Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра інформатики

Магістерська робота

освітній ступінь – магістр

На тему: **«Децентралізований застосунок на основі блокчейну для голосування»**

Виконав: студент 2-го року навчання,
Спеціальності
122 Комп’ютерні науки
Мороз Андрій Вікторович
Керівник: Гороховський С.С.
кандитат фіз.-мат. наук, доц.
Рецензент Франчук О.В
доцент к.н

Магістерська робота захищена
з оцінкою _____

Секретар ЕК С.А. Мелещенко
“____”_____2022р.

Київ 2022

Зміст

Анотація

Ця дипломна робота має на меті створити децентралізований застосунок на основі блокчейну для відкритого голосування. У цій роботі було виявлено вимоги для такого застосунку. На основі цих вимог був виконаний аналіз існуючих блокчейн платформ які б підходили для проектування та розробки такого застосунку. На обраній платформі було спроектовано архітектуру яка б дозволяла проводити голосування та розроблено proof-of-concept застосунок який би дозволив проводити відкриті голосування на децентралізованій платформі.

Ключові слова: Блокчейн, розподілені системи, децентралізація, електронне голосування.

Вступ

Голосування та вибори є основою сучасних демократій, оскільки саме завдяки цьому процесу громадськість має можливість показати свою політичну позицію та колективно обирати лідерів. Людство активно шукає шляхи вдосконалення процесу голосування від підрахунку піднятих рук і заповнення паперових бюлетенів до електронної подачі голосів заради покращення процесу голосування який колись був трудомістким, ненадійним і схильним до помилок. Нинішні системи голосування продовжують бути темою обговорень у світі та постійно піддаються критиці одні з найвідоміших прикладів: можливі втручання у президентські вибори США 2016 року, фальсифікації виборів у РФ, фальсифікація президентських виборів у Білорусі. Найвідоміший випадок фальсифікацій для України були президентські вибори 2004 року коли екзит-пол не підтверджував рішення центральної виборчої комісії України.

Впровадження електронної системи для різних видів голосування, це, мабуть, найбільш чіткий спосіб викорінити проблеми фізичних фальсифікацій голосів, а також зменшення вірогідності помилки при підрахунку голосів в ручну. Перша країна, яка прийняла електронну систему національних виборів була Естонія [1]. Незабаром послідували Швейцарія та Норвегія, впроваджуючи електронні системи для загальнодержавних [2] та виборів до рад [3].

Ці електронні системи голосування перебувають під пильною увагою дослідників безпеки. Однією з основних критик цих програм є не прозорість, оскільки, наприклад, Естонська система ніколи не публікувала сценарій голосування[4]. Також в риторику критики входила централізація цих систем. Оскільки системи контролюється одним основним суб’єктом і мають декілька векторів нападу на систему як наприклад вразливість до розподіленої атаки на відмову в обслуговуванні або DDoS. DDoS атака тягне за собою унеможливлення системи коректно функціонувати.

Як можливе вирішення недоліків централізованих систем електронного голосування, була запропонована система на основі технології блокчейн [5]. Блокчейн — це розподілена система, однорангова мережа яка керується консенсусом. Така система дозволяє зберігати дані прозорими, та за своєю природою є стійкою до несанкціонованого доступу та DDoS атак.

Розділ 1. Вибори та виборчі системи

1.1 Вибори

Вибори є, мабуть, найбільш очевидним і інтуїтивним засобом, який дозволяє населенню висловлювати свою волю та брати участь у процесі управління. Вибори – це узагальнений процес, який дозволяє особам висловити свої преференції, як правило, шляхом голосування.
Узагальнена мета виборів полягає в досягненні консенсусу, який точно та справедливо відображає уподобання виборців, які беруть участь у голосуванні. Це, на перший погляд, здається тривіальною проблемою, і такою і є в ситуаціях коли виборців або варіантів вибору не багато. Однак вибори стають більш і більше складними процессами в залежності скільки виборців, виборів голосу та цикліп припадають на процесс. Також існують соціальні, математичні та практичні обмеження, під які підпадають всі системи голосування.

1.2 Виборча системи

Виборча система — це поєднання правил, норм та процедур, які визначають, як приймається остаточний результат та коли виборці дійшли до консенсусу. Виборчу систему можна розкласти на три основні компоненти: доступ до виборчого бюлетеня, процесс вибору та алгоритм підрахунку голосів. Ці три компоненти можна з’єднати разом та створити різноманітний спектр виборчих систем з різними характеристиками та властивостями. Вибір та реалізація виборчої системи має безпосередньо важливий вплив на те, як демократичні системи в змозі працювати, а також вирішуються наскільки легітимною є система управління. Рішення щодо детальних впровадження у виборчу систему є одними з найважливіших рішень що зробить будь-яка

демократична організації, оскільки вибір моделі впливає на всі майбутні процеси прийняття рішень і формує модель подальшого управління. Погано розроблена виборча система може мати катастрофічний вплив на стан демократичної організації як у короткій так і у довгостроковій перспективі. Крім того, після обрання виборчої систему може бути важко змінити, оскільки політичні сили уже сформовані та зацікавленні у захисті нинішньої виборчої системи[6].

