання обміну криптовалютами, в обміні може брати участь довільна кількість учасників, а структура обміну задається за допомогою орієнтованого зваженого графу D(V, E). Множину вершин V задають учасники обміну, множину ребер E - факти бажаного обміну токенів різних блокчейнів.

Атомарний протокол обміну гарантує:

- якщо всі учасники слідують протоколу, всі обміни відбуваються;
- якщо існує коаліція, що порушує правила протоколу, ті учасники, що чинили чесно, не отримують збитків.
- не існує раціональної причини відхилятись від існуючого протоколу.

Загальна постановка завдання розробки атомарного протоколу обміну токенами досліджена у [15].

У даній роботі розглянемо обмін криптовалютами між двома учасниками, як найбільш розповсюджений випадок. Для даного випадку визначення атомарності має наступний вигляд:

Атомарний протокол обміну для двох учасників гарантує:

- Якщо обидва учасники слідують протоколу, обидва обміни відбуваються.
- Якщо один з учасників порушує протокол, інший не втрачає свої кошти.
- Жодному з учасників невигідно порушувати протокол.

2.3 Алгоритм атомарного обміну

Розглянемо протокол, що дозволяє досягти атомарності для обміну криптовалютами між двома учасниками.

Нехай Аліса має 1 токен блокчейну 1, а Боб - 1 токен блокчейну 2.

1. Аліса генерує секретне значення s і обчислює значення хеш-функції $\square as\square(s) = \square s$, не розголошуючи значення s.
2. Аліса публікує в блокчейні 1 хешований смарт контракт з часовою затримкою. В умовах контракту, Боб може отримати 1 токен блокчейну 1, за умови, що він назве таке значення s', що $\square as\square(s') = \square s$. Якщо Боб не називає таке значення, то через час t_1, Аліса може отримати свій токен назад.
3. Боб перевіряє контракт, який був опублікований Алісою на коректність.
4. Боб публікує аналогічний контракт на блокчейні 2. В умовах контракту, Аліса може отримати 1 токен блокчейну 2, за умови, що вона назве таке значення s'', що $\square as\square(s'') = \square s$, де hs Боб дізнається з контракту 1. Якщо Аліса не називає таке значення, то через час t_2, Боб може отримати свій токен назад.
5. Аліса використовує значення s у контракті Боба, отримує токен блокчейну 2 та розголошує значення s Бобу.

6. Боб використовує значення s вказане Алісою, використовує його для розблокування контракту Аліси та отримує токен блокчейну 1.

Схематично алгоритм обміну зображено на рисунку 2.2.

