НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»
УДК

«До захисту допущено»
Завідувач кафедри

_____ Дмитро ЛАНДЕ

“___”_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системи, технології та математичні методи кібербезпеки»

зі спеціальності 125 «Кібербезпека»

на тему: Підвищення рівня безпеки смарт-контрактів від шахрайства за рахунок використання реверсивних токенів

Виконав здобувач ступеня магістра ІІ курсу, групи ФБ-11мп

Топчій Микита Андрійович
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник Гальчинський Леонід Юрійович
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Здобувач ступеня магістра
_____
(підпис)

Київ – 2022 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський)
Спеціальність – 125 «Кібербезпека»
Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Дмитро ЛАНДЕ
(підпис)

«___»_____2022 р.

ЗАВДАННЯ

на магістерську дисертацію здобувачу ступеня магістра

Топчій Микити Андрійовича
(прізвище, ім'я, по батькові)

1. Тема дисертації Підвищення рівня безпеки смарт-контрактів від шахрайства за рахунок використання реверсивних токенів

науковий керівник дисертації Гальчинський Леонід Юрійович,
канд. техн.наук, доцент, доцент кафедри інформаційної безпеки,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «09» листопада 2022 р. № 4131-с

2. Термін подання здобувачем дисертації 06.12.2022 р.

3. Предмет дослідження Безпека смарт-контрактів у мережі Ethereum від шахрайської діяльності

4. Вихідні дані технічна документація, стандарти смарт-контрактів

5. Перелік завдань, які потрібно розробити

1. Проаналізувати предметну область. 2.Проаналізувати існуючі стандарти та реалізації смарт-контракти токенів ERC20/ERC721. 3.Виявити недоліки існуючих реалізацій. 4. Розробити модель стандарту, що б нівелював недоліки існуючих реалізацій. 5. Оцінити складнощі та недоліки реалізації реверсивних токенів. 6. Запропонувати модифікації, що здатні поліпшити користувацький досвід при використанні застосунків на основі контрактів реверсивних токенів

6. Орієнтовний перелік ілюстративного матеріалу 14 ілюстрацій

7. Орієнтовний перелік публікацій Топчій М., Гальчинський Л. (2022). Підвищення рівня безпеки смарт-контрактів від шахрайства за рахунок використання реверсивних токенів https://archive.logos-science.com/index.php/conference-proceedings/issue/view/6/6

8. Дата видачі завдання 01.09.2022

Календарний план

№	Назва етапів виконання	Термін виконання етапів	Примітка

з/п	магістерської дисертації	магістерської дисертації	
1	Аналіз предметної області. Ознайомлення із основами функціонування блокчейн-протоколів	01.09.2022	Виконано
2	Аналіз широковживаних стандартів токенів блокчейн-протоколу Ethereum	20.09.2022	Виконано
3	Виявлення недоліків існуючих стандартів та їх реалізації	05.10.2022	Виконано
4	Розробка моделі реверсивних токенів	30.10.2022	Виконано
5	Реалізація реверсивних токенів	30.11.2022	Виконано
6	Оформлення пояснювальної записки	05.12.2022	Виконано
7	Підготовка до захисту	13.12.2022	Виконано

Здобувач ступеня магістра _____ Микита ТОПЧІЙ

(підпис) (власне ім'я, ПРІЗВИЩЕ)

Науковий керівник _____ Леонід ГАЛЬЧИНСЬКИЙ

(підпис) (власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Робота обсягом 100 сторінок включає 14 ілюстрацій, 20 таблиць, 24 джерела літератури та 2 додатки.

Об’єктом дослідження є безпека смарт-контрактів у мережі Ethereum від шахрайства.

Предметом дослідження є модифікація наявних версій взаємозамінних та невзаємозамінних токенів для підвищення їх безпеки від шахрайства.

Методи дослідження - поєднання наявних реалізацій та стандартів токенів у мережі Ethereum.

Метою роботи є підвищення рівня захищеності смарт-контрактів від шахрайства у мережів Ethereum.

Результати роботи можуть бути використані для використання токенів при побудові різноманітних децентралізованих застосунків на основі смарт-контрактів.

Ключові слова: блокчейн, Ethereum, смарт-контракт, асиметрична криптографія, віртуальна машина

ABSTRACT

The work includes 98 pages, 14 illustrations, 20 tables, 23 bibliography references and 2 appendices.
The object of research is the security of smart contracts in the Ethereum network against fraud.

The subject of research is the modification of existing versions of fungible and non-fungible tokens to increase their security against fraud.

Research methods - a combination of existing implementations and token standards in the Ethereum protocol.

The purpose of the work is to increase the level of security of smart contracts against fraud in Ethereum protocol.

The results of the work can be used to use tokens when building various decentralized applications based on smart contracts.

Keywords: blockchain, Ethereum, smart contract, asymmetric cryptography, virtual machine

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

EIP - Ethereum Improvement Proposal, пропозиції по покращенню Ethereum

ERC - Ethereum request for comment

AMM - Automated Market Maker - автоматичний маркет-мейкер

VDF - Verifiable Delay Function

VRF - Verifiable Random Function

EOA - External Owned Account - аккаунт, що має приватний ключ

KYC - Know Your Customer - процес підтвердження особистості

DID - Decentralised Identification - децентралізована ідентифікація

FOMO - Fear of missing out - страх пропустити щось цікаве

DeFi - Decentralised Finance - децентралізовані фінанси

DEX - Decentralised Exchange - децентралізований обмінник

ETH - скорочення від Ethereum

API - Application Programming Interface

DApp - Decentralised Application - децентралізований застосунок

PoW - Proof of Work - консенсус на основі доказу виконаної роботи

PoS - Proof of Stake - консенсус на основі доказу володіння часткою

EVM - Ethereum Virtual Machine - віртуальна машина Ethereum

WASM - WebAssembly

ВСТУП

Розвиток технологій кожним роком набуває темпів, це неодмінно приносить людству величезні блага. Однак розвиток та впровадження тих чи інших технологій може займати тривалий час, допоки розробка вийде з вузьких кіл технічних спеціалістів у маси.
Так у 2008 році, коли було випущено технічний опис біткойну[15] світ побачив симбіоз розподілених систем обробки даних та криптографічних протоколів, що привів до появи платіжного протоколу у основі якого лежать ідеї рівності, незалежності, децентралізації.[5]

Хоча з тих пір пройшло не так вже й багато часу блокчейн-технології зробили величезний крок. Напевно, одним із найзначущих є смарт-контракти. Одним із перших, хто запропонував концепцію світового комп'ютера була команда Ethereum.
Дотепер сфера криптовалют досягла великих об'ємів. Однак це стало причиною того, що велика кількість шахраїв звернули на неї. Незважаючи на постійну роботу з покращення протоколів та стандартів мають місце випадки взломів та крадіжок. Для подальшого впровадження та розширення користувацького кругу криптовалют потрібно забезпечити достатній рівень безпеки, за якого користувачі зможуть без страху бути обдуреним їх використовувати у повсякденному житті.

У основі блокчейні лежить ідея постійності, тобто все, що туди було записане там і залишається. Опубліковані транзакції не можуть бути змінені, таким чином основна перевага накладає обмеження у випадку зловмисних дії, як от крадіжка. Так як не існує механізму, як у банках, що дозволив би повернути вкрадені кошти.

Про масштаби крадіжок говорять цифри: у 2020 році шкода від різноманітних атак була рівна 7.8 мільярдів доларів, вже у 2021 році цей показник зріс удвічі до 14 мільярдів.[4]

Таким чином вирішення цього питання є важливим для покращення стійкості до зловмисних дій та подальшого впровадження криптовалют у життя звичайних людей.

Для розуміння того яким чином функціонують блокчейн протоколи необхідно розглянути їх по частинах.

1 Теоретичні основи із сфери блокчейн технологій

1.1 Blockchain

Blockchain - це така структура даних, що складається з пов’язаних між собою блоків.[14] Кожен блок містить хеш попереднього таким чином можна відслідковувати цілісність даних. Кожен блок складається із транзакцій. Де транзакції - це записи, що містять інформацію про зміну стану. Наприклад, що користувач А надіслав користувачу В n монет. За допомогою блоків можна відновити стан системи у певний момент часу.

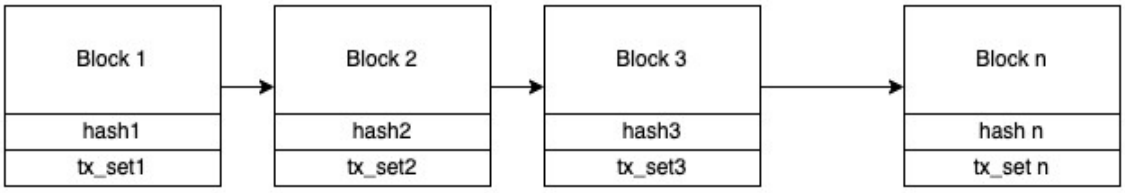


Рисунок 1.1 - Схематичне зображення структури блокчейну

1.2 DAG

Також у контексті блокчейну важливо відмітити таку структуру даних як DAG (Directed Acyclic Graph)[14].

Дана структура даних представляє собою направлений ациклічний граф. З його допомогою можна відслідковувати зміну станів. У кожного вузла даного графа є вхідні та вихідні ребра. Саме вони і показують як змінюється стан.

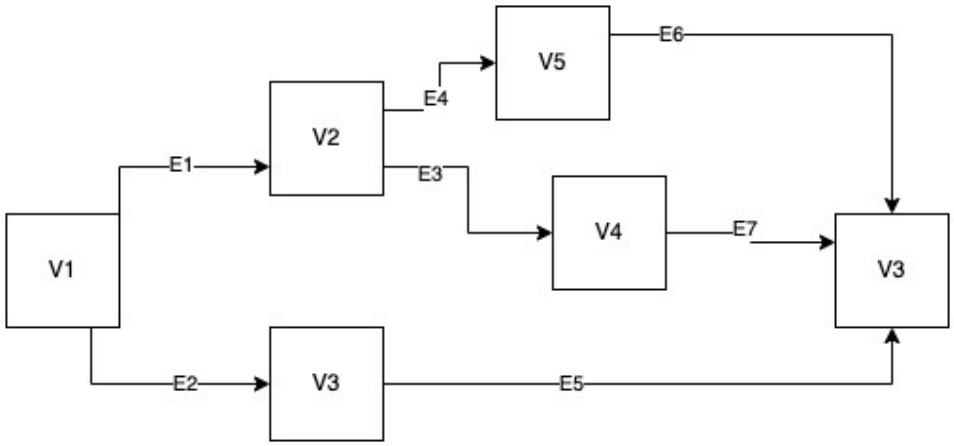


Рисунок 1.2 - Схематичне зображення структури DAG

1.3 Blockchain protocols

У основі блокчейн протоколів таких як Ethereum[6], Solana тощо. лежить така структура даних як блокчейн. Однак сама по собі вона доволі обмежена, тому поверх неї існує надбудова. Для прикладу буде розглянуто один з найпопулярніших протоколів Ethereum.

Умовно Ethereum можна розділити на три частини:

1. Мережевий рівень, рівень вузлів (нод)
2. Блокчейн
3. Віртуальна машина

Це спрощена модель однак на її основі доволі просто пояснити основні принципи функціонування протоколу.

1.3.1 Мережевий рівень, рівень вузлів

Даний рівень відповідає за взаємозв'язок окремо розташованих вузлів системи. Від нього залежить яким чином пакети даних будуть передаватись від одного вузла мережі до іншого.
На цьому рівні використовується велика кількість мережевих протоколів та алгоритмів. Наприклад, для розповсюдження повідомлень використовують сімейство gossip-алгоритмів, що є давно відомими та широко використовуються. За допомогою даного класу алгоритмів можна доволі швидко розповсюджувати повідомлення по мережі, адже кількість активних нод на кожному кроці зростає у експоненційній формі.

1.3.2 Блокчейн

Даний рівень відповідає за збереження даних. Кожний вузол має точну копію блокчейну. За рахунок цього система є відмовостійкою і вихід з ладу великої кількості вузлів мережі не вплине на актуальність та цілісність даних. Блоки у блокчейні отримуються з мережі, для чого використовується рівень вузлів та його алгоритми. Після отримання з мережі блоку, вузол перевіряє коректність транзакцій у ньому і у випадку її валідності вносить цей блок у свою локальну копію блокчейна. А мережею надсилає те, що він прийняв цей блок. Даний механізм називається консенсусом він займає важливе місце у розподілених системах, тому про нього мова йтиметься далі.

1.3.3 Рівень віртуальної машини

На поточному рівні виконуються усі дії пов'язані зі смарт контрактами. Саме віртуальна машина відповідальна за їх виконання. Наразі існує велика кількість віртуальних машин, що використовуються у блокчейн

протоколах, однак серед них можна виділити EVM - Ethereum Virtual Machine[6] та віртуальні машини на основі WASM (WebAssembly).
У даній роботі мова піде саме про EVM. Ethereum Virtual Machine - це стекова детермінована машина. Тобто для одного й того ж початкового стану при виконанні дій результат буде детермінований, такий що не залежить від вірогідності.
Для зміни стану виконуються транзакції. Блоки блокчейну зберігають транзакції. Таким чином маючи блокчейн та початковий стан віртуальної машини можна отримати кінцевий стан, застосовуючи транзакції із кожного блоку.

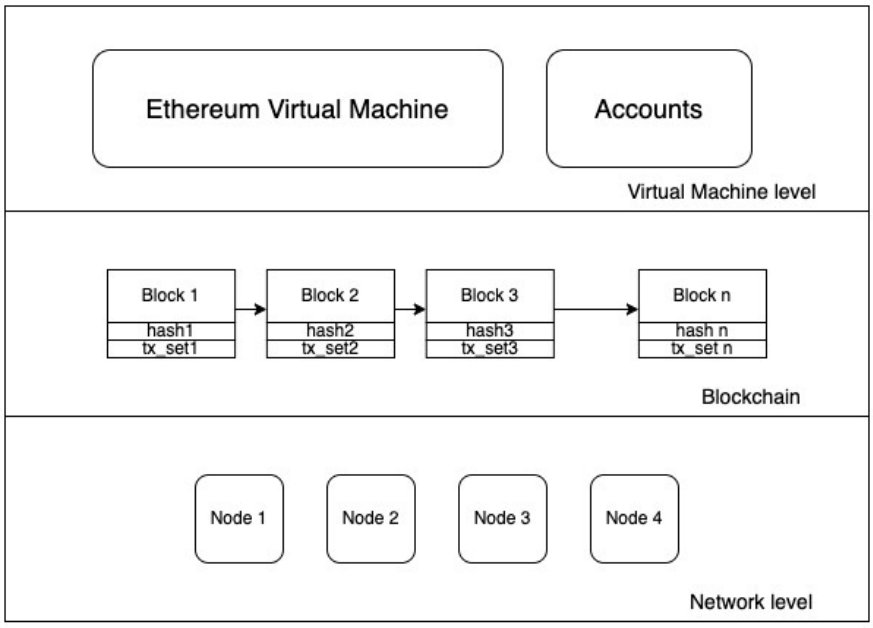


Рисунок 1.3 - Схематичне зображення рівнів блокчейн протоколу Ethereum

Зміну станів машини можна описати наступним чином[7].
Нехай Si - стан машини на і-му блоці.
Функція apply(S, TXs), така, що приймає два параметри - стан та транзакції.