1.3 Складові виборчої системи

1.3.1 Варіанти вибору

Варіанти вибору виборчої системи — це набір варіантів, які може виборець вибрати з поміж кого чи чого вирішується на виборах. Варіанти виборів можуть бути визначена шляхом первинного голосування, опитування, дебатів тощо.

1.3.2 Виборчий округ

Кількість варіантів виборів, які будуть обрані як переможці має велике значення на виборах. У представницьких демократіях одиниця яка вимірює репрезентативну позицію рахується за кількістю виборчих округів. Вибори, де виборчий округ має репрезентувати один переможний варіант вибору називають одномандантний виборчий округ. Якщо переможних варіантів може бути більше ніж один то це називають багатомандатним виборчим округом.

1.3.3 Виборчий бюллетень

Бюлетень – це засіб, за допомогою якого учасники голосування висловлюють свій голос. Структура виборчого бюлетеня впливає на те, як саме виборець може висловити свій вибір, наприклад, скільки голосів людина може віддати. Виборчий бюлетень напряму залежить від системи підрахунку голосів. Виборчий бюлетень можна описати як структуру даних яку можна використовувати у алгоритмі підрахунку голосів.

1.3.4 Виборча формула

Виборча формула – це процес переведення бюлетенів до фінальних результатів. Виборча формула може характеризуватися або алгоритмом підрахунку голосів або якщо це багатомандантний округ то пропорціональністю.

1.3.5 Метод підрахунку голосів

Метод підрахунку або алгоритм підрахунку голосів дуже близький пов’язаний з тим, як виборець має право маркувати бюлетень, і впливає на те, як потім позначений бюлетень підраховується у фінальний результат. Існують процеси різної складності підрахунки голосів. Загалом, метод підрахунку голосів буде мати прямий і значний вплив на результат виборів.

1.3.6 Пропорційність

Загалом, пропорційність використовується у багатомандатних округах та характеризує, наскільки голоси будуть перетворюватись у варіанти які перемогли. Особливо важливою пропорційність стає з великою кількістю мандатів. Чим ближче за пропорціями виграшні варіанти відносно кількості голосів що за них віддали тим більшою вважається пропорційність системи.

Аналіз загальної кількості голосів і їх співвідношення з врахованими голосами та з «зіпсованими голосами» є найпростішим способом щоб визначити пропорційність, зазвичай виражається як індекс диспропорційності.

1.4 Основні види виборчих систем

Виборчі системи, як правило, поділяються на три сімейства: мажоритарна система, пропорційна виборча система та змішана

1.4.1 Мажоритарна виборча система

Мажоритарна система голосування характеризується тим що вибір буде вважатися переможним лише з абсолютною більшістю голосів, тобто більше 50% голосів. Більшість мажоритарних систем голосування використовують кілька циклів виборів, де менш популярні варіанти видаляються в кожному циклі, щоб сформувати більшість.

1.4.2 Пропорційна виборча система

Метою пропорційної виборчої системи є вивести переможців на виборах, які точно відображають волю народу. Пропорційна система діє шляхом надання поперечного перерізу переможців на виборах, які визначаються пропорційно голосам які за них віддали.

Пропорційні системи мають ряд переваг:

- Це призводить до меншої кількості зіпсованих голосів
- Пропонує групам меншин більше представництва.
- Сприяє довгостроковому політичному здоров’ю та переговорам
- Приводить до більшої участі виборців та легітимності виборів.
- Підтримує більш різноманітний перетин представників.

Серед недоліків можна виділити

- Швидку та послідовну політику може бути важко просувати.
- Під час формування фракцій може виникнути законодавчий тупик.

Розділ 2. Блокчейн

Блокчейн — це розподілена база даних транзакцій, яка відстежує транзакції в постійно зростаючих пов’язаних блоках. Технології блокчейн були використані для створення незмінних публічних систем для відстеження даних про криптовалюти, активи та права на власність. Найбільш відомою технологією блокчейн, яка використовується сьогодні, є біткойн.
Технологія блокчейн існує, щоб забезпечити рішення задачі візантійських генералів в розподіленому середовищі. Це дозволяє сторонам без довіри до один до одного підтримувати базу даних транзакцій. Блокчейн використовує різні алгоритми для оцінки різноманітних транзакцій щоб у кінцевому результаті досягти консенсусу серед учасників системи.
Існують цікаві збіги та паралелі, які можна провести між механізмами консенсусу в комп’ютерних мережах та в демократичних системах.