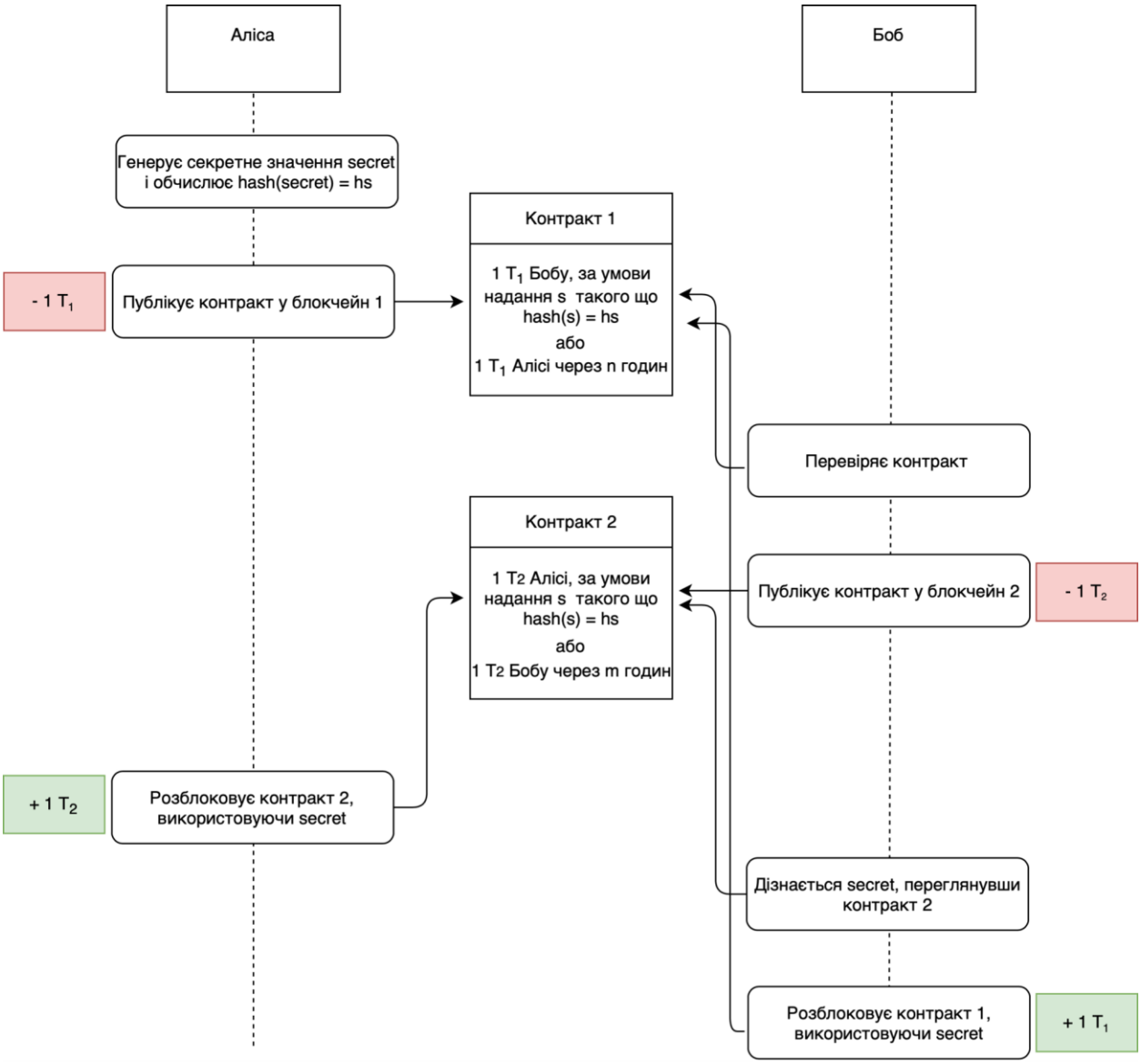

Рисунок 2.2 Схема атомарного обміну токенами

Доведемо, що наведений алгоритм є алгоритмом атомарного обміну.

1. Якщо всі учасники дотримуються протоколу, обидва обміни відбуваються.

   Справедливість даного пункту забезпечується детермінованістю значення хеш-функції та відкритістю інформації у блокчейні. Після

того, як Аліса дотримуючись протоколу розблокувала контракт Боба, Боб має усю інформацію для проведення другого етапу обміну.

2. Якщо один з учасників порушує протокол, інший не втрачає свої кошти.

   Розглянемо можливі варіанти порушення протоколу:

   а. Нехай Боб порушує умови на кроці 4 і не публікує свій контракт з зобов'язаннями. За властивостями хеш-функції, задача підбору секретного значення s, такого, що може розблокувати контракт Аліси, є обчислювально складним завданням, і не може бути виконаним у термін t_1. Тоді через час t_1 Аліса повертає свої кошти.

   б. Нехай Аліса порушує умови і не розблоковує контракт Боба. Тоді через час t_2 Боб повертає свої кошти.

   в. Нехай Аліса намагається отримати і кошти Боба і повернути свої. Тоді, вона розблоковує контракт Боба у останній момент, до того, як він зможе повернути свої кошти. За умови, що час блокування контракту Аліси t_1 приблизно дорівнює t2, Боб може не встигнути скористатись значенням Аліси для розблокування контракту. Це накладає умову, $t_1 >= t_2 + \Delta t$, де Δt - час публікації контракту на блокчейні, для забезпечення коректності алгоритму. Ця перевірка включається на етапі перевірки контракту Бобом на кроці 3. За виконання даної умови, атака не є можливою.

3. Жодному з учасників невигідно порушувати протокол.

   Як було розглянуто у попередньому пункті, жоден з учасників не може отримати чужі кошти у випадку порушення протоколу. Єдиний вихід, при відхиленні від протоколу - відповідний обмін не відбувається, але якщо обмін не є вигідним одній зі сторін, раціонально не брати участь

у обміні взагалі. Тому, обидвом сторонам раціонально дотримуватись алгоритму.

Отже, наведений алгоритм є атомарним протоколом обміну криптовалютами.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ ДЛЯ ПРОВЕДЕННЯ АТОМАРНОГО ОБМІНУ

3.1 Діаграма прецедентів

На першому етапі проектування, необхідно визначити можливі сценарії використання програмного забезпечення, дослідивши алгоритм описаний у розділі 2.2. На основі даного аналізу було розроблено UML діаграму прецедентів, що наведено на рисунку 3.1.