Таким чином

$$Si+1 = apply(Si, tx\_seti);$$

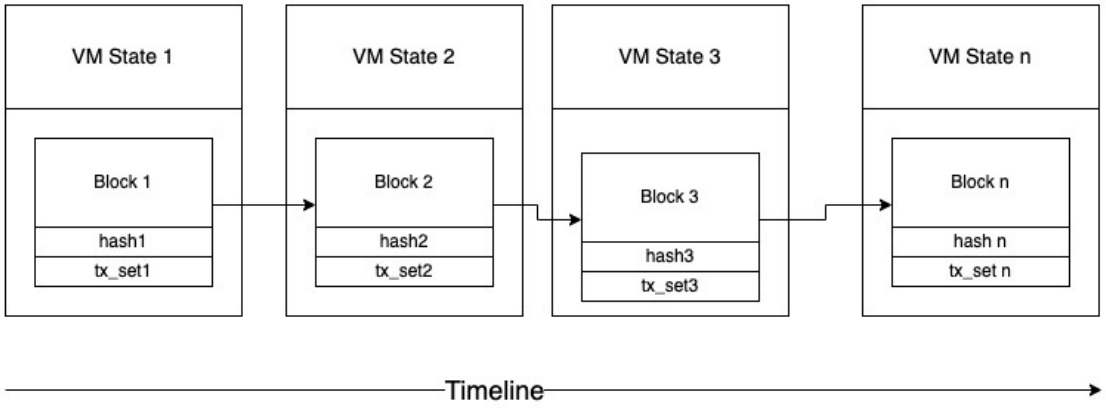


Рисунок 1.4 - Модель змін стану віртуальної машини

1.4 Консенсус

Так як велика кількість вузлів працює для підтримання системи, необхідний механізм, за яким буде відбуватися вибір правильного блока. Саме за це відповідає консенсус.

Існує велика кількість різноманітних видів консенсусів, однак найбільш розповсюдженими є Proof of Work, Proof of Stake, Proof of Authority.[14]

Також існує трилема децентралізації, що має наступний вигляд: блокчейн система може задовольнити лише дві із трьох характеристик

1. Децентралізація - визначає наскільки простими є вимоги до фізичних характеристик вузла
2. Масштабованість - як система буде себе вести при збільшенні кількості транзакцій.
3. Безпека - показує наскільки стійкою є система до зловмисних вузлів, тобто таких, що можуть генерувати некоректні дані.

1.4.1 Proof of Work (PoW)

PoW був спочатку винайдений як засіб для боротьби зі спамом (hashcash[10]). Якщо надсилання електронної пошти буде обчислювально дорогим, тоді масова розсилка електронної пошти буде надто дорогою, але для звичайного користувача вона буде майже безкоштовною.
Біткойн, який зробив технологію блокчейн популярною, розробив так званий алгоритм Proof of Work (PoW)[11]. В принципі, кожен учасник мережі Bitcoin може брати участь у генерації блоку. Щоб підтвердити транзакцію та ввести блок у блокчейн, майнер має надати відповідь або доказ на певний виклик. Майнери використовують PoW для перевірки транзакцій і майнінгу нових монет, але його основна мета — блокувати потенційні кібератаки або підозрілу діяльність у мережі.

У криптовалютних мережах «майнери» — це спеціальні вузли, які виконують обчислення PoW для набору транзакцій плюс хеш попереднього блоку для створення наступного блоку в блокчейні[13]. Оскільки блок містить хеш попереднього блоку, зміна історичного блоку потребуватиме регенерації всіх наступних блоків. Регенерація всіх хешів потребує інтенсивних обчислень і багато енергії, а енергія не безкоштовна. Це також займе багато часу. Процес перевірки роботи та генерування

блоків називається «майнінг». За цю роботу майнери винагороджуються ново згенерованими монетами, які додають до загальної емісії.

Алгоритм

1. Транзакції об’єднуються у вигляді блоків.
2. Майнери перевіряють транзакції всередині блоків як законні.
3. Потім майнери вирішують математичну задачу, відому як проблема підтвердження роботи.
4. Потім винагороду отримує той, хто першим розв’яже проблему.
5. Перевірені транзакції зберігаються в публічному блокчейні.

Переваги

1. Найбезпечніший
2. Плата за транзакцію необов'язкова
3. Легко перевіряються рішення
4. Важко знайти рішення
5. Складність пошуку рішень можна точно визначити кількісно

Недоліки

1. Низька пропускна здатність
2. PoW використовує величезну кількість обчислювальної потужності, що саме по собі знижує мотивацію
3. Він також вразливий для атак, оскільки потенційному зловмиснику потрібно мати 51% ресурсів майнінгу (хешрейт), щоб контролювати мережу, хоча це не просто зробити.
4. Зменшення винагороди за блок

1.4.2 Proof of Stake (PoS)

Механізм доказу частки володіння (PoS) працює за допомогою алгоритму[8][9], який вибирає учасників з найвищими ставками як валідаторів, припускаючи, що найвищі зацікавлені сторони мають стимул для забезпечення обробки транзакції. Ідея полягає в тому, що ті, у кого в обігу найбільше монет, можуть найбільше втратити, тому вони готові працювати в інтересах мережі. Кількість монет, яку може потребувати мережа, змінюється так само, як і складність у PoW.

У PoS блоки створюють не майнери, які виконують роботу, а “мінтери”, які роблять ставку на свої токени, щоб зробити ставку на те, які блоки дійсні. У випадку форка мінтери витрачають свої токени, голосуючи за те, який форк підтримати. Якщо припустити, що більшість людей голосує за правильний форк, валідатори, які проголосували за неправильний форк, «втратять свою частку» в правильному.
Поширеним аргументом проти доказу участі є проблема «нічого не поставлено на карту».[12] Занепокоєння полягає в тому, що, на відміну від PoW, валідатори майже не потребують обчислювальної потужності для підтримки форка, тому валідатори можуть голосувати за обидві сторони кожного форка, що відбувається. Тоді форки в PoS можуть бути набагато більш поширеними, ніж у PoW, що, на думку деяких людей, може зашкодити довірі до валюти.

Модель безпеки є економічною, заснованою на припущенні «теорії ігор» про те, що вартість придбання токенів, необхідних для того, щоб стати виробником блоків, є більшою, ніж готовий нести зловмисник, що поєднує безпеку мережі з цінністю її токена, тобто чим вища ціна токена, тим безпечнішою стає мережа[13]

Алгоритм

1. Валідатори повинні заблокувати деякі свої монети як ставки.
2. Після цього вони почнуть перевірку блоків. Це означає, що коли вони виявляють блок, який, на їхню думку, можна додати до ланцюжка, вони підтверджують його, роблячи на нього ставку.
3. Якщо блок буде додано, валідатори отримують винагороду, пропорційну їхнім ставкам.

1.5 Смарт-контракти

Говорячи про блокчейн протоколи одним з найважливішим поняттям є смарт-контракт.
Смарт-контрактов називають програму, що зберігається у блокчейні та виконується у контексті віртуальної машини. Для написання смарт-контрактів використовується велика кількість мов програмування, від загальноприйнятих та розповсюджених, таких як Python, JavaScript, Rust так і пропрієтарних як Solidity, Plutus, Sophia, !ink тощо.

На основі смарт-контрактів будуються блокчейн застосунки, або dApp - decentralized application. Однією з основних характеристик смарт-контрактів є те, що їх вихідний код доступний у блокчейні і будь-хто може перевірити його на вразливості чи можливі лазівки.

Для демонстрації можливостей смарт-контрактів продемонстровано приклад escrow-програми.

Її суть полягає у тому, щоб дві сторони, наприклад А та Б могли обмінятись деяким активами Аа та Аб відповідно. Однак зробити це peer-to-peer не можна адже тоді можливий випадок шахрайства, коли одна сторона надсилає актив, а інша - ні.
У фізичному світі зазвичай на допомогу приходить третя сторона, що виконує роль арбітра. Однак блокчейн має змогу це виконати за допомогою смарт-контрактів.

Алгоритм роботи:

1. У блокчейні розгортається спеціальний контракт, що буде виконувати роль третьої сторони
2. Сторона А надсилає на створений контракт свій актив Аа, вона має змогу його забрати у випадку, коли обмін не буде проведено
3. Сторона Б надсилає свій актив Ба та у цей момент відбувається обмін даними активами

Таким чином можна безпечно провести обмін не довіряючи третій особі. А можливість перевірити логіку смарт-контракту через його публічність надає гарантії безпеки.

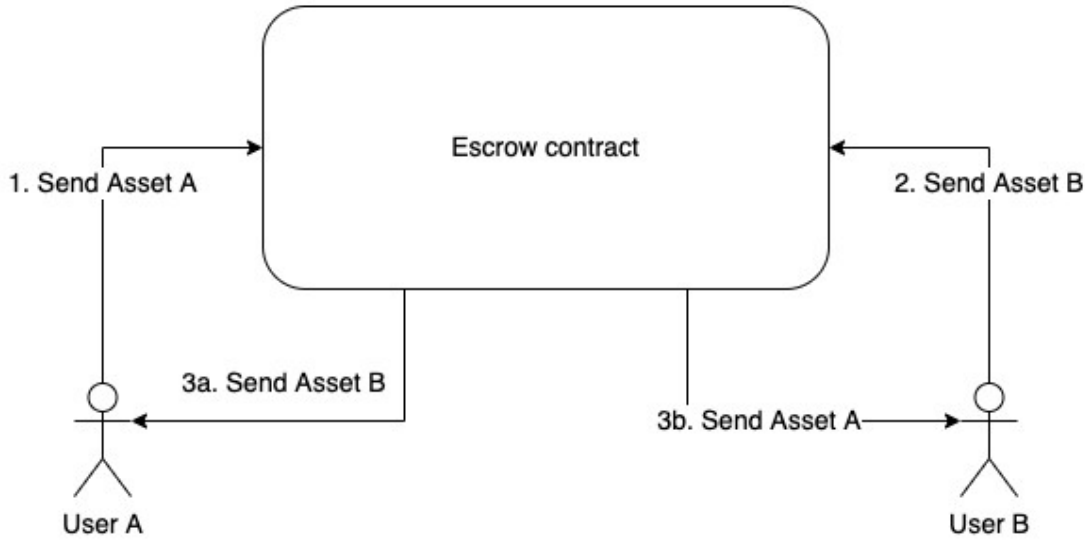


Рисунок 1.5 - Схема роботи контракту ескроу

1.5.1 Компонування смарт-контрактів

Окрім потужності концепції смарт-контрактів самих по собі, у випадку наявності екосистеми застосунків з’являється велика кількість додаткових можливостей використання функціоналу розумних контрактів. Мова йде саме про можливість комбінувати різні децентралізовані застосунки.

Це забезпечено тим, що смарт-контракти мають публічні інтерфейси, тобто кожен має змогу отримати інформацію про те, як функціонує застосунок із середини та яким чином можно викликати його методи. Так як усі розумні контракти знаходяться у спільному контексті вони можуть використовувати функціонал один одного.
Для наглядності наведемо приклад дещо модифікованого застосунку для обміну активами, що був запропонований у попередньому розділі.
У новому застосунку додамо перевірку чи є активи, що намагається обміняти кожна із сторін, легітимними, тобто чи належать вони до якогось схваленого списку.

Тоді алгоритм буде виглядати наступним чином:

1. У блокчейні розгортається контракт для обміну активів
2. Додатково розгортається контракт Whitelist, що буде зберігати інформацію про схвалені активи
3. Користувач А намагається надіслати на контракт escrow свій актив Аа, у випадку, коли цей актив є схваленим, то все проходить без збоїв
4. Користувач Б також намагається додати свій актив Аб контракту escrow, якщо даний актив є схваленим, то відбувається обмін цими активами

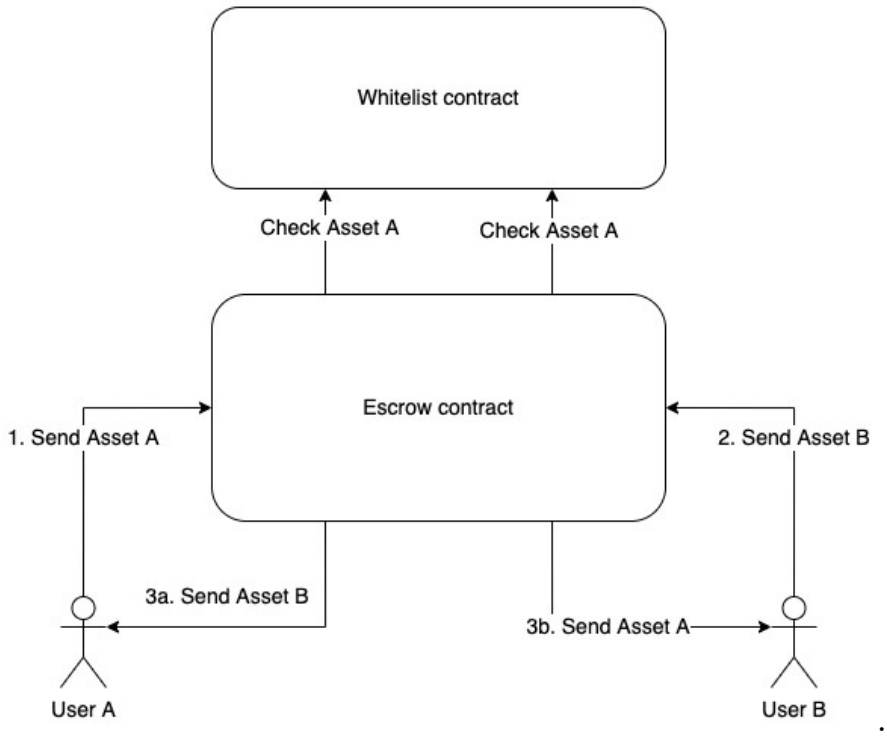


Рисунок 1.6 - Приклад компонування смарт-контрактів

1.6 Стандарти контрактів

Як було показано раніше смарт-контракти надають великі можливості при створенні децентралізованих застосунків. Однак як показав час якщо кожен розробник буде хаотично створювати свої види смарт-контрактів, то при їх компонування та взаємодії у подальшому можуть бути складнощі. Через це на базі кожного блокчейн протоколу створюється своя база стандартів. Далі буде розглянуто стандарти блокчейн-протоколу Ethereum.

Стандарти іменуються EIP - Ethereum Improvement Proposals. Вони можуть відноситись як до рівня смарт-контрактів, тобто application level, так і рівня безпосередньо блокчейн протоколу чи віртуальної машини, наприклад, інтегрування передкомпільованих функцій у віртуальну машину, що дозволяє значно підвищити ефективність виконання програм.

У контексті даної роботи буде розглянуто два з базових стандартів : ERC20 та ERC721.
Вони описують замінювані[1](fungible) та незамінювані[2][3] (Non-fungible) токени відповідно.

1.6.1 ERC20

Наступний стандарт дозволяє реалізувати стандартний API (Application Program Interface) для токенів у смарт-контрактах. Цей стандарт надає базові функції для передачі токенів, а також дозволяє підтвердити витрату токенів, щоб їх могла витратити інша третя сторона в мережі.

Стандартний інтерфейс дозволяє повторно використовувати будь-які токени Ethereum іншими програмами: від гаманців до децентралізованих бірж.

За специфікацією контракт за даним стандартом має наступний інтерфейс:

- name
- symbol
- decimals
- totalSupply
- balanceOf
- transfer
- transferFrom
- approve
- allowance

Найбільш доцільною аналогією з реального фізичного світу можуть бути звичайні фіатні гроші, накшталт гривні. Так гривню, що має користувач А та гривню, що має користувач Б можна обміняти одна на одну, адже вони є рівноцінними.

1.6.2 ERC721

Non-Fungible Token, незамінюваний

Наступний стандарт дозволяє реалізувати стандартний API (Application Program Interface) для NFT у смарт-контрактах. Цей стандарт надає базові функції для відстеження та передачі NFT.

NFT можуть представляти право власності на цифрові або фізичні активи. Прикладами використання можуть бути наступні варіанти:

- Фізична власність — будинки, унікальні твори мистецтва
- Віртуальні предмети колекціонування — унікальні картинки кошенят, колекційні картки
- Активи з «негативною вартістю» — кредити, тягарі та інша відповідальність

Стандартний інтерфейс дозволяє застосункам гаманця/брокера/аукціону працювати з будь-якими NFT на Ethereum.
Цей стандарт натхненний стандартом токенів ERC-20.

1.7 Недоліки поточних версій стандартів

1.7.1 Шахрайство з ERC20 токенами

Більшість усіх атак типу шахрайства із коштами користувачів здійснюється через так зване «схвалення».
Approve - це операція (функція смарт-контракту в стандарті токенів ERC20), яка надає дозвіл на використання токенів користувачів.
Схвалення може бути надано лише власником токена (користувачем).
Схвалення може бути обмеженим і необмеженим. Обмежене схвалення — це дозвіл витрачати точну кількість токенів, тоді як необмежене схвалення — це дозвіл витрачати будь-яку можливу кількість токенів, які користувач може мати (але менше ніж 256-бітове ціле число 0xFF..FF, що є технічним обмеженням реалізації)

Ідея (і ціль) усіх шахрайських атак — змусити користувача надати зловмиснику дозвіл на використання його коштів.
Після схвалення зловмисник негайно перераховує кошти користувачів на свій гаманець.