2.1 Біткоїн

У 2008 була опублікова біла книга “Bitcoin: A Peer-to-Peer Electronic Cash System” від автора з ім’я Satoshi Nakamoto[7]. Ця біла книга описувала ідеї нової форми валюти — біткойна. Ідея валюти була стати першою децентралізованою цифровою валютою яка б не потребувала автентифікації та довіри до інших користувачів. Ця досягається шляхом комбінування криптографії, алгоритму консенсусу, у випадку біткойн алгоритм консенсусу

є proof-of-work, та учасників мережі яких називають “майнерами”. Загалом, блокчейн дозволяє людям, які не довіряють один одному, приходити до консенсусу. Конкретніше у випадку біткойна, платформа дозволяє переслати валюту один одному через децентралізовану платформу.

2.1.1 Топологія мережі

Мережа біткоїн структурована як однорангова мережа, що складається з а різноманітність типів вузлів. Кожен вузол підтримує різні функціональні можливості. Загальний функціонал включав в себе:

- Функціональність майнінгу для створення нових блоків за допомогою вирішення складної математичної задачі.
- Гаманець, який надає користувачам можливість керувати своїми ключами, а також надсилати і отримувати транзакції.
- Повна копія блокчейну, що дозволяє вузлам перевіряти транзакції незалежно від інших вузлів мережі.
- Можливості мережевої маршрутизації, що дозволяє вузлам поширювати транзакції і знаходити нові вузли.

2.1.2 Типи вузлів

Найпоширеніші типи вузлів, класифіковані за відповідним функціоналом[8]:

- *Довідковий клієнт* - містить гаманець, майнер, повну копію блокчейну і функціональність мережевої маршрутизації
- *Повний вузол* - містить повну копію блокчейну та можливість мережевої маршрутизації.
- *Майнінговий вузол* - містить майнер, повну копію блокчейну та можливість мережевої маршрутизації.

- *Легкий гаманець* - містить гаманець та можливість мережевої маршрутизації. Цей тип вузла залежить від повноцінного вузла у оскільки не можуть перевірити транзакцію самі. Такий тип вузла частіше за все можна знайти на мобільних пристроях та пристроях де зберігати повну копію блокчейну не є раціональним.

2.1.3 Основні поняття

Блокчейн біткойн структурований як зв'язаний список блоків, як можна побачити на рис 1. Кожен блок містить набір транзакцій і посилання на попередній блок у ланцюжку. Блоки ідентифікуються за хешем SHA-256 його заголовка.

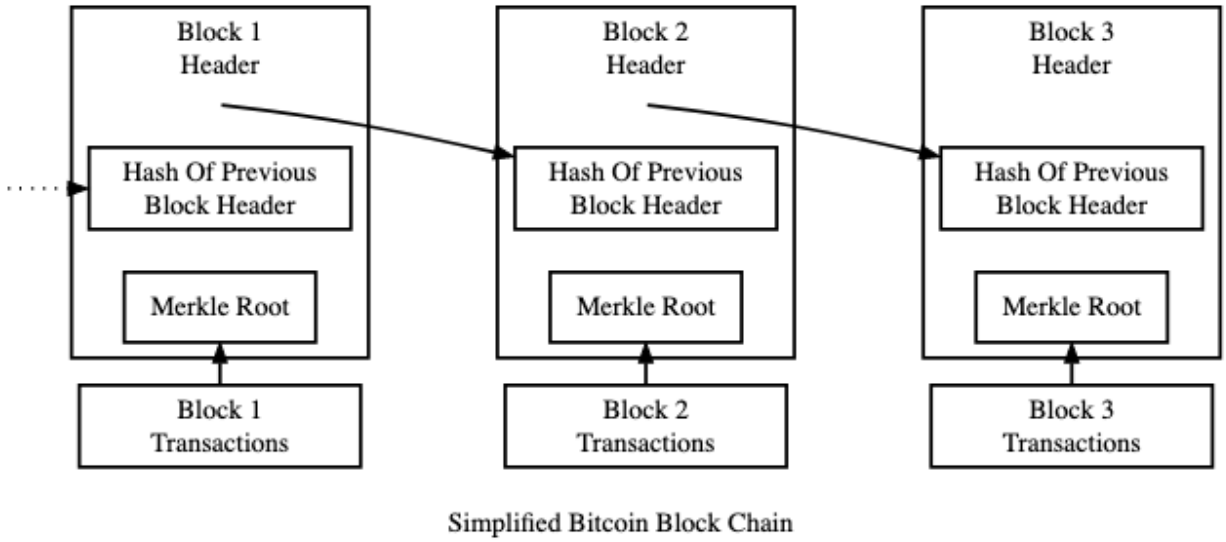

Рис. 2.1 Ланцюг блоків біткойн

2.1.4 Блоки

Блок - основний будівельний блок блокчейну. Блоки служать інструментом відмітки часу для транзакцій у мережі, а також підтверджують що певна

робота була виконана для підтримки безпеки мережі. Вміст блоку вказано в таблиці 2.2.