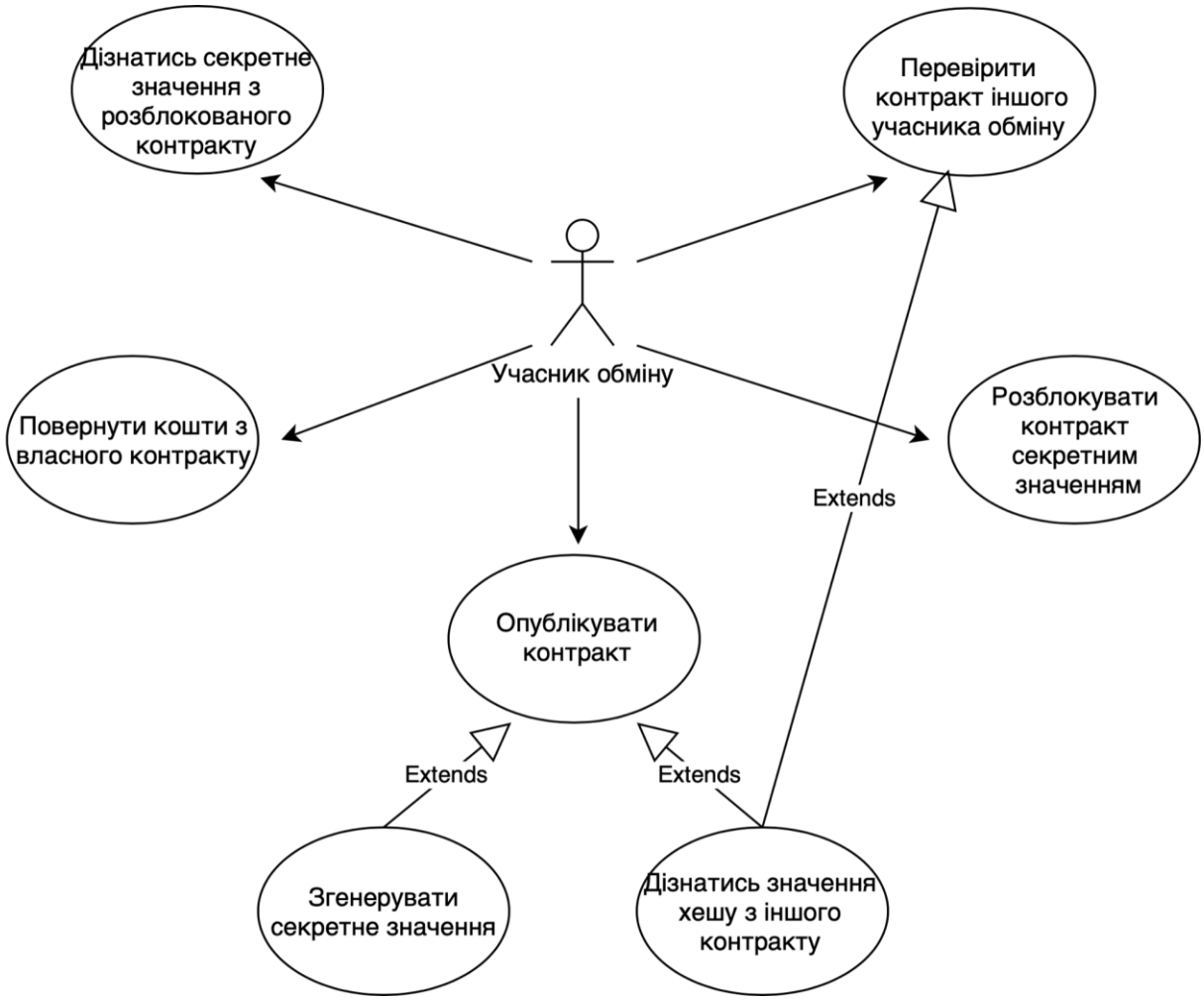

Рисунок 3.1 UML діаграма прецедентів для застосування програмного забезпечення для атомарного обміну криптовалютами

Програмне забезпечення буде розроблятись у вигляді консольного додатку. На основі даної діаграми можна легко спроектувати користувацький інтерфейс.

Інтерфейс додатку складатиметься з таких методів:

1. create-htlc - створення та публікації контракту до вибраного блокчейну.
2. redeem-htlc - розблокування контракту з використанням секретного значення.
3. refund-htlc - повернення коштів контракту після спливання часового блоку контракту.

4. extract-secret - визначення секретного значення, що було використано для розблокування контракту.
5. verify-htlc - перевірка умов опублікованого контракту.

3.2 Взаємодія з користувачем

Розглянемо можливі способи взаємодії користувачів з розробленим програмним забезпеченням(консольним додатком ACCS).

Найбільш поширеними сценаріями використання додатку є успішний обмін токенами між двома учасниками та неуспішний обмін з подальшим поверненням коштів ініціатором обміну.

Нехай Аліса та Боб намагають обмінятись токенами блокчейнів Bitcoin(BTC) та Ethereum(ETH). Аліса ініціює обмін, використовуючи свій токен BTC. Для простоти введемо наступні позначення: адреси Боба у відповідних мережах позначимо BobBTC та BobETH, відповідно адреси Аліси - AliceBTC та AliceETH. Всі ідентифікатори транзакцій позначатимемо transactionID, час позначимо у одиницях часу, де одиниця більше часу публікації транзакції до блокчейну. На рисунку 3.2 зображено діаграму послідовності для успішного обміну 1 токену BTC на 1 токен ETH між Алісою та Бобом у наведених позначеннях.