1.7.2 Шахрайство у вигляді витрати зовнішньою адресою або контрактом

Цей «підхід» базується на функції схвалення (Approve) стандарту ERC20. По суті, ця функція дозволяє власнику токена видати «схвалення» для того, щоб витратити певну кількість токена іншому користувачеві або смарт-контракту. Створення даної функції було викликано необхідністю вирішення стратегічної проблеми. Коли токени надсилаються на адресу певного смарт-контракту, сам цей смарт-контракт (приймач токенів) ніколи не викликається. Таким чином, неможливо створити складну логіку

в смарт-контракті (який отримує токени), який повинен ініціюватися початковою передачею токена. Щоб подолати цю проблему, була розроблена функція “схвалення”. Запускаючи виклик «схвалення» для смарт-контракту токенів, користувач «дозволяє» іншій адресі (наприклад, смарт-контракту фермерства) «брати та витрачати» його токени. Отже, замість підходу «переказ і виклик», який зазвичай працює з нативним ETH в рамках однієї транзакції, ми отримуємо підхід «підтвердження і виклик», який вимагає 2 окремих транзакцій за 2 окремими контрактами.

Існує кілька різних типів шахрайства, «на основі схвалення», але всі вони мають спільну рису — користувач повинен оформити початкову транзакцію схвалення, не розуміючи, який саме токен, яку точну адресу отримувача та яку саме суму витрат видає це схвалення.

1.7.3 Мімікрія під певний проект

Це один із найпоширеніших підходів для шахрайства.

Цей підхід ґрунтується на тому факті, що користувачі зазвичай виконують транзакції «схвалення», а потім «депозиту», коли розміщують токени для фермерських застосунків. Отже, ідея полягає в тому, щоб скопіювати сайт, перевести користувача на шахрайський веб-сайт і змусити користувача видати «схвалення» транзакції на адресу гаманця зловмисника або смарт-контракту. Потім негайно забрати токени з гаманця користувача та перевести їх на DEX (перетворити токен проекту на деякі ліквідні токени, як-от USDC або нативний ETH).

Як запобігти такому виду атаки:

- Користувач: переконайтеся, що ви підтвердили принаймні доменну адресу веб-сайту. Цей домен має бути чітко пов’язаний із проектом.
- Користувач: переконайтеся, що ви розумієте та перевірте адресу, для якої ви видаєте дозвіл. Проста перевірка за допомогою вкладки власників токенів BSCScan може дати підказку про те, чи ця адреса насправді є смарт-контрактом і містить значну кількість транзакцій/токенів.
- Користувач: переконайтеся, що ви не видаєте «нескінченне» схвалення в адресі контракту, якщо ви чітко не розумієте та очікуєте, що ви будете використовувати той самий контракт для майбутніх транзакцій (наприклад, маршрутизатор Pancake/Uniswap)
- Емітент токенів/проект: переконайтеся, що ви стежите за профілями/каналами/групами в соціальних мережах. Це найпопулярніші місця для розповсюдження зловмисниками наживки.
- Емітент токенів/проект: в ідеалі запровадити підхід, що ґрунтується на ризиках, для схвалення транзакцій

1.7.4 Rug pull, спустошення контракту

Rug pull — це зловмисні дії у криптовалютній індустрії, коли розробники криптовалюти лишають проект і тікають із коштами інвесторів. Зловживання зазвичай відбувається в екосистемі децентралізованих фінансів (DeFi), особливо на децентралізованих біржах (DEX[16][17]), де зловмисники створюють токен і розміщують його на DEX, а потім поєднують його з провідною криптовалютою, такою як Ethereum. DEX приваблює користувачів, оскільки ці типи бірж дозволяють користувачам

розміщувати токени безкоштовно та без перевірки, на відміну від централізованих бірж криптовалют. Крім того, створювати токени на протоколах блокчейну з відкритим кодом, таких як Ethereum, легко та безкоштовно. Зловмисники використовують ці два фактори у своїх інтересах.

Атака:
Після того, як значна кількість інвесторів обмінює свій ETH на вказаний токен, розробники вилучають усе з пулу ліквідності, зводячи ціну монети до нуля. Засновники монети можуть навіть створити тимчасовий ажіотаж навколо Telegram, Twitter та інших платформ соціальних мереж і спочатку влити значну кількість ліквідності в свій пул, щоб підвищити довіру інвесторів.

Засоби уникнення:

1. перевірити ліквідність у пулі.
    a. Скільки він має постачальників ліквідності? Хороший проект має значну кількість постачальників ліквідності (>20)
    b. Чи ліквідність «заблокована» в пулі і як довго? Більшість авторитетних проектів блокують об’єднану ліквідність на певний період. (>3 місяців)
    c. Іншою важливою характеристикою можливого шахрайства є стрімке зростання ціни монети протягом кількох годин. Наприклад, монета може змінюватися від 0 до 50X протягом 24 годин. Цей трюк призначений для стимулювання FOMO, що спонукає більше людей інвестувати в токен.
2. Виберіть усталені продукти. Шахрайства такого роду найчастіше трапляється з новими проектами, які не так сильно контролюються, як більш усталені криптовалюти.

3. Знати код. Один зі способів оцінити потенційну інвестицію, не заглиблюючись під капот самостійно, — перевірити її професійною організацією, яка користується повагою в галузі. Проекти, які отримали хороші оцінки від аудиторів, часто самі рекламують результати.

Як можна бачити з вище описаних атак - існує проблема втрати контролю над своїми коштами. Адже відсутній процес обернення транзакцій з переведенням активів.

Висновок до першого розділу

У даному розділі була подана основна інформація, що пов’язана із принципом функціонування блокчейн-протоколів. Було описано механізм їх роботи та принципи досягнення консенсусу серед учасників мережі. Описано важливу концепцію смарт-контрактів, їх можливості та засоби побудови децентралізованих застосунків. На прикладі простого застосунку резервного рахунку було описано концепцію компонування контрактів. Існування стандартів смарт-контрактів вирішило проблему інтеграцію їх один з одним, однак це не позбавило основні контракти ERC20 та ERC721 проблем із шахрайством.

2 МОДИФІКАЦІЯ ІСНУЮЧИХ РЕАЛІЗАЦІЙ ТОКЕНІВ ЗАДЛЯ ПІДВИЩЕННЯ БЕЗПЕКИ ВІД ШАХРАЙСТВА

2.1 Реверсивні токени

Як зазначалось у попередньому розділі, якби був механізм обернення виконання транзакції, то можна було б зменшити втрати від крадіжок. Однак окрім крадіжки, фінальність транзакції також працює проти користувача, коли кошти випадково надсилалися на неправильну адресу. У травні блокчейн на базі Cosmos надіслав 36 мільйонів доларів США на невірну адресу. Адреса містила друкарську помилку, через що кошти були втрачені. Кошти можна було б повернути, якби був спосіб скасування цієї операції.[23]

Мова про реверсивні транзакції ходила давно, з 2018 року. Коли було запропоновано обернути нативний токен ефір у контракт, що мав би реверсивні функції.

Увімкнути оборотні транзакції нелегко, і це пов’язано з багатьма цікавими технічними проблемами. Однією з основних задач роботи є створення такого токену, що буде обернено сумісним з існуючими токенами ERC20 та ERC721.
На схемі можна побачити приблизний процес реверсу токенів.

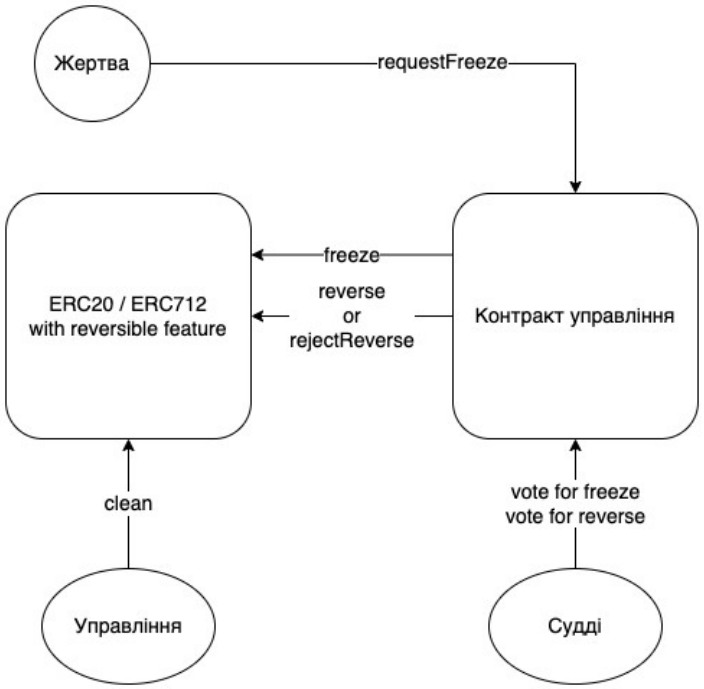


Рисунок 2.1 - Процес обернення токенів

Схема має наступні кроки :

1. Запит на заморожування. Жертва публікує запит на заморожування контракту управління разом із відповідними доказами та деякою сумою залогу. Запит на заморожування транзакції може бути ініційований лише адресою, на яку транзакція безпосередньо впливає.
2. Заморозка активів. Децентралізована група суддів вирішує прийняти або відхилити запит. Якщо його прийнято, судді дають вказівку контракту управління у блокчейні викликати функцію заморозки на атакованому контракті ERC-20 або ERC-721. Згодом активи, про які йде мова, заморожуються і більше не можуть бути передані. Для NFT це питання просте, однак для токенів ERC-20 це складніше.

3. Суд. Потім обидві сторони можуть представити докази децентралізованому набору суддів. Зрештою судді приймають рішення, після чого вони вказують контракту управління викликати функції реверсу або відхилити реверс для контракту ERC-20 або ERC-721, на який впливає контракт. Зворотна функція передає спірні (заморожені) активи їх первісному власнику. rejectReverse знімає блокування спірних активів і залишає їх там, де вони є. Судовий розгляд може бути тривалим і, можливо, займати кілька тижнів або місяців.

2.2.1 Пошук викрадених активів

До того моменту, коли жертва подає запит на блокування, зловмисник може вже перемістити викрадені активи через кілька облікових записів. Насправді зловмисник може контролювати пул транзакцій і перемістити активи, щойно побачивши запит на заморожування вкрадених активів. У випадку NFT зловмисник міг продати вкрадений NFT чесному користувачеві, який нічого не підозрює. У випадку з токеном ERC-20 зловмисник міг розділити вкрадені токени між кількома обліковими записами; можливо, він обміняв частину токенів на інший токен ERC-20 за допомогою чесного обміну в мережі; можливо, частину токенів було відправлено на “нульову адресу” ; або шахрай міг надіслати частину вкрадених активів до міксеру активів[23]. Нові оборотні стандарти повинні належним чином розглядати всі ці випадки.

У випадку справи щодо NFT ERC-721 заморожування застосовується до поточного власника NFT: або початкового зловмисника, або чесного користувача, який придбав у зловмисника вкрадений NFT. Якщо судді вирішать, що мала місце крадіжка, тоді контракт ERC-721 надсилає NFT назад власнику. Поточний власник NFT втрачає NFT. Ця політика

узгоджується з законодавством у багатьох країнах, але, звичайно, можна застосовувати й інші політики.

У разі справи про вкрадені токени ERC-20 все складніше. До моменту заморожування кошти могли бути розподілені між багатьма наступними рахунками, деякі чесні, а деякі нечесні. Далі буде надано приклад алгоритму, який призначає часткову відповідальність кожному з наступних рахунків, які отримали частину вкрадених коштів. Потім до цих рахунків застосовується часткове заморожування. Реалізація цієї стратегії заморожування вимагає від контракту ERC-20 ведення журналу транзакцій протягом періоду оскарження, щоб функція заморожування могла відстежувати кошти, коли її викликає контракт управління. Якщо судді вирішать, що відбулася крадіжка, контракт ERC-20 надсилає заморожені токени з підозрілих облікових записів на облікового запису початкового власника.

2.2.2 Деталі із реалізації

Реалізація мовою Solidity розділена на дві частини: (i) основні контракти ERC-20 і ERC-721, які відстежують усі баланси та транзакції, і (ii) контракт управління, який обирає суддів і збирає голоси. Поточні версії контрактів розширюють звичайні версії контрактів ERC20 та ERC721.
У звичайному контракті ERC20, контракт керує великою кількістю записів балансів, однак на кожний обліковий запис існує один запис балансу.
У реверсивному варіанті ERC20 токену існує два види балансу R-баланс та NR-баланс.
Перший відповідає за врахування токенів, щодо яких може бути проведений процес реверсу, в той час як до другого виду такий процес вже неможливий.

1. NRbalance – це поточний баланс рахунку через вхідні транзакції, період оскарження яких минув. Кошти в NRbalance не підлягають поверненню: вони більше не підлягають потенційному замороженню та поверненню.
2. RBalance – це залишок на рахунку із останніх вхідніх транзакцій. Кошти на балансі можуть бути задіяні у процесі повернення.

По завершенню періоду оскарження кошти переміщуються з Rbalance до NRbalance. Це робиться за допомогою функції очищення (Clean).
Коли власник рахунку надсилає кошти зі свого власного рахунку на інший рахунок (скажімо, для здійснення обміну активами), власник рахунку вказує, яку суму потрібно вилучити з Rbalance, а скільки – з NRbalance. Тобто стандартна функція передачі ERC-20 тепер доступна в двох варіантах: transfer() і Rtransfer().
Перший варіант перераховує з необоротного балансу (NRBalance), а другий – з оборотного (RBalance). У будь-якому випадку переказані кошти додаються до R-балансу одержувача. Таким чином дана реалізація є сумісна зі стандартними реалізаціями класичного контракту ERC20. Коли контракту потрібно спалити токени в обліковому записі (наприклад, у зв’язку з вилученням фіатних чи базових токенів), зазвичай буде спалено лише токени з NR-балансу облікового запису. Однак можуть бути ситуації, коли контракт готовий спалити з Rbalance.

Крім нового інтерфейсу передачі, контракт ERC-20 і ERC-721 відкриває нові функції інтерфейсу заморожування, реверсування, відхилення реверсу та очищення.

Більш детальний опис кожного з методів:

1. reverse(): надсилає всі заморожені активи, пов’язані із заявленою крадіжкою, назад початковому власнику. Для ERC-20 приймає як аргумент дійсний ідентифікатор спору. Для ERC-721 приймає аргументи (tokenId, index), де індекс ідентифікує транзакцію, що скасовується.
2. rejectReverse(): розморожує всі суми, пов’язані зі спором. Для ERC-20 приймає як аргумент ідентифікатор спору; для ERC-721 приймає як аргумент tokenId.
3. clean(): оборотні контракти зберігають деякі дані транзакцій у ланцюжку. Функція очищення видаляє інформацію в ланцюжку для транзакцій, період оскарження яких минув.

2.2 Процес повернення

Надалі буде надано опис процесу повернення вкрадених коштів.

2.2.1 Заморозка активів

Після факту крадіжки з контракту ERC-20 або ERC-721 жертва публікує запит на заморожування в контракті управління. Запит містить ідентифікатор транзакції, яка є порушенням, посилання на доказ того, що мала місце несанкціонована передача, і певну суму залогу. Якщо докази переконали суддів, вони доручають контракту про управління викликати функцію заморозки у відповідному контракті ERC-20 або ERC-721.[23] Після заморожування активів будь-яка спроба їх передати буде неможливою. Якщо суддів не переконали, вони вказують контракту

управління відхилити запит, а жертва втрачає суму залогу, що надавала при створенні запиту.