Розмір	Поле	Опис
4 байти	Розмір блоку	Розмір блоку в байтах
80 байтів	Заголовок блоку	Поля які описують заголовок блоку
1-9 байт	Лічильник транзакцій	Кількість транзакцій у блоці
Змінно	Транзакції	Всі транзакції записані у цей блок.

Таб. 2.1 Вміст блоку біткойн[8]

2.1.5 Заголовок блоку

Заголовок блоку відповідає за відстеження метаданих блоку біткоїн. Він також використовується для представлення блоку шляхом хешування його вмісту алгоритмом SHA-256. Вміст заголовка блоку можна побачити у таблиці 2.2

Розмір	Поле	Опис
4 байти	Версія протоколу	Номер версії для відстеження оновлення програмного забезпечення/протоколу
32 байт	Хеш попереднього блоку	Посилання на хеш попереднього блоку в ланцюжку
32 байт	Дерево Меркла	Хеш кореня дерева Меркла усіх транзакцій цього блоку
4 байти	Позначка часу	Приблизний час створення цього блоку у форматі часу Unix.
4 байти	Складність	Складність цього блоку для консенсусу proof-of-work

4 байти	Нонс(криптографічний нонс)	Лічильник, що використовується для алгоритму proof-of-work

Таб. 2.2 Вміст заголовка блоку[13]

2.1.5 Дерево Меркла

Дерево Меркла — це структура даних, яка дозволяє ефективно перевірити вміст великої кількості даних. Дерева Меркла широко використовуються в однорангових мережах, щоб гарантувати, що блоки даних надходять незмінними та непошкодженими. Дерева Меркла складаються з вузлів хешів. Вони мають унікальну властивість і дозволяють перевірити існування певного хешу у дереві за O(log(n)) час.

2.1.6 Транзакції

В оригінальній білій книзі Сатоші електронна монета визначається як токен, який можна передавати серед ланцюгів цифрових підписів:
“Ми визначаємо електронну монету як ланцюжок цифрових підписів. Кожен власник передає монету наступному власнику, підписуючи хеш попередньої транзакції та публічний ключ наступного власника та запусує їх до кінця монети. Одержувач може перевірити підписи, щоб перевірити ланцюжок власників монети.”[7]
На рисунку 2.2 можна побачити як Власник 1 хешує транзакцію, і надає права власності на електронну монету Власнику 2. Процесс повторюється і Власник 2 підписує транзакцію і передає електронну монету Власнику 3.

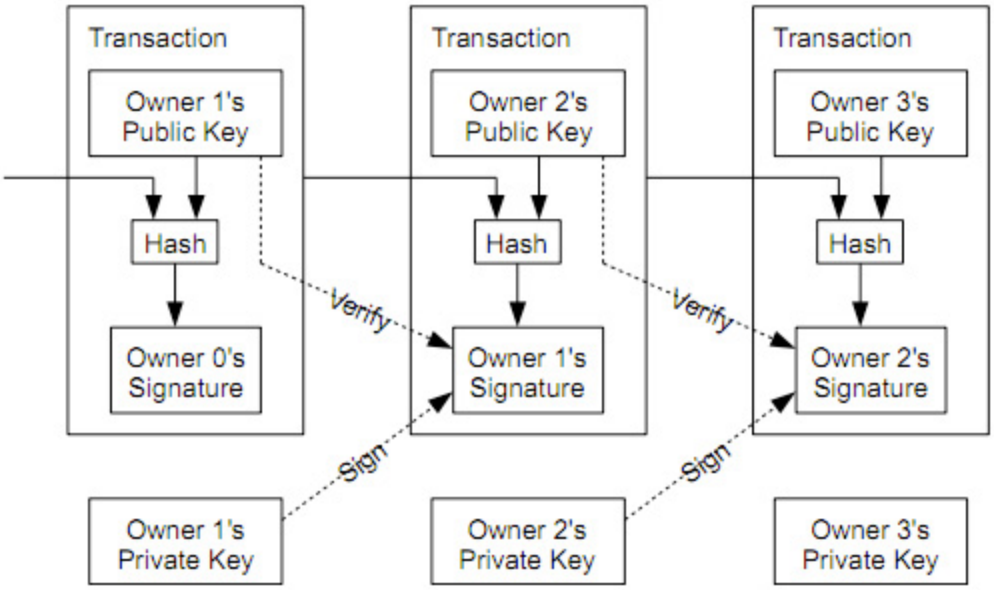

Рис. 2.2 Ланцюжок підписів, що представляють електронну монету

2.1.7 Доказ виконаної роботи

Доказ виконаної роботи(PoW) - це основний алгоритм консенсусу в блокчейн біткойн. PoW схожий на алгоритм hashcash[7, 9]. Метою цього алгоритму є створення проблеми яку легко перевірити на правильність, але важко вирішити спочатку. Алгоритм PoW надає засіб для псевдовипадкового вибору вузлів майнінгу для створення нових блоків транзакцій. Імовірність того, що майнер буде обраний, безпосередньо пов’язана до обсягу роботи, яку виконує майнер.