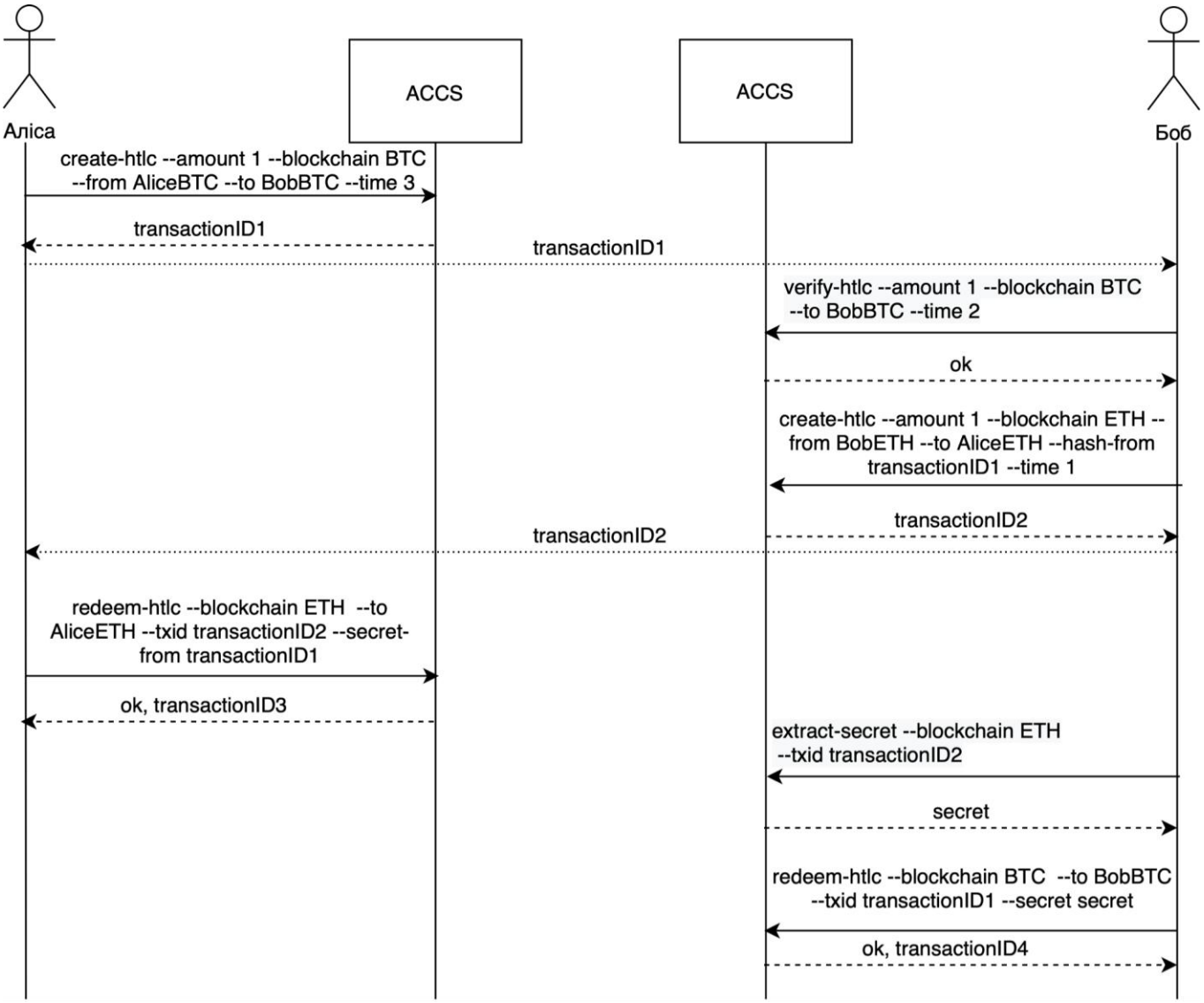

Рисунок 3.2 UML діаграма послідовності застосування програмного забезпечення для успішного атомарного обміну криптовалютами

На рисунку 3.3 зображено діаграму послідовності для неуспішного обміну між Алісою та Бобом у наведених позначеннях. В даному сценарії, Боб не виконує свою частину протоколу, і Аліса через дві одиниці часу, повертає свої кошти.