2.2.2 Заморозка активів ERC721

Функція заморзи активів для токену ERC721 є доволі тривіальною.
На початку треба додати нові поля для смарт-контракту, що будуть містити допоміжну інформацію. Ці поля будуть типом мапа або, як вона називається у деяких мовах, словник. Тобто буде зберігати пари (ключ : значення).
Перша мапа буде мати наступний вигляд : (uint256 => bool)
Де ключем буде беззнаковий цілий тип розмірності 256 біт, а значенням за цим ключем логічного типу, тобто True або False.
Друга мапа матиме наступний вигляд : (uint256 => Queue)
Ключем є той самий беззнаковий цілий тип, а значення за ключем - спеціальна структура Черга, її опис буде надано далі.
Перша мапа використовується для індикації чи є токен із певним ідентифікатором типу uint256 замороженим. Таким чином True означає, що він заморожений, а False - ні. У випадку, якщо актив є замороженим, то його передача (transfer) відбуватися не може.
Друга мапа міститиме інформацію про власників певного токену з певним ідентифікатором та як ці власники змінювались.
Наприклад,

tokenId1 => [(owner1, timestamp1), (owner2, timestamp2),...];

Структура Queue представляє собою чергу, у якій містяться у хронологічному порядку записи виду : (новий_власник, час_придбання).

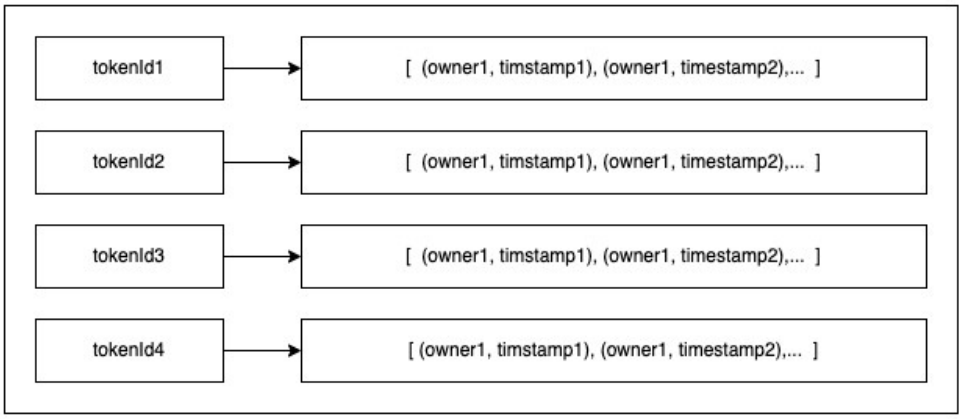


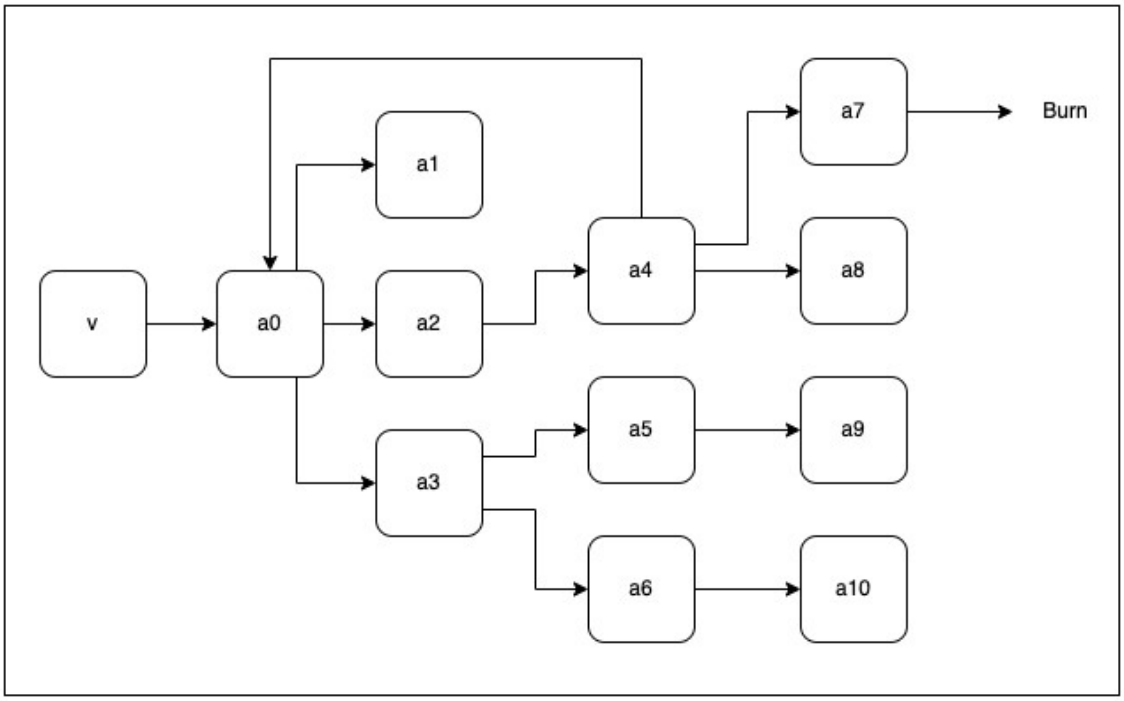
Рисунок 2.2 - Структура сховища для зберігання додаткової інформації про переміщення токенів

Контракт управління (governance) викликає метод freeze, передаючи туди як параметри ідентифікатор токена та індекс. У даному випадку індекс вказує на запис структурі Queue, що містить поточного власника токена та час його придбання.

2.2.3 Заморозка активів ERC20

Функція заморожування в контракті ERC-20 набагато складніша. Проблема полягає в тому, що токени могли бути переведені на кілька облікових записів між моментом крадіжки та запитом на заморожування.

На рисунку можна бачити один із можливих сценаріїв переміщення активів з адреси жертви v на адресу зловмисника a0. Подальші перекази від a0, які відбулися після спірної транзакції, але до запиту на заморожування, позначаються як спрямовані ребра на графу. Усі ці подальші адреси можуть містити вкрадені кошти, які, можливо, доведеться заморозити.


v
a0
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10
Burn

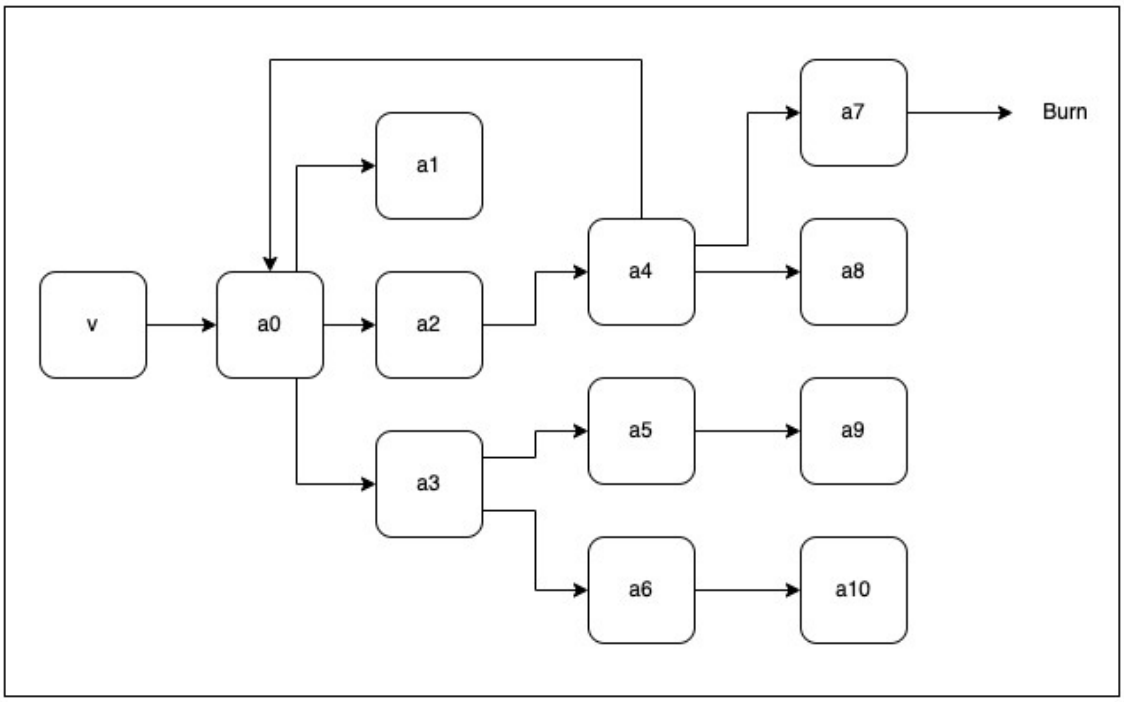


Рисунок 2.3 - Граф переказів коштів

Інша складність полягає в тому, що токени можуть бути спалені і просто зникнути з системи. Наприклад, у контракті стейблкойну, який забезпечений фіксованою валютою, обліковий запис a8 може вибрати викупити свої монети на фіксовану валюту, після чого монети в a8 спалюються. Якщо контракт підтримує записування токена з Rbalance a8, тоді може бути недостатньо активів для заморожування за адресою a8, і викрадені кошти буде остаточно втрачено. Ось чому спалювання коштів зазвичай застосовуються до NR балансу облікового запису.

Ще одна складність полягає в тому, що одна адреса, скажімо, адреса a1, може володіти коштами, отриманими від кількох крадіжок. Коли кошти заморожені, системі потрібно запам’ятати суму, заморожену для кожного запиту на повернення, і якщо запит на повернення буде схвалено, лише кошти, пов’язані з цим запитом, повинні бути зняті з рахунку a1.

2.2.4 Обернення транзакції

Після завершення судового розгляду спірної транзакції контракт управління викличе функцію rejectReverse або функцію Reverse у контракті ERC-20 або ERC-721.

1. Скасування ERC-20 приймає як вхід ClaimID, який є індексом у масиві претензій. Запис вказує на зобов'язання кожного підозрюваного облікового запису в спірному переказі. Функція переказує вказану суму з R-балансу підозрілих рахунків початковому власнику та знімає замороження
2. Скасування ERC-721 приймає як вхідні дані TokenID та індекс у масиві власників. Це вказує на власника до оскаржуваної операції. Потім актив передається початковому власнику, і актив розморожується.

Висновок до другого розділу

Даний розділ містить схематичний опис процедури за рахунок якої можливий процес повернення втрачених коштів. Тобто це дає змогу у разі крадіжки чи взлому звернутись потерпілій стороні та мати можливість оскаржити певну транзакцію. Надано опис додаткових процедур, що зможуть використовуватись для досягнення оборотності токенів.

3 РЕАЛІЗАЦІЯ РЕВЕРСИВНИХ ТОКЕНІВ

3.1 Розрахунок заморожуваних адрес для ERC20

Як зазначалось у попередньому розділі функція заморожування для взаємозамінних токенів є набагато складнішою. Проблема полягає в тому, що токени могли бути переведені на кілька облікових записів між моментом крадіжки та запитом на заморожування.

Інша складність полягає в тому, що токени можуть бути спалені і просто зникнути з системи.
Якщо контракт підтримує спалювання токена з Rbalance, тоді може бути недостатньо активів для заморожування за підозрілою адресою, і викрадені кошти буде втрачено без можливості їх відновлення та повернення власнику.
Іншою складністю є те, що одна адреса, може володіти коштами, отриманими від кількох крадіжок.
Якщо на одному рахунку є кілька подій заморожування, заморожена сума накопичується. Зокрема, якщо на рахунку a є загальна кількість заморожених монет s, тоді контракт ERC-20 відхилить будь-який переказ, який призведе до того, що R-баланс рахунку a опуститься нижче s.

3.1.1 Розрахунок підозрілих адрес

Далі буде надано опис стратегії розрахунку суми, що буде заморожено на підозрілих адресах.

Облікові записи, які підлягають заморожуванню, називаються підозрілими адресами. Алгоритм блокування переслідуватиме вкрадені кошти на графу транзакцій і заморожуватиме кошти на підозрілих адресах. Алгоритм

заморозить активи, які максимально наближені до жертви у графі крадіжки.

Перш ніж описати детальний алгоритм, необхідно спочатку розглянути кілька прикладів. Припустимо, що s монет викрадено у жертви v і переведено на рахунок a0 у момент часу T0. У момент часу Tf > T0 контракт ERC-20 отримує запит на заморожування цієї транзакції.
Приклад 1:
Якщо в момент часу Tf R-баланс на a0 становить s або більше, то s монет буде заморожено на a0, і процес завершується.

Приклад 2:
Припустимо, що в деякий час між T0 і Tf транзакція передає s/4 монети від a0 до a1 і додаткові s/4 монети від a0 до a2. Залишок R-баланс на a0 становить s/2, і жодні наступні транзакції не застосовуються до a0. Потім у момент заморожування весь баланс s/2 R на a0 буде заморожено. Крім того, якщо Rbalance на a1 і a2 дорівнює рівно (s/4), то (s/4) монети будуть заморожені на кожному з цих рахунків. Тепер, коли було заморожено загальну кількість s монет, процес припиняється. Важливо відмітити, що a1 або a2 можуть бути адресами чесної біржі або мікшер-сервісу, і в цьому випадку частина пулу ліквідності монети на біржі або змішуванні буде заморожена.

Приклад 3.
Припустимо, що після спірної транзакції від v до a0 існує друга транзакція a0 → a1. Зрозуміло, що якщо на момент заморожування на a0 недостатньо коштів, то деякі зобов’язання щодо заморожування повинні бути передані a1, як пояснювалося в попередньому параграфі. Однак припустимо, що за деякий час до транзакції a0 → a1 існує транзакція a1 → a2, яка переказує

кошти від a1 до a2. Пропозиція полягає у тому, щоб жодне із зобов'язань щодо заморожування не переходило до a2, оскільки транзакцію a1 → a2 було опубліковано до того, як спірні кошти надійшли до a1. Кошти, надіслані на a2, безпосередньо не беруть участь у суперечці. Можна навести приклади, коли ця політика може збити із шляху, але загалом стверджується, що a2 не має брати участь у замороженні.

Ці приклади свідчать про те, що процес заморожування є ітеративною процедурою, яка заморожує максимально можливу кількість на кожному кроці. Якщо R-баланс на поточному вузлі недостатній, тоді зобов'язання передається нащадкам цього вузла. Процес припиняється, коли монети s заморожено або коли більше не залишається нащадків для обробки.

Важливо наголосити, що весь алгоритм заморожування виконується в одній транзакції. Це гарантує, що залишки на рахунку не можуть змінитися під час виконання процесу заморожування.

3.1.2 Визначення

Для подальшого розгляду алгоритму необхідно ввести деякі позначення та визначення, що будуть надані далі.

Припустимо, що функція заморожування викликається для заморожування транзакції t0, яка передає s монет з адреси v на адресу a0. Нехай tf буде опублікованою транзакцією заморожування в ланцюжку. Для опису алгоритму заморожування використовуватимуться наступні позначення:

toFreeze(a) - це кількість монет, яку транзакція заморожування tf заморозить за адресою a. На початку алгоритму toFreeze(a) = 0 для всіх a, за винятком a0, де

$$toFreeze(a0) := min(s, (Bal(a0))) \quad (1)$$

Величина Bal(a) для адреси a є доступним Rbalance в a на момент tf. Цей Bal(a) розраховується як Rbalance на a на початку транзакції заморожування мінус кількість монет, уже заморожених на a через попередній спір. Таким чином (1) заморозить максимальну можливу кількість на a0.

$$t = (a \text{ -> } b), \quad (2)$$

Транзакція t, за якої передається з адреси a на адресу b *val(t)* монет

burnedAt(a) кількість монет, що були спалені за адресою a із Rbalance між першою транзакцією, що надсилала частину підозрілих активів до адреси a та транзакцією заморозки tf.

3.1.3 Алгоритм

Спочатку буде описано алгоритм заморожування, який застосовується, коли граф G (Рисунок 3.1) з коренем a0 є орієнтованим ациклічним графом (DAG). Далі буде надано варіант, коли граф буде ациклічним: тобто випадок, за якого частина коштів повертається до попереднього облікового запису.

Алгоритм заморожування реалізовано у функції CalcFreeze. Ця функція називається

CalcFreeze(t0), (3)

де t0 є спірною транзакцією.

Алгоритм починається з побудови топологічно відсортованої списку вершин графу транзакцій. Топологічно відсортований — це список L вершин у G, де кожна вершина в G з'являється в L точно один раз, так що для кожного ребра (a → b) вершина a з'являється в L перед b. Кожен DAG має топологічно відсортований список вершин L, і L може бути побудований за лінійний час за кількістю ребер.