2.1.8 Алгоритм хешування

Алгоритм PoW залежить від криптографічної функції хешування SHA-256, яка є члена сімейства алгоритмів SHA-2, та видає 256-бітний 32-байтовий хеш-результат. Основні вимоги, яким повинна відповідати криптографічна хеш-функція:

- детермінована, тобто той самий вхід завжди буде давати однаковий вихід
- швидко обчислюється
- неможливо визначити вхід з виходу, тобто невелику зміну в вхідні дані призводять до значних, здавалося б, випадкових змін на виході
- неможливо знайти колізію в отриманих хешах

Ці властивості криптографічних хеш-функцій забезпечують стійкість до зіткнень. Це означає, що обчислювально неможливо знайти вхідні дані які б могли нам підрахувати конкретний хеш-результат. Ця властивість стійкості до зіткнень використовується для побудови псевдовипадковий процес вибору, який визначає, який вузол здатний обробити наступний блок транзакцій.

2.1.9 Складність

Мережа біткойн має складність в 256 біт, 32 байт, це значення перераховується кожні 2016 блоків. Значення перераховується таким чином, щоб псевдо випадковий процес “майнінгу” для створення нового блоку займав приблизно 10 хвилин.

Не дивлячись на те мережа повинна псевдо випадково “обирати” майнерів, насправді ймовірність вибору майнера можна підрахувати як ймовірність поділивши хешрейт майнера на хешрейт всієї мережі: $\frac{\text{хешрейт майнера}}{\text{хешрейт мережі}}$.

Майнери багаторазово обчислюють хеші SHA-256, поєднуючи попередній блок хеш заголовка, поточні дані блоку та нонс. Мета майнера — знайти нонс, який створює а хеш менший за поточну складність. Завдяки властивостям хеш-функції SHA-256 нонс можна підібрати тільки за

допомогою вичерпного пошуку. Як тільки майнер знаходить підходящий хеш, він створює новий блок.

2.1.9 Мережа

Як тільки майнер генерує дійсний блок, він поширює цю інформацію в мережу. Інші вузли перевіряють що блок є дійсним та додають його собі у ланцюг. Одна з транзакцій, яку майнери включатимуть у свої
блок це перша транзакція у блоці яку ще називають coinbase transaction і вона позначає яку нагороду отримав майнер за створення блоку.

2.1.10 Безпека

Всі властивості блокчейну біткойн в поєднанні зі стимулом отримання винагороди забезпечують безпеку у вигляді криптографії, використання електроенергії та обчислювального обладнання. Атаки, які загрожують безпеці мережі, повинні або зламати криптографічні примітиви мережі, або шукати методи зменшення витрат на електроенергію та апаратне забезпечення щоб перевершити хешрейт іншої частини мережі.

2.2 Етеріум

Етеріум(Ethereum) — це система на основі блокчейну, спочатку запропонована наприкінці 2013 року після біткойна та випущена в 2015 році.[11] Ethereum представив кілька нових функцій у своїй мережі, які були недоступні в мережі біткойн. Зокрема, Ethereum пропонує розподілену децентралізовану обчислювальну платформу, повну за Тьюрингом, через віртуальну машину Ethereum (EVM) мета якої забезпечити прикладний рівень, який “працює точно так, як запрограмовано без будь-якої можливості простою, цензури, шахрайства або втручання третьої сторони”.[10, 12]

2.2.1 Валюта

Як і біткойн, блокчейн Ethereum використовує свій власний токен, ефір — також відомий як eth або ETH який виступає як базова валюта цього протоколу блокчейн. Найменша одиниця ефіру є вей [10, 13]

2.2.2 Обліковий Запис

Облікові записи є примітивом Ethereum, який забезпечує абстракцію над процес підпису ланцюга в блокчейні біткойн. Ця абстракція допомагає спростити концепцію права власності токену, а також розширити уявлення про те, що взагалі є токеном.

Облікові записи ідентифікуються за 160-бітовим кодом, цей 160-бітовий код і є адресою облікового запису. Внутрішньо характеризується 4 властивостями:

- Нонс - інкрементально зростаючий лічильник, який представляє кількість трансакції, надіслані з рахунку.
- Баланс - кількість ефіру, вираженого у вей, який є на записі.
- Корінь зберігання - 256-бітовий хеш кореню дерева Меркла, який кодує вміст сховища облікового запису
- Хеш код - незмінний хеш коду EVM, що відповідає обліковому запису.