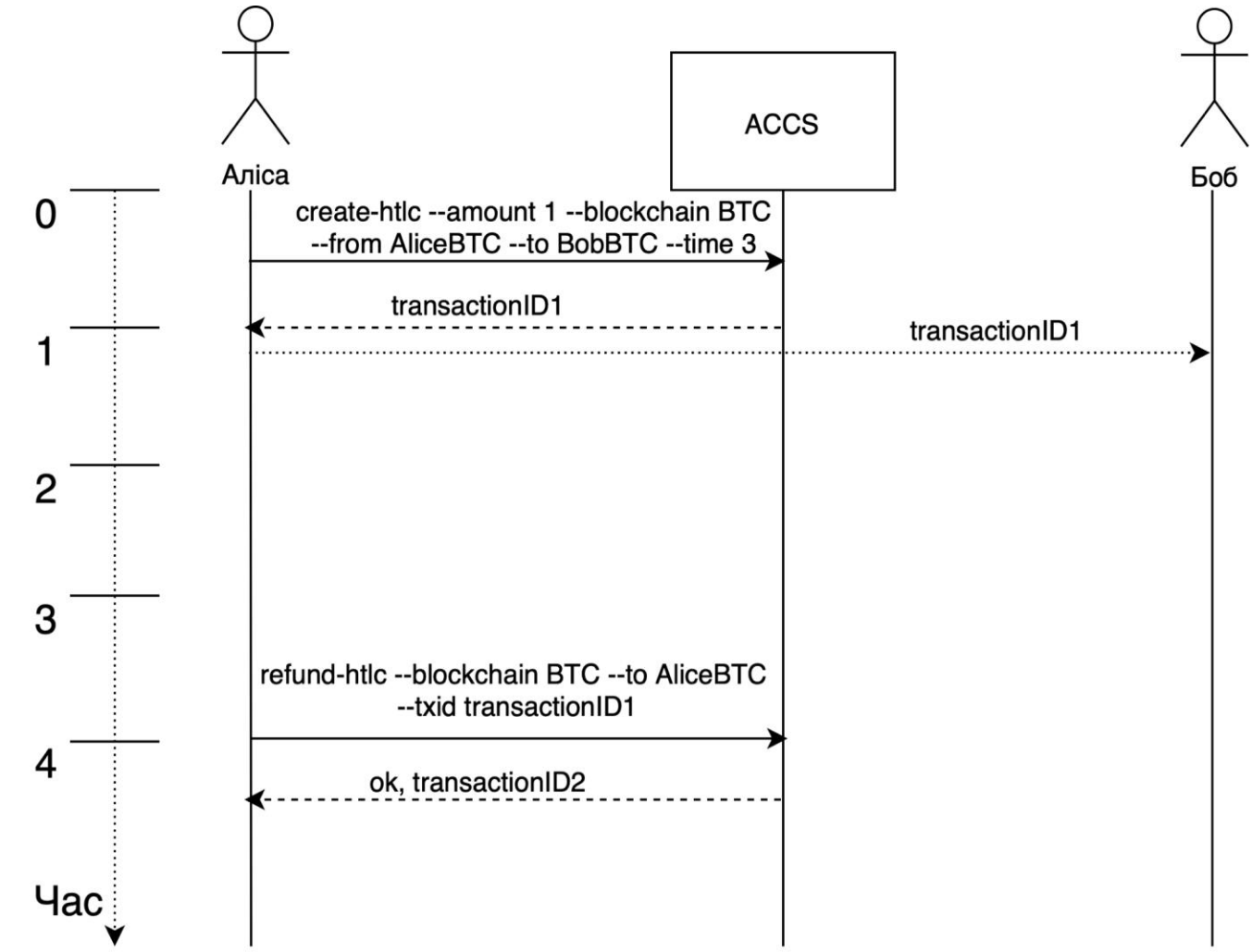

Рисунок 3.3 UML діаграма послідовності застосування програмного забезпечення у випадку повернення коштів ініціатором атомарного обміну криптовалютами

3.3 Інструменти реалізації

Для реалізації програмного забезпечення було обрано мову програмування Golang.

З одного боку, мова є компільованою, тобто не вимагає великих витрат на інтерпретацію та наявність віртуальної машини, що дозволяє зберегти велику ефективність та портативність програмного забезпечення.

З іншого боку, існують офіційні реалізації вузлів мереж Ethereum та Bitcoin - geth та btcd, відповідно. Це дозволяє використовувати готові реалізації для примітивів відповідних блокчейнів - як-то структури транзакцій, стандарти кодування даних, чи відповідні константи. Також, важливими елементами імплементаціями вузлів є наявність середовища виконання розумних контрактів, що дозволяє налаштувати автоматизоване тестування реалізації контрактів.

Ще однією перевагою використання мови програмування Golang є наявність зручних бібліотек для написання користувацького інтерфейсу(у вигляді консольного додатку). Для написання програмного забезпечення було використано модуль Cobra[16], який є стандартом у даній сфері.