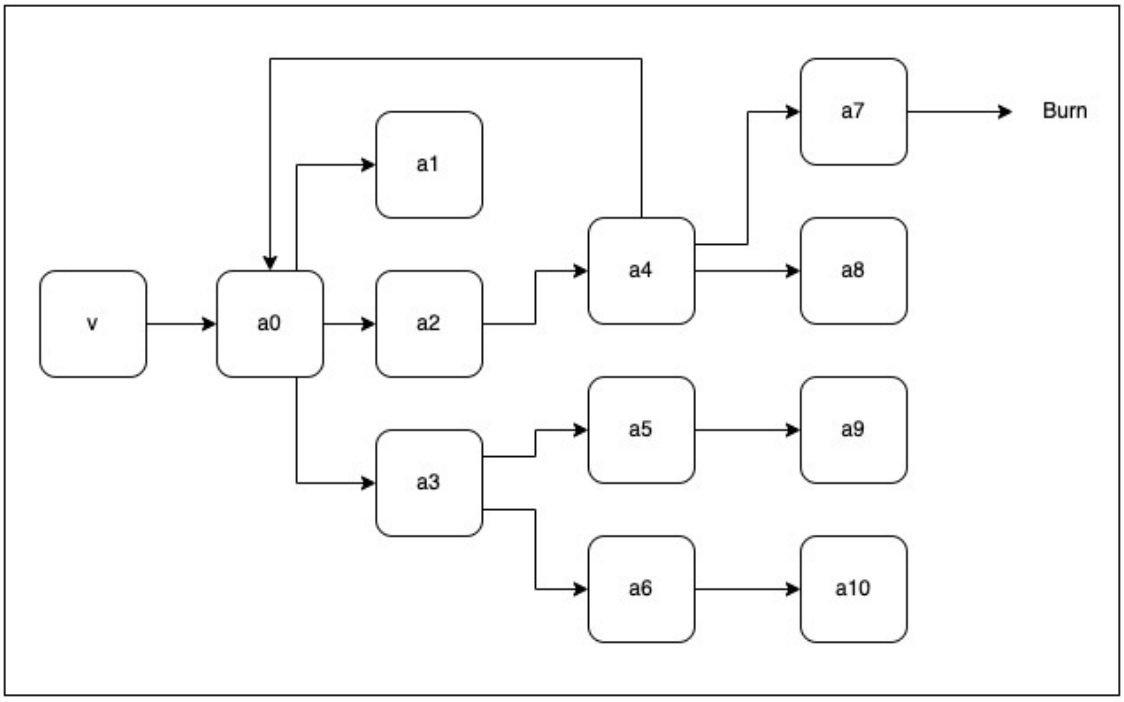


Рисунок 3.1 - Граф G переказів коштів

Для кожної адреси *a* в *G* алгоритм створює значення *oblig(a)*, яке вказує суму зобов'язання, яка передається на адресу *a* та її нащадків у результаті спірної транзакції. Значення *oblig(a)* може збільшуватися щоразу, коли

алгоритм обробляє адресу, з якої надіслано кошти на *a*. Для всіх *a* масив *oblig(a)* спочатку встановлюється рівним нулю.

Алгоритм намагається заморозити максимально можливу кількість у кожному вузлі *a*, починаючи з кореня *a0*. П’ятий рядок обчислює τ', кількість, що залишилася для заморожування після того, як доступний RBalance на *a* буде заморожено. Крім того, алгоритм віднімає суму Rbalance, спалену на *a*, тому що це зобов’язання не має передаватися нащадкам *a*. Решта суми, τ', має передаватися як зобов’язання нащадкам *a*. Цикл відбувається у зворотному хронологічному порядку, а саме від останньої транзакції до найстарішої. Це продовжується, доки весь τ' не буде переданий як зобов’язання нащадкам *a*.

3.1.4 Псевдокод алгоритму

CalcFreeze(t0): // транзакція заморожування t0=(v->a0)

L:= [топологічно відсортований список вершин графа зобов'язань]

oblig(a0) := val(t0) // зобов'язання a0 відповідно транзакції t0

Для кожного *a із списку L: // перебір елементів списку L починаючи із кореневого елемента a0*

τ := oblig(a) // загальна сума зобов'язань облікового запису a від його батьківських вершин

toFreeze(a) := min(τ, Bal(a)) // сума для заморожування на аккаунті a

τ’ := τ - toFreeze(a) - burnedAt(a) // сума, що залишилась до заморожування із вирахуванням коштів, що були спалені

якщо τ’ <= 0 : продовжити // завершення для вершини a
E(a) := sort({ t= (a->b) }) // список транзакцій, що виходили із облікового запису a у оберненому хронологічному порядку

Для кожного t=(a->b) із списку E(a) виконати:
    ob := min(τ`, val(t))
    oblig(b) += ob;
    τ` -= ob // зобов'язання одержувача b на ob токенів
if τ` <= 0: завершити
    // всі транзакції для a оброблені

Важливо, що кожна адреса *a* відвідується лише один раз і лише після обробки всіх її батьків. Отже, час виконання є лінійним за кількістю ребер у графі. Точніше, якщо граф мітить V вузлів та E ребер, то час виконання дорівнює O(V + E) завдяки використаній структурі даних, що зберігає ребра в хронологічному порядку.
Після завершення цього алгоритму контракт додасть кількість *toFreeze(a*) до кількості монет, заморожених за адресою *a* (важливо, що *a* може вже мати заморожені монети через попередій процес). Це означає, що наступний переказ з адреси *a* буде невдалим, якщо спричинить зменшення Rbalance *a* нижче спільно замороженої суми.

3.1.5 Алгоритм видалення циклів графу переказів

Алгоритм заморожування використовує топологічно відсортовані вершини графа, топологічна сортировка можлива лише за умови, коли граф є ациклічним. А як було показано на рисунку 3.1, не завжди задовольняється дана умова. Тому використання топологічного сортування потребує деякого препроцесингу вершин.

Для початку необхідно розглянути можливий випадок виникнення циклів у графі переказів коштів. Припустимо, що граф містить транзакцію

$$t = (a0 \text{ -> } a1)$$

$$val(t) = 10$$

внаслідок якої буде надіслано 10 токенів від a0 до a1.

Наступною транзакцією буде

$$t1 = (a1 \text{ -> } a0)$$

$$val(t1) = 3$$

відповідно від a1 до a0 було надіслано 3 токени.

Таким чином дані дві транзакції можуть бути спрощені до однієї вигляду:

$$t3 = (a0 \text{ -> } a1)$$

$$val(t3) = 7$$

Тобто таким чином вдалося спростити граф видаляючи з нього цикли.

Для більш загального опису даного підходу представимо,

$$t0 = (v \text{ -> } a0)$$

як підозрілу транзакцію, що буде оскаржуватися.

На етапі передобробки відбувається аналіз графу переказів починаючи із облікового запису a0. Від a0 розглядаються цикли транзакцій

$$c0, c1, \ldots, ck\text{-}1,$$

де $ci = (bi \text{ -> } bi+1 \bmod k)$

що були опубліковані після підозрілої транзакції.

Нехай *c* - це найменша із транзакцій із циклу, що розриває цикл.

Нехай в $d := val(c)$

Для видалення циклу можна вдатись до видалення транзакції *c* з графу переказів та зменшення кожного значення переказу циклу на величину *d*. Даний процес буде повторюватись до тих пір, поки граф не стане ациклічним. Після чого до нього можна буде застосувати алгоритм топологічного сортування.

3.1.6 Програмна реалізація

Контракт ERC-20 зберігатиме таку кількість інформації яка необхідна для підтримки процесу заморожування. Для цього необхідно ввести структура, яка виконує дві функції:
(i) коли надходить запит про заморозку транзакції, контракт повинен підтвердити, що транзакція відбулася в межах вікна оскарження, і (ii) для адреси *a*, контракт повинен ідентифікувати всі адреси нижчого рівня, які отримали кошти від *a* після того, як відбулася підозріла транзакція.

Для зберігання інформації буде створено структуру Spenditure

Задана кількість блоків буде формувати епоху. Для кожної епохи структура витрат містить перелік усіх транзакцій, які відбулися протягом цієї епохи для кожної адреси джерела. Кожен запис витрат на малюнку містить кортеж із наступними даними
(отримувач, сума, номер блоку). Щоразу, коли контракт обробляє запит на переказ, він додає запис «Витрати» до відповідного масиву в структурі витрат.

Функція *freeze* приймає як аргументи три значення : (епоха, відправник, індекс),

де епоха визначає епоху, коли відбулася спірна операція, відправник ідентифікує платника, а індекс визначає елемент масиву, що містить конкретну структуру витрат. Аргумент індексу надається контрактом управління (governance contract) шляхом off-chain (за межами блокчейну) - пошуку в структурі витрат, щоб знайти спірну транзакцію.

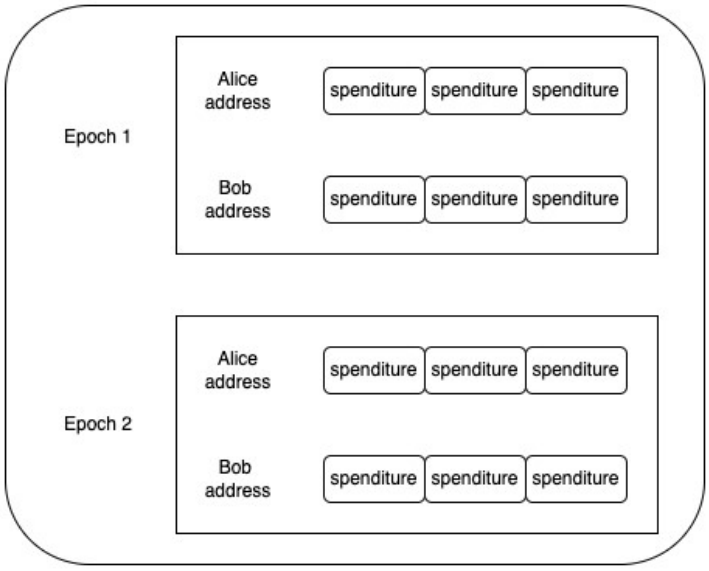


Рисунок 3.2 - Структура збереження інформації про перекази токену ERC20

Функція заморожування перевіряє, чи запитана трансакція знаходиться у часовому вікні оскарження, і якщо так, запускає алгоритм заморожування, використовуючи структуру даних про витрати, щоб визначити ребра графа. Крім того, усі ці суми заморожування записуються у другому вторинному масиві, який називається вимогами(claims), який індексується щойно згенерованим 256-бітним ClaimID. Якщо жертва перемагає в суді, цей масив використовується для переказу правильної суми з кожної підозрюваної адреси жертві. Функція заморожування повертає ClaimID, який вказує, як скасувати транзакцію.

3.1.7 Функція очистки

Щоб мінімізувати зберігання надлишкової інформації у блокчейні, контракт містить функцію очистки, яка враховує епоху та масив адрес. Функція видаляє всі записи масивів витрат із вказаною епохою та адресами відправника для епох, для яких минуло вікно оскарження. Для кожного видаленого запису зазначена сума витрат переноситься з R-балансу облікового запису на його NR-баланс. Кожен може викликати функцію очищення та звільнити мережеву пам’ять, яка більше не потрібна.

3.1.8 Обернення транзакції

Після завершення судового процесу над транзакцією через контракт управління викликається функція rejectReverse або функцію reverse контракту.

- Функція скасування для ERC-20 приймає як вхід ClaimID, який є індексом у масиві претензій. Запис вказує на зобов’язання кожного підозрілого облікового запису в спірному переказі. Функція переказує вказану суму з Rbalance підозрілих рахунків початковому власнику та очищає замороження.
- Функція для скасування ERC-721 приймає як вхідний аргумент TokenID та індекс у масиві власників. Це вказує на власника оскаржуваної операції. Потім актив передається початковому власнику, і актив розморожується.

3.2 Процесс суддівства

У описаній реалізації взаємозамінних та невзаємозамінних токенів важливе місце займає група процесів суддівства, тобто все від початкового підбору групи суддів до подальшої їх координації та комунікації. Дана тема є дуже комплексною так як охоплює багато суміжних дисциплін: економіка, біхевіоризм, теорії ігор тощо. Тому у даній частині буде надано один із можливих варіантів вирішення цього питання. Хоча кінцеві користувачі матимуть змогу самостійно обрати/розробити власний підхід до суддівства через відкриту природу блокчейн-інфраструктур.

Деякі з ключових питань у розробці цього процесу:

- як обираються судді,
- як судді отримують винагороду та
- як перешкодити суддям, які поводяться неприпустимо, наприклад суддям, які беруть хабарі або приймають навмисне некоректне рішення щодо спірних операцій.

Цей процес регулюється контрактом управління. Один контракт управління може регулювати багато контрактів взаємозамінних та невзаємозамінних токенів. Судді можуть голосувати on-chain або off-chain, тобто за допомогою внутрішніх транзакцію у блокчейн-протоколі або за допомогою інструментів поза блокчейном, що потім транслюються у мережу.

Коли буде зібрано достатню кількість голосів за справою, через контракт управління викликається відповідна функція в контракті, щоб задовольнити справу, або її закрити. Для забезпечення нейтральності суддів, важливо, щоб голосування залишалося таємним, доки не буде

зібрано достатньо голосів. Це можна зробити, наприклад, за допомогою певної схеми голосування «здійснення та розкриття» або будь-якої іншої напівприватної схеми голосування.

3.2.1 Обрання суддів

У даному процесі передбачається достатньо велика кількість доступних суддів. Вони отримуватимуть винагороду за свою роботу. При появі запита на заморожування підозрілої транзакції, контракт управління випадковим чином обирає набір суддів із пулу.
Кількість необхідних суддів може бути фіксованим, однак також може збільшуватись при зростанні цінності спірної транзакції. Таким чином можна буде підвищити середню швидкість прийняття суддівського рішення.

Для генерації випадковостей може бути застосовано метод Beacon randomness.
Обраний пул суддів буде приймати рішення відносно початкового запиту на заморожування, а також у подальшому прийматиме рішення щодо схвалення або скасування повернення коштів.

Залишається відкритим питанням хто зможе вступити до даного складу суддів. Адже це може стати причиною хибності прийнятих рішень. Одним із можливих варіантів може бути звичайний KYC процесс (Know Your Customer), або більш складний та децентралізований процес децентралізованої ідентифікації[22] (DID - Decentralised Identification).

3.2.2 Оплата суддям

Для ініціації процесу заморожування сторона, що подає запит прикріплює певну суму коштів, у подальшому ці надіслані кошти можуть бути використані як винагороди для суддів.
При компенсації суддям передбачається те, що вони утримують кошти за віддані голоси.
Важливо відмітити, що розмір компенсації буде залежати від прийнятого рішення при голосуванні.

Якщо судді схвалюють запит на заморожування, сума коштів ставки позивача за вирахуванням комісії залишається зафіксованою в контракті управління. Якщо пізніше позивач програє суд, заставлена сума може компенсувати зусилля відповідача. Якщо позивач виграє судовий процес, він може повернути свою заставлену частку. Важливо відзначити, що якщо судді відхилять запит на початкове заморожування, сума застави позивача за вирахуванням фіксованих гонорарів суддів має бути спалена.

3.2.3 Оплата приорітету

Хоча жертви повинні надати мінімальну ставку разом із запитом на заморожування, вони можуть за бажанням надати додаткову ставку, якщо того захочуть. Ця додаткова «підказка» від жертви потенційно може вказувати на пріоритетність справи. Судді могли розглядати справи за пріоритетністю, а не за хронологією.

3.2.4 Методи сприяння суддівської чесності

Суддя може не проголосувати вчасно, якщо це відбувається систематично, то контракт управління може вилучити такого суддю з пулу.

Складнішим питанням є випадок хабарництва. Коли хтось намагається підкупити суддів для прийняття потрібного рішення. Існують декілька технічних вирішень даного питання.
Одним з таких рішень є збереження у секреті обраний пул суддів по певній справі. Суддя знає, що він є обраним, однак він не знає інших учасників голосування. Після завершення процесу голосування інформація розкривається із доказами того, що правильний пул суддів віддав свої голоси. Важливо, що докази виявляються на основі опублікованих даних від обраних суддів, а не набору довірених осіб. Таким чином, сторона, яка бажає підкупити суддю, не знатиме, кого підкупити, оскільки набір суддів розкривається лише після того, як усі вони проголосували. Це ускладнює підкуп суддів.