Хоча всі облікові записи є структурно ідентичними, корисно розрізняти два практичні типи облікових записів, з якими можна зустрітися та взаємодіяти з блокчейном Ethereum, зовнішніми обліковими записами та внутрішніми обліковими записами:

Зовнішній обліковий запис — також називаються простими обліковим записом, або обліковим записом без смарт-контракту. Це обліковий запис, значення хеш коду якого дорівнює значенню хеш-результату функції Kessak-256 пустого рядка. Тобто це звичайний аккаунт користувача який не має в собі логіку смарт-контракту для EVM.

Внутрішній обліковий запис — також називають обліковим записом смарт-контракту. Навпаки, це обліковий хеш код якого має певний код для EVM.

Обидва типи облікових записів мають можливість надсилати та отримувати ефір, а також взаємодіяти з контрактами, які були розгорнуті в мережі Ethereum,однак, є деякі ключові відмінності між двома типами облікових записів, які варто виділити[14]:

Зовнішній обліковий запис:

- Зовнішні облікові записи керуються за допомогою криптографії публічного ключа.
- Створення зовнішнього облікового запису не коштує ефіру.
- Тільки зовнішні облікового записи можуть ініціювати транзакції
- Транзакції між зовнішніми рахунками можуть бути лише переказами ETH або інших токенів

Внутрішній обліковий запис:

- Внутрішні облікові записи керуються за допомогою коду
- Створення внутрішнього облікового запису коштує ефір, який відображає вартість зберігання коду в мережі Ethereum.
- Внутрішні облікові записи можуть виконувати код через EVM після отримання транзакцій, що забезпечує широкий діапазон мережевих функцій.
- Внутрішні облікові записи можуть надсилати транзакції лише у відповідь на отримання транзакцій.

2.2.3 Транзакції

Транзакція — це створена криптографічно підписана інструкція зовнішнім агентом і подана в мережу Ethereum. Є два види

транзакцій, які варто розрізняти, транзакції створення контракту та операція передачі повідомлення. Обидва види транзакцій мають такі спільні поля:

1. to — 160-бітова адреса, що представляє обліковий запис одержувача. Це значення опущено при операції створення контракту.
2. from — підпис, що ідентифікує відправника транзакції за адресою.
3. value — кількість ефіру, виражену у вей, яка має бути передана одержувачу.
4. nonce — значення, що дорівнює кількості транзакцій, надісланих відправником.
5. gasPrice — значення, що представляє суму вей, яку потрібно сплатити за одиницю газу.
6. gasLimit — значення, що представляє максимальну кількість газу, яка можна використати для виконання транзакції.

Транзакція створення контракту включають наступне додаткове поле:

1. init — фрагмент EVM-коду, який виконується лише один раз і потім відкидається. він повертає body, другий частину коду, яка виконується щоразу при отриманні повідомлень. Повідомлення можуть приходити при отриманні транзакції чи при внутрішньому виклику функцій.

В той же час операція передачі повідомлення має наступне додаткове поле:

1. data — масив байтів необмеженого розміру, який містить дані які повинні бути передані у повідомленні.

2.2.4 Віртуальна Машина Етеріум

Віртуальна машина Етеріум (EVM) — це середовище виконання, в якому обробляється код Ethereum. EVM обробляє низькорівневий байт-код та робить зміни у стані блокчейну. Це можуть операції: зчитування, обробки або запису даних. Специфікація EVM надає низькорівневий набір інструкцій

який і визначає доступні операції які повинна підтримувати EVM. Туди входять: коди операцій, вхідні дані, вихідні дані та інші деталі реалізації. EVM можна описати функцією зміни стану: $Y(S, T) = S'$. Заданий набір транзакцій T і початкове значення стану S та функція переходу стану Y (S, T), створить новий вихідний стан, S'. [10] Існує кілька реалізацій EVM, які були написані різними мовами програмування як Go, Python, C++.

2.2.5 Підтримка мов програмування

Існує кілька мов високого рівня, орієнтованих на EVM і можуть використовуватися для створення “smart-contracts”. Наприклад, Solidity який схожий на C++ та JavaScript, або Vyper, який більше схожий на python з його простотою. Ці мови мають вбудовані компілятори які використовуються для перетворення програмного коду в низькорівневий байт-код EVM. Solidity, наприклад, використовує компілятор solc.

2.2.6 Топологія мережі

Як і біткойн, Ethereum існує як мережа вузлів, кожен з яких підтримує різні функції, які спільно працюють для підтримки блокчейну та екосистеми блокчейну. Ці вузли можна класифікувати за функціональними можливостями, які вони підтримують.