Для інтеграційного тестування розробленого програмного забезпечення було використано симуляційну мережу Bitcoin, тобто локальний вузол btcd, що працює у режимі simnet та симуляційну мережу Ethereum, яка реалізована в утиліті Ganache, і є частиною Truffle Suite.

В якості середовища розробки було використано середовище GoLand 2020 з плагіном для підтримки мови програмування Solidity.

РОЗДІЛ 4. ТЕСТУВАННЯ

Проведемо тестування розробленого програмного забезпечення для двох сценаріїв: успішного та неуспішного обміну криптовалютами.

Для тестування успішного обміну криптовалютами необхідно виконати такі кроки:

1. Запустити реалізацію вузла Біткойн та гаманець у режимі симуляції.
2. Запустити реалізацію вузла Ethereum у режиме симуляції.
3. Запустити два вікна терміналу: одне для симуляції дій Аліси, інше для дій Боба.
4. У вікні Аліси опублікувати контракт до блокчейну Bitcoin за допомогою команди create-htlc з часовою затримкою у 10 хвилин.
5. У консолі Боба перевірити контракт Аліси командою verify-htlc з правильним набором параметрів.
6. У вікні Боба опублікувати контракт до блокчейну Ethereum за допомогою команди create-htlc з часовою затримкою у 5 хвилин та значенням хешу з контракту Аліси.
7. У вікні Аліси розблокувати контракт Боба, використавши секретне значення за допомогою команди redeem-htlc.
8. У вікні Боба визначити секретне значення, що було використане Алісою, командою extract-secret.
9. У консолі Боба розблокувати контракт Боба, використавши секретне значення за допомогою команди redeem-htlc.

Результати тестування даного сценарію наведено на рисунках 4.1 та 4.2.

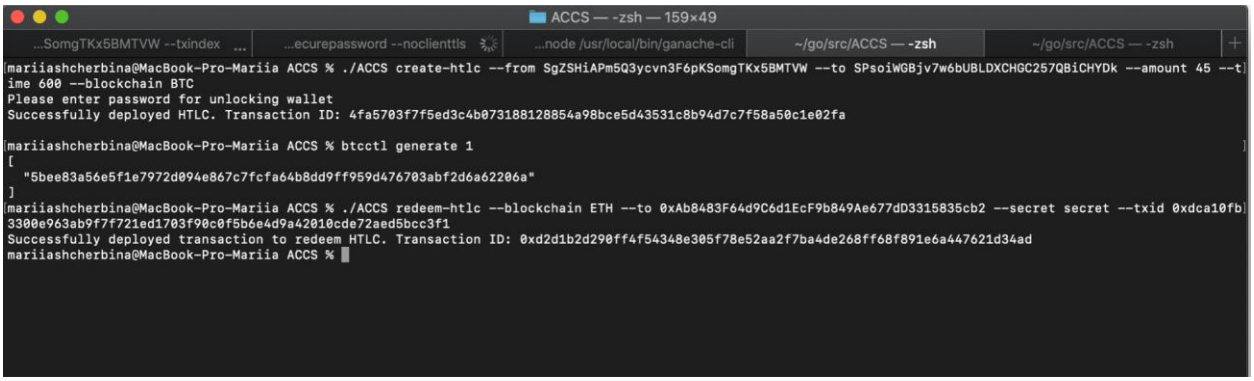

Рисунок 4.1 Тестування розробленого програмного забезпечення у випадку успішного обміну криптовалютами. Консоль Аліси.

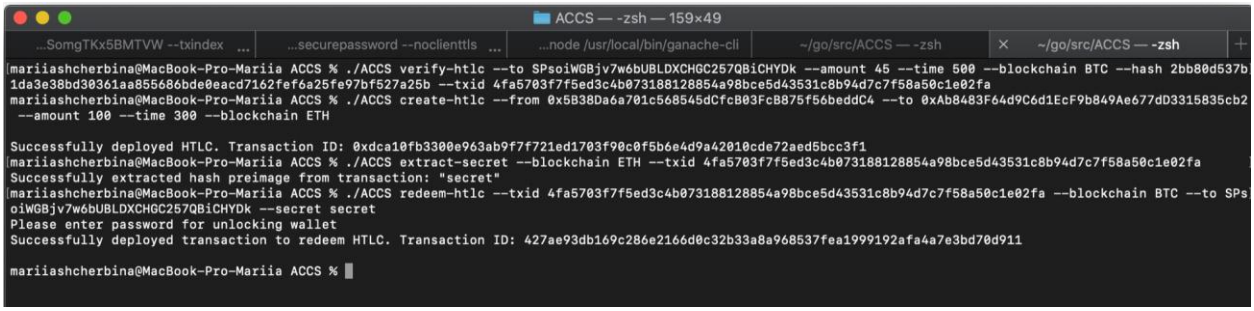

Рисунок 4.2 Тестування розробленого програмного забезпечення у випадку успішного обміну криптовалютами. Консоль Боба.