3.3 Модифікації

3.3.1 Використання on-chain інструментів автоматизації

У описаній концепції зворотних токенів має місце взаємодія сторін, таких як потерпілий та суддівський пул. Через специфіку функціонування блокчейн протоколу Ethereum, за якої ініціювати зміни даних у мережі можна лише за рахунок транзакції, підписаної EOA (external owned account) виникає можлива проблема, коли запит на заморожування, що надіслав потерпілий не буде розглянуто, через необхідність проведення для цього додаткових транзакцій. Вирішенням даного питання може стати використання on-chain інструментів автоматизації процесів.

Принцип використання такого підходу полягає у наступному. Інструмент автоматизації підключається до контракту управління. Нехай суб’єкт, що відповідає за моніторинг вхідних запитів має називається Keeper. Основною його задачею буде аналіз стану пула вхідних запитів у кожному блоці. Далі у випадку появи нових необроблених запитів даний Keeper надсилає транзакцію, що ініціює процес розгляду вхідного запиту заморожування.
У додатку можна побачити, яким чином підключається Keeper до існуючого контракту. (Додаток А)

На рисунку зображено схему взаємодії у разі підключення інструменту автоматизації.

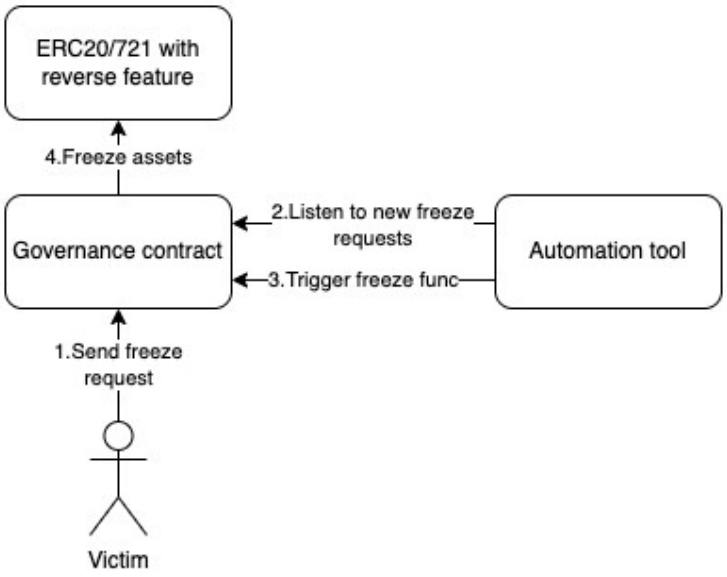


Рисунок 3.3 Схема взаємодії контрактів за наявності інструменту автоматизації

Можливою причиною використання такого підходу є необхідність платити комісію за транзакцію для підтвердження вхідних запитів. Через це

гіпотетично може виникнути ситуація за якої, запити з невеликою сумою залогу, однак достатньою для розгляду, можуть довгий час перебувати у пулі без розгляду суддями, тобто такий запит буде ні спростованим, ні підтвердженим, що однозначно стане на заваді масової інтеграції реверсивних токенів.

Прикладом інструменту автоматизації може стати вже готове рішення від іменитих компанії чи власноруч написаний інструмент. Проблемою останнього є необхідність додаткових аудитів безпеки та підтримання цілої інфраструктури, що для невеликих проектів може бути тягарем. Окрім цього дана інфраструктура матиме вразливість у вигляді централізації, що є проблемою як для можливих атак на неї так и зловживань зі сторони власників.

Тому найоптимальнішим варіантом для невеликих та середніх проектів/протоколів є використання засобів, що стали стандартами індустрії, наприклад, Chainlink Keeper.
Даний засіб автоматизації дає змогу налаштувати смарт-контракт таким чином, щоб виконання виконувалось за одним із сценаріїв :

1. На основі часового проміжку
2. На основі запрограмованої логіки

Для випадку із реверсивними токенами доцільним буде використання саме другого варіанту.
Для цього у контракті управління (Governance contract) будуть наявні декілька структур, одна з них - PendingRequests та ActiveRequests.
Обидві структури будуть зберігатися у вигляді масиву вхідних запитів або у вигляді мапінгу.

Перша структура зберігатиме нові запити, які ще не були оброблені, друга - запити, що були оброблені та чекають на рішення: прийняти чи відмовити.

Таким чином Keeper буде проводити моніторинг структури PendingRequest перевіряючи його стан у кожному блоці. У випадку появи нової заяви буде автоматично сформовано пул суддів та даний запит перейде до структури ActiveRequests, в той же час PendingRequest буде очищено.

Окрім цього важливо відмітити використання автоматичних транзакцій для випадку, коли справа про повернення коштів завершена і необхідно очистити усі дані, що накопичились в процесі, адже для виклику даної функції необхідно заплатити комісію, тому звичайні користувачі не будуть мати мотивацію для цього.

3.3.2 Процес відбору суддів

Для більшої прозорості та чесності для кожної справи про повернення коштів необхідно формувати випадкові набори суддів. Таким чином є необхідність у засобі, що надав би можливість отримувати випадкові значення.
Наразі для отримання випадкових значень у блокчейн протоколах використовуються декілька підходів: RANDAO[20], RANDAO + VDF[21], порогові підписи.

Найбільш простим варіантом є саме RANDAO, його суть полягає у тому, що усі учасники мережі локально обирають псевдовипадкове число, після цього кожен учасник надсилає геш-значення отримане із цього числа. Далі

кожен із учасників по черзі розкривають ці числа та виконують над розкритими значеннями операцію XOR, результат операції і стає результатом роботи даного протоколу.

Однак для більшої простоти реалізації та підтримки проекту має місце використання перевірених on-chain засобів генерації випадкових значень. Прикладом може бути сервіс VRF (Verifiable Random Function) від Chainlink.
VRF від Chainlink - це генератор випадкових чисел, чесність якого можна довести за допомогою криптографічних механізмів. Для кожного запиту VRF генерує одне або декілька випадкових значень та криптографічний доказ того, яким чином дані значення були отримані. Даний доказ публікується у блокчейні та перевіряється on-chain перед тим як його буде використано якимось смарт-контрактом. Даний процес надає можливість перевірити, що при генерації результату ніяка третя сторона не вмішалася задля маніпуляцій над ним.

На рисунку 3.4 зображено схематично яким чином буде виконуватись взаємодія контрактів з урахуванням сервісу отримання випадкових чисел та інструменту автоматизації, що був описаний раніше.

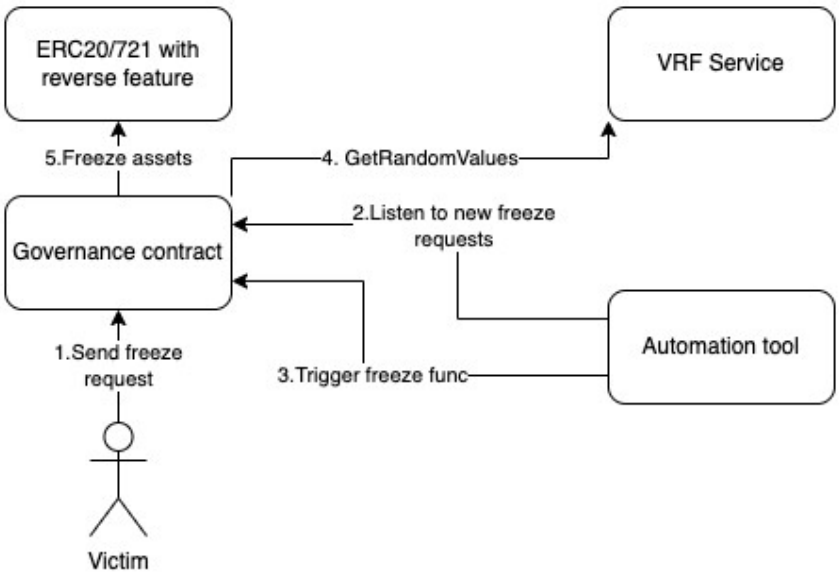


Рисунок 3.4 Схема взаємодії за наявності інструменту автоматизації та сервісу генерації випадкових значень

3.3.3 Порівняння із стандартним варіантом реалізації токенів

Так як для функціонування реверсивної версії токена необхідна деяка кількість додаткових даних, що зберігаються у мережі, важливо порівняти витрати на виконання звичайних функцій, таких як transfer, transferFrom, approve для контарктів ERC20 та ERC721.

У блокчейн протоколах виконання транзакцій потребує оплату за використані потужності. Задля вимірювання використаних ресурсів використовується таке поняття як газ. Газ - це одиниця обчислення. Кожної операції всередині EVM(Ethereum Virtual Machine) відповідає певна кількість газу, що витрачається при виконанні.
Таким чином вартість виконання транзакції обчислюється наступним чином:

$$tx\_cost = gas\_amount * gas\_price,$$

де gas_amount - це кількість одиниць газу, що будуть витрачені при виконанні функції,
gas_price - ціна обчислювальної одиниці (газу).

Отже із рівняння вище можна зробити висновок, що від складності обчислень залежить кінцева вартість транзакції. Таким чином це безпосередньо впливає на кінцевого користувача.

Нижче наведено таблиці (Таблиця 3.1, Таблиця 3.2) із порівнянням витрат газу для базових функцій у звичайному варіанті реалізації токенів та реверсивному варіанті. (Додаток Б)

Таблиця 3.1 - Порівняння використання газу для функцій ERC721

Функція	Standart ERC721	ERC721 Reversible
approve	48719	61523
transferFrom	60124	129437

Таблиця 3.2 - Порівняння використання газу для функцій ERC20

Функція	Standart ERC20	ERC20 Reversible
transfer	34310	191838
approve	46248	46223
transferFrom	42184	200218

Із наведеної таблиці бачимо, що для контракту невзаємозамінного токену для функції approve вартість зросла, однак у незначному обсязі, близько

26%. Однак вплив на функцію transferFrom значно більший, витрати зросли на 115%, що стає доволі відчутним на фоні звичайної реалізації.

Для контракту взаємозамінного токену (ERC20) ситуація наступна, виконання функції approve не змінилося у вартості, однак вартість виконання функції transfer зросла в 5,6 разів, а transferFrom у 4,7 рази.

Враховуючи поточну вартість газу, що становить близько 17 gwei, звичайна вартість transfer дорівнює 1.16$, тобто модифікована версія буде коштувати близько 6,5$, що однозначно стане перепоною для впровадження даної реалізація як стандарту індустрії.
Однак дана проблема викликана високою вартістю газу у мережі Ethereum, що є мережею першого рівня. В той же час існують рішення другого рівня (L2 solution), такі як Polygon, Optimism, ZKSync, що мають значно нижчу вартість виконання транзакцій, мова йде на порядки. Так, наприклад, виконання такої ж транзакції з переказу коштів від одного гаманця до іншого у мережі Polygon коштуватиме близько 0.003-0.005$.

З усього вищесказаного можна зробити висновок, що використання даного стандарту наразі є можливим та бажаним саме у EVM-сумісних блокчейн протоколах через їх значно більш дешевшу вартість транзакцій.

3.3.4 Обмеження мережі

Будь-яка мережа має свої обмеження, а блокчейн-протоколи через свою розподілену природу подібні обмеження примножують. Так у головній мережі Ethereum із впровадженням EIP-170[18] було обмежено максимальний розмір контракту, що може бути завантажений у мережу. Дане обмеження склало 24 кілобайти на один контракт.

Поточна реалізація реверсивних токенів не задовольняє дані вимоги навіть із увімкненим оптимізатором компілятора, що дозволяє зменшити кінцевий розмір байткоду. Однак вирішенням цієї проблеми є використання проксі-контрактів, а саме EIP-2535 Diamonds, Multi-Faucet Proxy[19].
Цей стандарт дозволить вирішити питання із обмеженнями розміру кінцевого контракту.

На рисунку схематично зображено принцип роботи даного механізму [24].

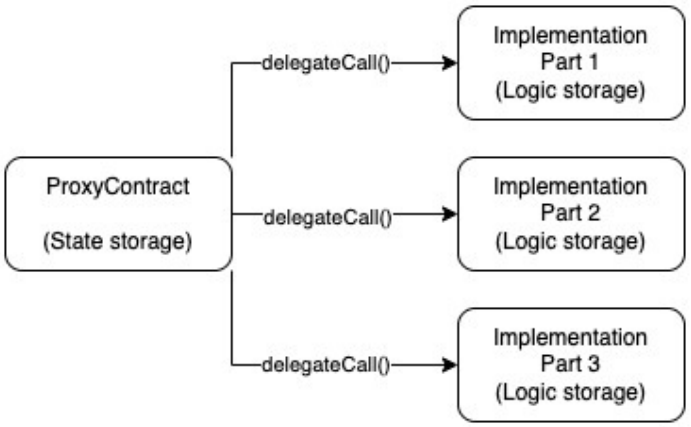


Рисунок 3.5 Схема роботи проксі контракту Diamond Proxy

Для функціонування використовується декілька контрактів. Найголовніший із них - ProxyContract, безпосередньо до нього будуть звертатись інші застосунки або користувачі. Даний контракт зберігає стан, у нашому випадку це вся необхідна інформація про баланси користувачів тощо. Для виконання операцій над цим станом ProxyContract за допомогою делегованих викликів звертатиметься до контрактів, що містять логіку, на рисунку 3.5 вони зображені як Implementation Part n.

Таким чином усю необхідну логіку реверсивних токенів можна розбити на декілька контрактів, до яких буде звертатись контракт, що зберігає стан. Це вирішує проблему максимального розміру контракту у мережі. Додатковою перевагою даного підходу є можливість оновлювати логіку контракту незважаючи на незмінний характер блокчейну, це надасть можливість у разі необхідності додати нові функції чи виправити наявні у разі виявлення помилок.

Висновки до третього розділу

У даному розділі було надано опис програмної реалізації запропонованої моделі реверсивних токенів. Для функціонування даної моделі, стандартні реалізації токенів мають бути модифікованими та містити додаткові дані. Як показало дослідження із впливу цих модифікацій на вартість виконання даних транзакцій, дана модель має деякі обмеження для кінцевого користувача для мереж першого рівня, однак має великий потенціал для виконання у мережах другого рівня як от Polygon чи ZkSync.
Додатково було описані можливості з інтеграції таких рішень, як інструменти автоматизації виконання у мережі та on-chain генератор випадкових значень. Дані модифікації контракта управління здатні значно підвищити ефективність процесу суддівства та повернення токенів.

4 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

4.1 Опис ідеї стартап-проекту

Починаючи розробку будь-якого стартап-проекту необхідно перш за все провести аналіз наявних продуктів на ринку, що в тій чи іншій мірі можуть створювати конкуренцію. Окрім цього важливо зрозуміти та вміти пояснити кінцевому користувачеві необхідність використання вашого продукту чи послуги.