2.2.7 Типи вузлів

Існує декілька реалізацій клієнтів протоколу Ethereum написаних на різних мовах програмування. Geth клієнт написаний на Golang, Parity клієнт написаний на Rust, pyethereum клієнт написаний на Python. Деякі з цих клієнтів підтримують запуск у різних режимах:

- Повний вузол — це повна реалізація протоколу Ethereum. Повні вузли обробляють та перевіряють усі транзакції, які були додані до блокчейну Ethereum та допомагає підтримувати надійність мережі. Щоб підтримувати цей функціональн, повний вузол повинен зберігати повну копію блокчейну. Повний вузол може розгортати смарт-контракти, майнити блоки мережі, розгортати гаманець, а також проводити маршрутизацію по мережі.
- Віддалений вузол підтримує підмножину функцій, яку підтримує повний вузол. Загалом віддалений вузол має функціонал гаманця та можливість надсилати транзакції до мережі. Інші складніші функції зазвичай вимагають взаємодії з повним вузлом, який здатний виконувати запити від імені віддаленого вузла.

2.2.8 Мережа

Як і біткойн, блокчейн Ethereum працює за допомогою алгоритму Proof-of-Work (PoW) для досягнення консенсусу в усій мережі. Алгоритм PoW використаний в Ethereum, Ethash, допомагає зміцнити довіру та надійність всієї мережі, а також захищає блокчейн і гарантує, що виконання коду EVM було оброблено, а результати отримані після виконання, відповідають очікуванням. Ethereum пропонує крипто економічне стимулювання в формі ефіру для стимуляції участі у процесі роботи алгоритму.[10]

2.2.9 Блоки

Протокол Ethereum групує колекції транзакцій у блоки. Блоки пов’язані з попередніми блоками за допомогою криптографічного хеша, який відображає

попередні стани блокчейну. При сукупній обробці ці блоки відображають нинішній стан блокчейну Ethereum.

2.2.10 Доказ виконаної роботи

Алгоритм PoW Ethereum, Ethash, — це алгоритм, який спочатку був схожий на алгоритми Hashimoto та Dagger. Основною мотивацією алгоритму Ethash було створення алгоритму PoW, який був би ASIC–стійкий. Основний механізм, який використовується для досягнення стійкості ASIC, те що алгоритм тісно пов’язанний з пам’яттю та потребує значно більше пам’яті на додаток до обчислювальних потужностей щоб правильно підрахувати рішення. Вимагаючи велику кількість операцій, пов’язаних із пам’яттю, до більшості яких алгоритм робить стає стійким до різних операцій пов’язаних з роботою з пам’яттю та кешуванням. Крім того, вимоги до пам'яті розроблені таким чином, щоб з часом зростати та змінюватися. У певному сенсі алгоритм Ethash може бути кращим описується як доказ пам’яті.[15, 16] Мережа Ethereum намагається відійти від Ethash до меньш вимогливого до електроенергії та обчислювальних потужностей алгоритму Proof-of-Stake (PoS). Команда Ethereum успішно перевала одну з тестових мереж Ropsten на новий алгоритм PoS, міграція основної мережі запланована на серпень 2022 року.

2.2.11 Газ

Щоб підтвердити, що код переданий в EVM виконано як і очікувалося, кожен повний вузол мережі повинен повторно обчислити всі транзакції та виконати будь-який код який був переданий до EVM під час виконання цих транзакцій. Не важко уявити як це може призвести до проблем у мережі, наприклад, якійсь участник мережі може запустити нескінченний цикл в мережу. Для

вирішення цієї проблеми було додану абстракцію газ який має ринкову вартість і повинен бути оплачений відправником транзакції наперед під час створення транзакції. Кожна операція, обчислена EVM, має певну ціну яку відправник повинен оплатити, ця оплата газу потім виплачується вузлу, який успішно карбує створює блок. Якщо недостатня кількість газу надається відправником транзакції, мережа видасть помилку Out-of-gas exception, після чого виконання операції зупиняється, стан мережі відновлюється а весь газ який було витрачено відходить вузлу який обробляв операцію. Якщо відправник транзакції відправив більше газу ніж потрібно йому повертається повна сума яка не була витрачена. Під час генерації транзакції в мережі Ethereum у творця транзакції є вибір: визначити, скільки вей вони готові платити за одиницю газу. І звичайно, транзакції які пропонують вищу ціну за одиницю газу швидше обробляються в мережі, оскільки вузли мотивовані отримати більшу нагороду за обробку транзакції. На рисунку 2.3 можна побачити повний список цін у газі за операцію описаних у жовтій книзі Ethereum.

Назва	Значення	Опис
Gzero	0	Нічого не сплачено за операції набору Wzero.
Gbase	2	Обсяг газу для оплати за операції набору Wbase
Gverylow	3	Обсяг газу для оплати за операції набору Wverylow
Glow	5	Обсяг газу для оплати за операції набору Wlow
Gmid	8	Обсяг газу для оплати за операції набору Wmid
Ghigh	10	Обсяг газу для оплати за операції набору Whigh
Gextcode	700	Кількість газу для оплати за операції набору Wextcode