Для неуспішного обміну криптовалюти, використаємо другий сценарій з пункту 3.2, тобто відшкодування Алісою вартості свого розумного контракту.

Кроки для тестування:

1. Запустити реалізацію вузла Біткойн та гаманець у режимі симуляції.
2. Опублікувати контракт до блокчейну за допомогою команди create-htlc з часовою затримкою у 10 хвилин.
3. Просимулювати генерацію наступного блоку за допомогою команди btcctl generate 1.

4. Спробувати миттєво розблокувати даний контракт, і впевнитись, що програма не створює таку транзакцію.
5. Зачекати 10 хвилин.
6. Повторити спробу розблокувати контракт і впевнитись, що цього разу програма відпрацювала корректно.
7. Просимулювати генерацію наступного блоку за допомогою команди btcctl generate 1.
8. Перевірити транзакцію з повернення коштів у блокчейні.

Результати тестування даного сценарію наведено на рисунку 4.2.

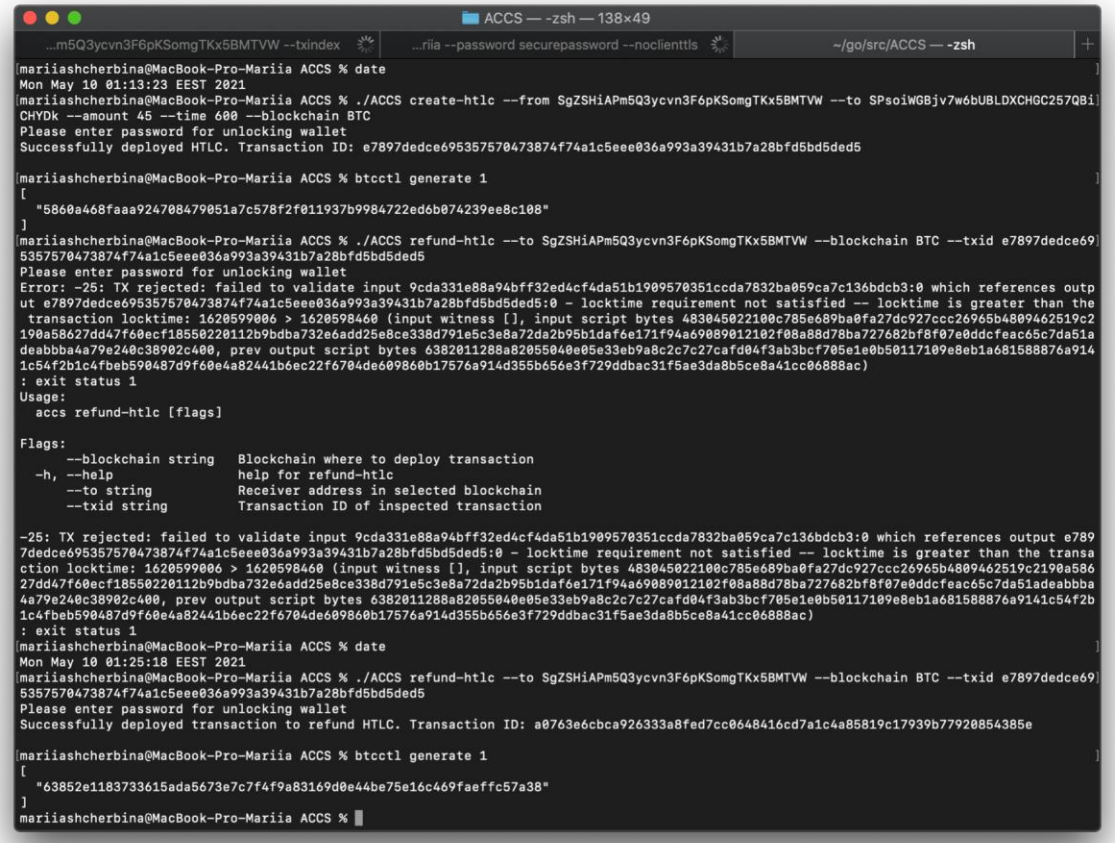

Рисунок 4.2 Тестування розробленого програмного забезпечення у випадку повернення коштів ініціатором атомарного обміну криптовалютами

ВИСНОВКИ

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

- поглиблено знання у криптографії та блокчейн-технологіях;
- досліджено існуючі системи та технології, що дозволяють проводити обмін токенами різних блокчейнів атомарно;
- розроблено та описано алгоритм проведення атомарного обміну криптовалютами;
- розроблено хешовані смарт-контракти з часовою затримкою для мереж Bitcoin та Ethereum;
- розроблено інтерактивну систему, що дозволяє провести атомарний обмін криптовалютами.
- протестовано розроблене програмне забезпечення, з використанням мережі у симуляційному та тестовому режимах;

Результатом дипломної роботи є програмне забезпечення для проведення атомарного обміну криптовалютами для токенів мереж Bitcoin та Ethereum.