У таблиці 4.1 наведено опис ідеї, напрямку та вигоди стартап-проекту

Таблиця 4.1 - Опис ідеї, напряму та вигоди

Зміст ідеї	Напрям використання	Вигоди для кінцевого користувача
Створення децентралізованого обмінника у якому будуть функціонувати лише реверсивні токени, що потенційно призведе до підвищення якості використання застосунку кінцевими користувачами	Звичайні користувачі чи компанію зможуть використовувати даний застосунок як альтернативу централізованим сервісам	Відсутня необхідність у звернення до третьої сторони задля проведення процесу обміну.
	Інтеграція із іншими децентралізованими застосунками	Більша гнучкість при виборі сервісів.
	Можливість стати провайдером ліквідності	Провайдери ліквідності отримають змогу

		заробляти за рахунок надання своїх коштів

Таблиця 4.2 Визначення сильних, слабких та нейтральних характеристик ідей проекту

№ п/п	Техніко-економічні характеристики ідеї проекту	Потенційні товари конкурентів	W(слабка сторона)	N(нейтральна сторона)	S(сильна сторона)
1.	Відсутність необхідності покладатися на третю сторону при обміні коштів	Надає подібний сервіс	Можлива недостатня кількість ліквідності на початку	Можливість обміну валют	Використання реверсивних токенів
2.	Використання у основі реверсивних токенів	Відсутні сервіси, що б використовували токени із можливістю реверсу	Додаткові накладні витрати при користуванні реверсивних токенів	Не змінюється дизайн взаємодії із сервісом для кінцевого користувача	Можливість повернення токенів у разі шахрайських дій
3.	Можливість надавати ліквідність у пули ліквідності задля отримання прибутку	Більшість із існуючих AMM мають дану можливість	Можливі наявні “непостійні втрати” через невелику ліквідність пулу	Немає	Можливість отримати дохід від надання ліквідності

4.2 Технологічний аудит ідеї проекту

Даний розділ містить опис аудит технології для реалізації ідеї

Таблиця 4.3 - Технологічні можливості здійснення ідеї

Ідея	Наявність технологій	Технологія та реалізація	Доступність технології
Створення децентралізованого обмінника	EVM - сумісні блокчейни	Uniswap, Curve,	Протоколи та стандарти є відкритими
Надання можливості надавати ліквідність у пули	Необхідна розробка	Стандарти ERC20/ERC721	Стандарти є відкритими

4.3 Аналіз можливості запуску стартап-проекту на ринок

Даний розділ містить аналіз попиту, його наявність, динаміку, обсяг. Проаналізовані можливості ринку.

Таблиця 4.4 - Характеристика потенційного ринку проекту

№ п/п	Показники стану ринку	Характеристика
1.	Кількість гравців	10
2.	Загальний об’єм ринку, млн ум.од	1000 млн ум.од
3.	Динаміка ринку	Зростає

4.	Наявність обмежень для виходу	Наявні
5.	Специфічні умови для сертифікації чи дотримання стандартів	Наявні

З результатів можна дійти висновку, що наразі ринок децентралізованих обмінників, та безпосереднього блокчей-протоклів зростає, не дивлячись на складні часи у світовій економіці. Однак важливо дотримуватись сертифікацій тих юрисдикцій, де буде працювати даний сервіс через політику, що спрямована на протидію відмиванню коштів.

Надалі надано інформацію про потенційних клієнтів даного сервісу

Таблиця 4.5 - Потенційні клієнти продукту

№ п/п	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних цільових груп	Вимоги споживачів до товару
1.	Відсутність децентралізованих обмінників на основі смарт-контрактів, що б надавали можливість у разі потреби провести процес повернення	1. Звичайні користувачі, що оперують невеликими сумами коштів 2. Інституційні інвестори, для яких може бути цікава дана ідея для мінімізації втрат	Перша та друга група користувачів головним чином відрізняються сумами, над якими вони оперують, і тим самим по різному впливають на	Захист від можливих атак, як економічного так і технічного характеру. Можливість проведення процесу оскарження рішень

	коштів		ринок. У випадку других вплив може бути доволі значущий	

Таблиця 4.6 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція
1.	Відсутність попиту	Стан ринку може бути незадовільний, через що можливий спад економічної діяльності	Очікування відновлення задовільного стану ринку
2.	Відсутність ліквідності	Розрахункові пули для пар токенів можуть мати недостатній об'єм задля забезпечення необхідних умов функціонування	Додаткове вливання коштів у пули ліквідності для створення можливості їх використання
3.	Непостійні втрати	Через децентралізований характер застосунку, що працює на основі пулів ліквідності можливий перегин у бік одного з токенів, через що провайдери ліквідності отримають неспівпадіння балансів	Додаткове фінансування пулів для стабілізації курсів. Збільшення ліквідності пулів для підвищення їх стійкості до змін обмінних курсів

Таблиця 4.7 - Фактори можливостей

№ п/п	Фактор	Зміст можливостей	Можлива реакція
1.	Відсутність децентралізованих обмінників, що оперували б із реверсивними токенами	Наразі відсутні сервіси, що б надавали можливість проводити обмін реверсивних версій токенів	Розширення обмінних пар. Створення додаткової ліквідності обмінним пулам

Таблиця 4.8 - Ступеневий аналіз ринкової конкуренції

Особливість конкурентного середовища	Як проявляється особливість	Дії компанії для втримання конкуренції
1. Тип конкуренції чиста	Відсутні монополісти у даній галузі	Надання послуг з обміну валют
2. Рівень конкурентної боротьби національний	Надання послуг на території України та за її межами	Впровадження більш досконалих систем моніторингу
3. За галуззю міжгалузева	Можливість використання у галузях із різних сфер	Отримання клієнтів із різних сфер
4. За видами товарів товарно видова	Сервіс надає послуги користувачам із використанням реверсивних токенів	Надання обмінних послуг
5. За характером	Впровадження	Підтримка реверсивних

конкурентних переваг нецінова	реверсивних токенів, що надають можливості із безпеки від шахрайства	токенів
6. За інтенсивністю марочна	Послуги, що надаються є привабливими	Покращення користувацького досвіду

Таблиця 4.9 - Аналіз галузевої конкуренції за М.Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти в галузі	Постачальники	Клієнти	Товари замінники
	Немає	Інші децентралізовані обмінники	Звичайні користувачі та інституційні інвестори	Звичайні рядові користувачі та інституційні інвестори	Відсутні
Висновки	Інтенсивність виходу на ринок	Висок можливість виходу на ринок	Не є перепоною для виходу на ринок	Зацікавлені в ефективності роботи обмінника із реверсивними токенами	Відсутні обмеження через товари-замінники

Дивлячись на стан ринку цей проект може мати можливості виходу та подальшого розвитку. Основною сильною стороною є унікальність послуги у контексті реверсивних токенів.

Таблиця 4.10 - Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обгрунтування
1.	Якість продукту	Продукт показав високу якість
2.	Наявність прямих конкурентів	Наразі відсутні прямі конкуренти, що б використовували той же технологічний підхід
3.	Масштабованість продукту	Продукт у разі потреби може масштабуватись за рахунок додавання коштів у пули ліквідності

Таблиця 4.11 - Порівняльний аналіз сторін проекту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів						
			-3	-2	-1	0	+1	+2	+3
1.	Відсутність прямих конкурентів на ринку	15							+
2.	Гнучкість методу	18					+		
3.	Застосування для компаній із різних галузей	13						+	
4.	Ціна послуги	17							+

Таблиця 4.12 - Порівняльний аналіз сторін проекту

Сильні сторони : унікальність запровадження реверсивних токенів, що дозволить підвищити користувацький досвід кінцевим користувачам	Слабкі сторони: необхідність побудови додаткової інфраструктури задля забезпечення ефективного відбору суддів та обробки запитів
Можливості: можливість функціонування у контексті із іншими децентралізованими застосунками, що дозволить будувати більш стійкі зв'язки всередині сфери	Загрози: проблеми пов’язані із зависокою вартістю газу у рішеннях першого рівня

На основі проведеного SWOT-аналізу було розроблено альтернативи для ринкової поведінки при виведенні проекту на ринок та для його оптимального функціонування.

Таблиця 4.13 - Альтернативи для виходу на ринок

№ п/п	Альтернатива - орієнтований перелік заходів повеедінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Проведення рекламних заходів	Висока	до 9 місяців
2.	Впровадження програми для розширення ліквідності	Висока	до 12 місяців
3.	Інтеграція із іншими застосунками	Середня	до 6 місяців
4.	Реалізація нового функціоналу	Висока	до 18 місяців

4.4 Розроблення ринкової стратегії проекту

Перш за все необхідно окрслити межі ринку функціонування даного сервісу. У Таблиці 4.14 приведено аналіз групи користувачів

Таблиця 4.14 - Аналіз груп користувачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах груп	Інтенсивність конкуренції в секторі	Простота входу
1.	Звичайний користувач	Клієнти зацікавлені у даному продукті	Середній	Низька	Середня, ринок доволі сформований
2.	Інституційні інвестори	Зацікавлені у можливості надання ліквідності	Середній	Низька	Середня

Таблиця 4.15 - Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекта	Стратегія охоплення ринку	Ключові конкурентноспроможні позиції	Базова стратегія розвитку
Розвиток завдяки інтеграціям з існуючими проектами	Інтеграції, рекламна кампанія	Великі перспективи	Стратегія розвитку через актуальність продукту

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

Чи є проект першопроходцем ринку	Чи буду шукати нових споживачів чи збирати існуючих	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
Ні	Шукати та збирати	Частково, використання АММ	Стратегія поміркованого лідерства

4.5 Розроблення маркетингової програми проекту

У даному розділі розглянуто розробку та формування концепції кінцевого продукту, що була сформована на основі визначених аспектів конкурентоспроможності товару

Таблиця 4.17 - Визначення ключових переваг концепції потенційного товару

№ п/п	Потреби	Вигоди, що пропонує товар	Ключові переваги над конкурентами
1.	Можливість обміну реверсивних токенів	Наявність можливості працювати із реверсивними токенами	Наразі немає конкурентів, що б надавали таку можливість
2.	Можливість надавати ліквідність	Можливість отримати вигоду із надання ліквідності	Робота із реверсивними токенами

Таблиця 4.18 - Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікації клієнтів	Ключові пропозиції для позиціонування	Завдання рекламного повідомлення
Прогресивна поведінка	Пошта, месенджери	Високий рівень користувацького досвіду, можливості обернення токенів	Висвітлення основних характеристик продукту та того, що його відрізняє від конкурентів

Висновки до четвертого розділу

У даному розділі було проведено аналіз перспектив для запуску проекту децентралізованого обмінника на основі реверсивних токенах, що надало б кінцевим користувачам даного сервіса додаткові гарантії безпеки. Після отриманих результатів було зроблено висновок про доцільність про подальшу розробку даного проекту та його запуск.

ВИСНОВКИ

Дана дипломна робота спрямована на дослідження існуючих версій стандартних реалізацій смарт-контрактів мережі Ethereum для виконання на EVM. Для розгляду було обрано два основних стандарти токенів у екосистемі - ERC20 та ERC721, взаємозамінний та невзаємозамінний токени відповідно. Де-факто вони є фундаментом будь-якого децентралізованого застосунку у блокчейн-протоколі Ethereum. Не дивлячись на це дані стандарти не позбавлені недоліків, що часом стають причиною великих фінансових втрат. Саме тому дана робота покликана надати можливий із варіантів поліпшення цієї ситуації.
Для вирішення цього питання було проаналізовано наявні реалізації та стандарти контрактів, їх слабкі місця та компроміси реалізації.
Задля загального розуміння сфери у першому розділі було надано загальні обриси функціонування блокчейн-протоколів, їх особливості та обмеження.

У другому розділі було введено концепцію токенів, що можуть бути повернені. Були описані основні аспекти моделі токенів, як додаткові функції керування тощо. Описано модель токену за якого для вирішення спірних питань використовується набір незалежних суддів.

В результаті роботи над третім розділом було запропоновано реалізацію взаємозамінних та невзаємозамінних токенів та також приклад реалізації контракту управління. Також було проведено порівняння стандартної реалізації токенів, що не мають змогу реверсу та реверсивних токенів. В результаті порівняннь двох реалізацій було виявлено значні збільшення витрат обчислювальних ресурсів у випадку реверсивних токенів. Це привело до висновку, що поточна реалізація токенів може мати деякі економічні складності у випадку її використання у Ethereum мережі, однак може бути гарно використана у мережах другого рівня, що

функціонують як надбудови на мережею Ethereum, через свої значно менші витрати при виконанні транзакцій.

Додатково у третьому розділі було розглянуто можливі надбудови, модифікації поточної версії токенів для збільшення попиту на даний стандарт та покращення користувацького досвіду за рахунок часткової автоматизації деяких процесів при веденні судової справи.

У четвертому розділі було досліджено можливий варіант запуску децентралізованого обмінника на основі AMM із використання реверсивних токенів, що б надало кінцевим користувачам додаткові гарантії безпеки. Було проведено багатопоказниковий аналіз та зроблено висновок про доцільність подальшої розробки даного децентралізованого застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. *Improvement Proposals*, no. 20, November 2015. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-20.
2. William Entriken, Dieter Shirley, Jacob Evans, Nastassia Sachs, "EIP-721: Non-Fungible Token Standard," *Ethereum Improvement Proposals*, no. 721, January 2018. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-721.
3. Witek Radomski, Andrew Cooke, Philippe Castonguay, James Therien, Eric Binet, Ronan Sandford, "EIP-1155: Multi Token Standard," *Ethereum Improvement Proposals*, no. 1155, June 2018. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-1155.
4. Chainalysis. The 2022 crypto crime report, 2022. [Online serial]. Available : https://go.chainalysis.com/rs/503-FAP-074/images/Crypto-Crime-Report-2022.pdf
5. Malte Moser, Ittay Eyal, and Emin Gun Sirer. Bitcoin covenants. In Financial Cryptography, volume 9604 of Lecture Notes in Computer Science, pages 126–141. Springer, 2016.
6. Ethereum White Paper (A Next Generation Smart Contract & Decentralized Application Platform) by Vitalik Buterin
7. ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER BERLIN VERSION beacfbd – 2022-10-24 DR. GAVIN WOOD
8. Vlad Zamfir, Nate Rush, Aditya Asgaonkar, Georgios Piliouras, “Introducing the “Minimal CBC Casper” Family of Consensus Protocols”, Ethereum Research, November 5, 2018
9. Thomas Kerber , Markulf Kohlweiss , Aggelos Kiayias , Vassilis Zikas, “Ouroboros Crypsinous: Privacy-Preserving Proof-of-Stake”, The University of Edinburgh and IOHK, May 15, 2019

10. Adam Back, «Hashcash — A Denial of Service Counter-Measure», technical report, August 2002
11. Ben Laurie and Richard Clayton, «'Proof-of-Work' Proves Not to Work», WEIS 04.
12. Poelstra, Andrew. “On Stake and Consensus.” (2015).
13. Chaudhry, Natalia and Muhammad Murtaza Yousaf. “Consensus Algorithms in Blockchain: Comparative Analysis, Challenges and Opportunities.” *2018 12th International Conference on Open Source Systems and Technologies (ICOSST)* (2018): 54-63.
14. Sabry, Sana Sabah et al. “The road to the blockchain technology: Concept and types.” *Periodicals of Engineering and Natural Sciences (PEN)* (2019): n. pag.
15. Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, 2009
16. Hayden Adams, Noah Zinsmeister, Dan Robinson, “Uniswap v2 Core”, March 2020
17. Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, Dan Robinson, “Uniswap v3 Core”, March 2021
18. Vitalik Buterin, "EIP-170: Contract code size limit," *Ethereum Improvement Proposals*, no. 170, November 2016. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-170.
19. Nick Mudge, "EIP-2535: Diamonds, Multi-Facet Proxy," *Ethereum Improvement Proposals*, no. 2535, February 2020. [Online serial]. Available: https://eips.ethereum.org/EIPS/eip-2535.
20. randao.org, “Randao: Verifiable Random Number Generation”, September 11, 2017
21. Benjamin Wesolowski, “Efficient verifiable delay functions”, École Polytechnique Fédérale de Lausanne EPFL IC LACAL, Station 14, CH-1015 Lausanne, Switzerland

22. Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, Christopher Allen, “Decentralized Identifiers (DIDs) v1.0”, 19 July 2022

23. Kaili Wang, Qinchen Wang, Dan Boneh, “ERC-20R and ERC-721R : Reversible Transaction on Ethereum”, October 11, 2022

24. Микита Топчій and Леонід Гальчинський. "ПІДВИЩЕННЯ РІВНЯ БЕЗПЕКИ СМАРТ-КОНТРАКТІВ В МЕРЕЖІ ETHEREUM ВІД ШАХРАЙСТВА ЗА РАХУНОК ВИКОРИСТАННЯ РЕВЕРСИВНИХ ТОКЕНІВ." Collection of scientific papers «ΛΌГОΣ» November 11, 2022; Paris, France (2022): 71-77.