ДОДАТОК А

Лістинг коду смарт-контракту для платформи Ethereum

HTLC.sol

```
pragma solidity >=0.7.0 <0.9.0;

/**
 * @title HTLC
 * @dev Hashed Time-Locked Contract
 */
contract HTLC {
    address payable sender;
    address payable receiver;
    uint256 amount;
    uint timeLockExpirationTimestamp;
    bytes secret;
    bytes32 hash;
    bool active;

    constructor(uint _timeLockExpirationTimestamp, address _receiver, bytes32
_hash) payable {
        sender = payable(msg.sender);
        receiver = payable(_receiver);
        amount = msg.value;
        hash = _hash;
        timeLockExpirationTimestamp = _timeLockExpirationTimestamp;
        active = true;
    }

    modifier TimeLockExpired {
        require(block.timestamp>=timeLockExpirationTimestamp);
        _;
    }

    // RequireActive використовується для визначення коректності викликів
методів, що модифікують стан контракту.
    // Після того, як контракт було розблоковано, він стає доступним
виключно для читання .
    modifier RequireActive {
        require(active);
        _;
    }
```

```
    // Дозволяє виконати операцію повернення коштів, за умови що термін часової затримки сплив і контракт досі активний
    function refund() TimeLockExpired RequireActive public payable {
        active = false;
        sender.transfer(amount);
    }

    // Дозволяє виконати розблокування контракту з використанням секретного значення за умови, що контракт перебуває в активному стані і вказане значення є коректним
    function redeem(bytes memory candidate) RequireActive public payable {
        require(sha256(candidate) == hash);
        active = false;
        secret = candidate;
        receiver.transfer(amount);
    }

    function getSecret() public view returns (bytes memory) {
        return secret;
    }

    function getReceiver() public view returns (address) {
        return receiver;
    }

    function getExpirationTimestamp() public view returns (uint) {
        return timeLockExpirationTimestamp;
    }

    function getAmount() public view returns (uint256) {
        return amount;
    }

    function getHash() public view returns (bytes32) {
        return hash;
    }
```

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Електронний ресурс] / Satoshi Nakamoto – Режим доступу до ресурсу: https://bitcoin.org/bitcoin.pdf.
2. Мережа Bitcoin [Електронний ресурс] – Режим доступу до ресурсу: https://bitcoin.org/ru/.
3. Мережа Ethereum [Електронний ресурс] – Режим доступу до ресурсу: https://ethereum.org/en/.
4. Мережа Stellar [Електронний ресурс] – Режим доступу до ресурсу: https://www.stellar.org.
5. Децентралізована біржа Uniswap [Електронний ресурс] – Режим доступу до ресурсу: https://app.uniswap.org/#/swap.
6. Міжланцюговий гаманець Liquality [Електронний ресурс] – Режим доступу до ресурсу: https://liquality.io.
7. Реалізація Bitcoin вузла мовою програмування Go [Електронний ресурс] – Режим доступу до ресурсу: https://pkg.go.dev/github.com/btcsuite/btcd.
8. Geth - офіційна реалізація Ethereum вузла мовою програмування Go [Електронний ресурс] – Режим доступу до ресурсу: https://geth.ethereum.org.
9. Офіційна документація Truffle Suite [Електронний ресурс] – Режим доступу до ресурсу: https://www.trufflesuite.com/docs.
10. Lightning Network [Електронний ресурс] – Режим доступу до ресурсу: https://lightning.network.
11. Chaum D. Blind signatures for untraceable payments [Електронний ресурс] / David Chaum – Режим доступу до ресурсу: http://www.hit.bme.hu/~buttyan/courses/BMEVIHIM219/2009/Chaum.BlindSigForPayment.1982.PDF.

12. Poon J. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments [Електронний ресурс] / J. Poon, T. Dryja – Режим доступу до ресурсу: https://lightning.network/lightning-network-paper.pdf#page=30.

13. Bowe S. BIP-199 [Електронний ресурс] / S. Bowe, D. Hopwood – Режим доступу до ресурсу: https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki.

14. Опис мови програмування Bitcoin Script [Електронний ресурс] – Режим доступу до ресурсу: https://en.bitcoin.it/wiki/Script.

15. Herlihy M. Atomic Cross-Chain Swaps [Електронний ресурс] / Maurice Herlihy – Режим доступу до ресурсу: https://arxiv.org/pdf/1801.09515.pdf.

16. Cobra [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/spf13/cobra.