ДОДАТКИ

Додаток А
Програмний код

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

interface IERC20ReversibleFeature {
    function freeze(uint256 epoch,address from,uint256 index) external returns (bytes32
claimID);
    function reverse(bytes32 claimID) external;
    function rejectReverse(bytes32 claimID) external;
    function clean(address[] calldata addresses, uint256 epoch) external;
}

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

interface IERC721ReversibleFeature {
    function freeze(uint256 tokenId,uint256 index) external  returns (bool successful);
    function reverse(uint256 tokenId, uint256 index) external returns (bool successful);
    function rejectReverse(uint256 tokenId) external returns (bool successful);
    function clean(uint256[] calldata tokenIds) external;
}

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

struct Owning{
```

```
    address owner;
    uint256 startBlock;
}

struct OwnersQueue{
    uint256 first;
    uint256 last;
    mapping(uint256 => Owning) queue;
}

contract OwnersUtils{

    function _enqueue(OwnersQueue storage _queue, Owning memory _data ) internal {
        _queue.queue[_queue.last] = _data;
        _queue.last += 1;

    }

    function _dequeue( OwnersQueue storage _queue ) internal {
        require(_queue.last != _queue.first, "Queue is empty");
        delete _queue.queue[_queue.first];
    }
    function _get( OwnersQueue storage _queue, uint256 idx) internal view returns ( Owning
memory data ){
        return _queue.queue[idx];
    }

    function _length(OwnersQueue storage _queue) internal view returns(uint256){
        return _queue.last - _queue.first;
    }

    function _getFirst(OwnersQueue storage _queue) internal view returns(uint256){
        return _queue.first;
    }
```

```
    function _getLast(OwnersQueue storage _queue) internal view returns(uint256){
        return _queue.last;
    }

    function _getOwnersQueueArray(OwnersQueue storage _queue) internal view
returns(Owning[] memory){
        Owning[] memory ownersArr = new Owning[](_length(_queue));
        for(uint i = _queue.first; i < _queue.last; i++){
            ownersArr[i - _queue.first] = _queue.queue[i];
        }
        return ownersArr;
    }
}

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

import "./OwnersQueue.sol";
contract ERC721Reversible is OwnersUtils {

    string private _name;

    // Token symbol
    string private _symbol;

    // dispute window
    uint256 public REVERSIBLE_WINDOW = 10000;
    address private governanceContract;
    // Mapping owner address to token count
    mapping(address => uint256) private _balances;

    // Mapping from token ID to approved address
```

```
    mapping(uint256 => address) private _tokenApprovals;

    // Mapping from owner to operator approvals
    mapping(address => mapping(address => bool)) private _operatorApprovals;

    mapping(uint256 => bool) public _frozen;
     // Mapping from token ID to owner address
    mapping(uint256 => OwnersQueue) public _owners;
    event Freeze(address from,address to,uint256 tokenId,uint256 blockNumber,uint256
index);
    event ReverseSuccess(uint256 tokenId, address from);
    event ReverseReject(uint256 tokenId);
constructor(
        string memory name_,
        string memory symbol_,
        uint256 reversiblePeriod_,
        address _governanceContract
    ) {
        _name = name_;
        _symbol = symbol_;
        REVERSIBLE_WINDOW = reversiblePeriod_;
        governanceContract = _governanceContract;

    }

    modifier onlyGovernance() {
        require(
            msg.sender == governanceContract,
            "ERC721R: Unauthorized"
        );
        _;
    }

    function balanceOf(address owner) public view returns(uint256){
```

```
        require(owner != address(0), "Incorrect address");
        return _balances[owner];
    }
    function ownerOf(uint256 tokenId) public view returns(address){
        OwnersQueue storage queue = _owners[tokenId];
        return _get(queue, _owners[tokenId].last - 1).owner;

    }

    function name() public view returns(string memory){
        return _name;
    }

    function symbol() public view returns( string memory){
        return _symbol;
    }

    function freeze(uint256 tokenId,uint256 index) public onlyGovernance returns(bool
status){
        uint256 firstOwners = _getFirst(_owners[tokenId]);
        uint256 lengthOwners = _getLast(_owners[tokenId]);

        require(index >= firstOwners && index < firstOwners + lengthOwners - 1, "Verification
failed");

        address from = _get(_owners[tokenId], index).owner;
        address to = _get(_owners[tokenId], index + 1).owner;
        uint256 blockNumber = _get(_owners[tokenId], index + 1).startBlock;

        require(blockNumber > block.number - REVERSIBLE_WINDOW, "Reverse is no
longer available " );

        _frozen[tokenId] = true;
```

```
        emit Freeze(from, to, tokenId, blockNumber, index);
        return true;
    }

    function reverse(uint256 tokenId, uint256 index) external onlyGovernance returns(bool
status){
        address currentOwner = _get(_owners[tokenId], _getLast(_owners[tokenId]) - 1).owner;
        address initialOwner = _get(_owners[tokenId], index).owner;
        _frozen[tokenId] = false;
        _transfer(currentOwner, initialOwner, tokenId);
        emit ReverseSuccess(tokenId, initialOwner);
        return true;
    }

    function _transfer(
        address from,
        address to,
        uint256 tokenId
    ) internal {
        require(
            ownerOf(tokenId) == from,
            "Incorrect owner"
        );
        require(to != address(0), " Token can not be transfered to null address ");
        require(_frozen[tokenId] == false, " Frozen token can not be transfered");

        // Clear approvals from the previous owner
        // _approve(address(0), tokenId); //TODO : Add approvals feature

        _balances[from] -= 1;
        _balances[to] += 1;
        _enqueue(_owners[tokenId], Owning(to, block.number));

    }
```

```
    function rejectReverse(uint256 tokenId) external onlyGovernance returns (bool status){
        _frozen[tokenId] = false;
        emit ReverseReject(tokenId);
        return true;
    }

    function clean(uint256[] calldata ids) external{
        for(uint256 i = 0; i < ids.length; i++){
            uint j = _getFirst(_owners[ids[i]]);
            while (_length(_owners[ids[i]]) > 1) {
                if( _get(_owners[ids[i]], j + 1).startBlock < block.number -
REVERSIBLE_WINDOW ){
                    _dequeue(_owners[ids[i]]);
                    j++;
                }else {
                    break;
                }
            }
        }
    }


}

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.12;

import "../chainlink/AutomationCompatible.sol";
import "../chainlink/VRFConsumerBaseV2.sol";
import "../chainlink/VRFCoordinatorV2Interface.sol";
import "../interfaces/IERC20ReversibleFeature.sol";
import "../interfaces/IERC721ReversibleFeature.sol";
```

```
interface IJudgePoolsCreator {
    struct GetPoolResponse {
        bytes32 judgesHash;
        address multisigContract;
    }

    function getJudgePools(
        uint256[] memory _randomWords,
        uint32 _poolSize
    ) external returns (GetPoolResponse[] memory);
}

struct Request {
    uint256 id;
    bytes32 tx_hash;
    bytes proof;
    bytes32 judgesHash;
    address multisigContract;
    uint256 collateral;
    address issuer;
}

contract Governance is AutomationCompatible, VRFConsumerBaseV2{
    Request[] pendingRequests;
    mapping(uint256 => Request) activeRequests;
    uint256 numPendingRequests;
    uint256 numActiveRequests;

    bool requestRandomStatus; // true - random words is requested, false - vice versa
    bool isReceivedRandoms; // false - random words are not received yet
    uint256 lastRequestRandomsId; // id of last request for random words

    uint32 poolSize; // size of judges pool for one issuer request
```

```
address tokenContract; // address of reversible token
address poolsCreator; // address of contract responsible for creating judge pools

uint256[] s_words;
VRFCoordinatorV2Interface COORDINATOR;

uint16 requestConfirmation;
uint64 subscriptionId;
uint32 callbackGasLimit;
bytes32 keyHash;

modifier onlyJudges(uint256 _requestId) {
   require(
      msg.sender == activeRequests[_requestId].multisigContract,
      "Governance: Incorrect caller"
   );
   _;
}

constructor(
   address _vrfCoordinator,
   address _token,
   uint32 _poolSize,
   uint16 _requestConfirmation,
   uint64 _subId,
   uint32 _callbackGasLimit,
   bytes32 _keyHash
) VRFConsumerBaseV2(_vrfCoordinator) {
   tokenContract = _token;
   numPendingRequests = 0;
   numActiveRequests = 0;
   poolSize = _poolSize;
   requestConfirmation = _requestConfirmation;
```

```
    subscriptionId = _subId;
    callbackGasLimit = _callbackGasLimit;
    keyHash = _keyHash;
    requestRandomStatus = false;
}

function createRequest(
    bytes32 _txHash,
    bytes calldata _proof
) external payable returns (uint256 id) {
    numPendingRequests++;
    pendingRequests[numPendingRequests] = Request(
        numPendingRequests,
        _txHash,
        _proof,
        "",
        address(0),
        msg.value,
        msg.sender
    );
    id = ++numActiveRequests;
}

function checkUpkeep(
    bytes calldata checkData
) external override returns (bool upkeepNeeded, bytes memory performData) {
    if (pendingRequests.length != 0 && requestRandomStatus == false) {
        upkeepNeeded = true;
        performData = hex"01";
    } else if (
        pendingRequests.length != 0 &&
        requestRandomStatus == false &&
        isReceivedRandoms == true
    ) {
```

```
            upkeepNeeded = true;
            performData = hex"02";
        }
    }

    function performUpkeep(bytes calldata performData) external override {
        if (keccak256(performData) == keccak256(hex"01")) {
            // if queue of input request is not empty _requestRandomWords for generation judge
pools
            _requestRandomWords();
        } else if (keccak256(performData) == keccak256(hex"02")) {
            IJudgePoolsCreator.GetPoolResponse[]
                memory pools = IJudgePoolsCreator(poolsCreator).getJudgePools(
                    s_words,
                    poolSize
                );

            require(pools.length == pendingRequests.length, "Incorrect amount of created
pools");
            _activateRequests(pools);



        }
    }

    function fulfillRandomWords(
        uint256 requestId,
        uint256[] memory randomWords
    ) internal override {
        require(requestId == lastRequestRandomsId, "Incorrect requestId");
        s_words = randomWords;
        requestRandomStatus = false;
        isReceivedRandoms == true;
    }
```

```
    // internal functions
    function _activateRequest(uint256 _id, IJudgePoolsCreator.GetPoolResponse memory
poolResponse) internal {
        Request memory pendingRequest = pendingRequests[_id];
        pendingRequest.judgesHash = poolResponse.judgesHash;
        pendingRequest.multisigContract = poolResponse.multisigContract;
        activeRequests[numActiveRequests + _id] = pendingRequest;
    }

    function _activateRequests(IJudgePoolsCreator.GetPoolResponse[] memory pools)
internal {
        for(uint i = 0; i < pools.length; i++ ){
              _activateRequest(i, pools[i]);
        }
        delete pendingRequests;
        delete numPendingRequests;
        isReceivedRandoms = false;
        requestRandomStatus = false;
        delete lastRequestRandomsId;
    }

    function _requestRandomWords() internal returns (uint256 requestId) {
        requestId = COORDINATOR.requestRandomWords(
            keyHash,
            subscriptionId,
            requestConfirmation,
            callbackGasLimit,
            uint32(pendingRequests.length) * poolSize
        );
        lastRequestRandomsId = requestId;
        requestRandomStatus = true;
        return requestId;
    }
```

```
    // ERC20Reveresible functions

    function freezeERC20(uint256 _requestId, uint256 epoch, uint256 index ) external
onlyJudges(_requestId) returns(bytes32 ){
        address issuer = activeRequests[_requestId].issuer;
        return IERC20ReversibleFeature(tokenContract).freeze(epoch, issuer, index);
    }

    function reverseERC20(uint256 _requestId, bytes32 _claimId) external
onlyJudges(_requestId){
        IERC20ReversibleFeature(tokenContract).reverse(_claimId);
    }

    function rejectReverseERC20(uint256 _requestId, bytes32 claimID) external
onlyJudges(_requestId){
        IERC20ReversibleFeature(tokenContract).rejectReverse(claimID);
    }

    function cleanERC20(address[] calldata addresses, uint256 epoch) external{
        IERC20ReversibleFeature(tokenContract).clean(addresses, epoch);
    }
    // ERC721Reversible functions

    function freezeERC721(uint256 _requestId, uint256 tokenId,uint256 index) external
onlyJudges(_requestId) returns(bool){
        return IERC721ReversibleFeature(tokenContract).freeze(tokenId, index);
    }

    function reverseERC721(uint256 _requestId, uint256 tokenId,uint256 index) external
onlyJudges(_requestId) returns (bool) {
        return IERC721ReversibleFeature(tokenContract).reverse(tokenId, index);
    }
```

```
    function rejectReverse(uint256 _requestId, uint256 tokenId) external
onlyJudges(_requestId){
        IERC721ReversibleFeature(tokenContract).rejectReverse(tokenId);
    }

    function clean(uint256[] calldata tokenIds) external{
        IERC721ReversibleFeature(tokenContract).clean(tokenIds);
    }

}
```

Додаток Б

Програмний код для порівняння споживання газу стандартної реалізації та реверсивної

```
import { ethers } from "hardhat";

async function main() {
  const [ admin, alice ] = await ethers.getSigners();
  const ERC721R = await ethers.getContractFactory("ExampleERC721R");
  const erc721r = await ERC721R.deploy(100, 20000, admin.address);

  const ERC721 = await ethers.getContractFactory("StandartERC721");
  const erc721 = await ERC721.deploy("some_name", "SNN");
  await erc721.deployed();

  await erc721.connect(admin).mint(1);
  await erc721.connect(admin).mint(2);
  await erc721.connect(admin).mint(3);

  /* Function to test
    * transferFrom
    * approve
    * safeTransferFrom
  */
```

```
  // erc721 testing

  const approveGas = await erc721.connect(admin).estimateGas.approve(alice.address, 1);
  console.log("Approve gas ERC721 ", approveGas.toString());

  const transferFromGas = await
erc721.connect(admin).estimateGas.transferFrom(admin.address, alice.address, 1);
  console.log("transferFrom gas ERC721: ", transferFromGas.toString());


  // erc721 testing

  const approveGas2 = await erc721r.connect(admin).estimateGas.approve(alice.address, 1);
  console.log("Approve gas ERC721R", approveGas2.toString());

  const transferFromGas2 = await
erc721r.connect(admin).estimateGas.transferFrom(admin.address, alice.address, 1);
  console.log("transferFrom gas ERC721R : ", transferFromGas2.toString());
}
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});

import { ethers } from "hardhat";
async function main() {
    const [ admin, alice ] = await ethers.getSigners();
    const ERC20 = await ethers.getContractFactory("StandartERC20");
    const ERC20R = await ethers.getContractFactory("ExampleERC20R");
    const erc20 = await ERC20.connect(admin).deploy("someName", "SNN");
    await erc20.deployed();
    await erc20.mint();
    await erc20.connect(alice).mint();
```

```
    const erc20r = await ERC20R.connect(admin).deploy(100000000, 10000, admin.address);
    await erc20r.deployed();
    /*
       Function to compare
          * transfer( recipient, amount )
          * approve ( spender, amount )
          * transferFrom( sender, recipient, amount )

    // erc20

    const transferGas1 = await erc20.connect(admin).estimateGas.transfer(alice.address, 100);
    console.log("Transfer gas ERC20 : ", transferGas1.toString());
    await erc20.connect(admin).transfer(alice.address, 100);
    const approveGas1 = await erc20.connect(admin).estimateGas.approve(alice.address,
10000000000000);
    await erc20.approve(alice.address, 10000);
    console.log("Approve gas ERC20 : ", approveGas1.toString());
    const transferFromGas1 = await erc20.connect(alice).estimateGas.transferFrom(
admin.address, alice.address, 10 );
    console.log("TransferFrom Gas ERC20 : ", transferFromGas1.toString());

    console.log("Reversible feature \n ----------------------------------------------------------");

    const transferGas2 = await erc20r.connect(admin).estimateGas.transfer(alice.address, 100);
    console.log("Transfer gas ERC20R : ", transferGas2.toString());

    const approveGas2 = await erc20r.connect(admin).estimateGas.approve(alice.address,
100);
    await erc20r.connect(admin).approve(alice.address, 100);
    console.log("Approve gas ERC20R : ", approveGas2.toString());

    const transferFromGas2 = await erc20r.connect(alice).estimateGas.transferFrom(
admin.address, alice.address, 100 );
    console.log("TransferFrom Gas ERC20R : ", transferFromGas2.toString());
```

```
}

main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
  });
```