Gbalance	400	Кількість газу для оплати операції Balance
Gsload	200	Ціна за операцію LOAD
Gjumpdest	1	Ціна за операцію JUMPDEST
Gsset	20000	Ціна за операцію SSTORE, якщо значення зберігання встановлено на відмінне від нуля
Gsreset	5000	Ціна за операцію SSTORE, коли значення нуля пам’яті залишається незмінним або встановлено на нуль.
Rsclear	15000	Надається відшкодування (додається до лічильника відшкодувань), коли значення для зберігання встановлено на нуль або значення відмінне від нуля
Rselfdestruct	24000	Повернення коштів за самознищення облікового запису
GSelfdestruct	5000	Кількість газу для оплати операції SELFDESTRUCT
Gcreate	32000	Ціна за операцію CREATE
Gcodedeposit	200	Сплачується за кожен байт в операції CREATE для запису коду в стан машини.
Gcall	700	Ціна за операцію CALL
Gcallvalue	9000	Ціна за передачу ненульового значення в операції CALL
Gcallstipend	2300	Стипендія за виклик контракту віднімається від Gcallvalue для передачі ненульової значень.
Gnewaccount	25000	Ціна за операцію CALL або SELFDESTRUCT, яка створює обліковий запис.
Gexp	10	Часткова ціна за операцію EXP
Gexpbyte	50	Часткова ціна множення на [log256(e)] операції EXP

Gmemory	3	Ціна за кожне додаткове слово при розширенні пам'яті
Gtxcreate	32000	Оплачується всіма транзакціями, що створюють контракт після переходу Homestead.
Gtxdatazero	4	Ціна за кожен нульовий байт даних або код транзакції
Gtxdatanonzero	68	Ціна за кожен ненульовий байт даних або код транзакції
Gtransaction	21000	Ціна за будь-яку транзакцію
Glog	375	Часткова ціна за операцію LOG
Glogdata	8	Ціна за кожен байт даних операції LOG
Glogtopic	375	Ціна за кожну тему операції LOG
Gsha3	30	Ціна за кожну операцію SHA3.
Gsha3word	6	Ціна за кожне слово для вхідних даних для операції SHA3
Gcopy	3	Часткова ціна за операцію *COPY, помножена на кількість скопійованих слів, округлена в більшу сторону
Gblockhash	20	Ціна за операцію BLOCKHASH

Wzero = {STOP, RETURN}

Wlow = {MUL, DIV, SDIV, MOD, SMOD, SIGNEXTEND}

Wmid = {ADDMOD, MULMOD, JUMP}

Whigh = {JUMPI}

Wextcode = {EXTCODESIZE}

Wbase = {ADDRESS, ORIGIN, CALLER, CALLVALUE, CALLDATASIZE, CODESIZE, GASPRICE, COINBASE TIMESTAMP, NUMBER, DIFFICULTY, GASLIMIT, POP, PC, MSIZE, GAS}

Wverylow = {ADD, SUB, NOT, LT, GT, SLT, SGT, EQ, ISZERO, AND, OR, XOR, BYTE, CALLDATALOAD, MLOAD, MSTORE, MSTORE8, PUSH*, DUP*, SWAP*}

Таб. 2.3 Ціни за операції EVM в газ та операції

Розділ 3. Інтернет голосування

Інтернет-голосування, яке іноді називають дистанційним електронним голосуванням, є системою, де виборці можуть віддати свій голос через інтернет.

Хоча на даний момент немає децентралізованої система голосування, інтернет-голосування вже існує. В Естонії, i-Voting доступний з 2005 року, а на виборах, 3 березня 2019 року через інтернет проголосувало 43.8% учасників.[17] Уряд країни завіряє що інтернет голосування допомагає країни зберегти країні 11000 робочих днів за кожен рік в якому присутнє голосування.[18]

Веб-сайт e-Estonia [19] пояснює, як працює їх система “Під час визначеного попереднього періода голосування, виборець входить в систему за допомогою ID-картки або Mobile-ID і віддає свій голос. Дані про особу виборця видаляють з виборчого бюлетеня до того, як він надходить до Національної виборчої комісії для підрахунку, тим самим забезпечуючи анонімність виборця”.

У 2014 році Алекс Халдерман, експерт з кібербезпеки, та його команда відвідали Естонію, щоб з’ясувати, чи США мають запровадити подібну систему голосування[20]. Серед низки проблем, Гальдерман виділив, що

“система інтернет голосування Естонії сліпо довіряє виборчим серверам і комп’ютерам виборців”[21]. З децентралізованою системою такої довіри до центрального серверу не потрібно б було мати.

У корумпованих країнах, де поширені фальсифікації [22], децентралізована система голосування вирішила б проблему питання довіри. Виборцям доведеться довіряти блокчейну, а не уряду. Однак, наприклад, у сільських та бідних регіонах Мексики [23] та Аргентини [24], купівля голосів є широко поширеною. Завдяки інтернет голосуванню голоси було б легше контролювати. Сьюзан Стоукс, американський політолог, пише: “Чим точніше машина може стежити за виборцями, тим більший потенціал для купівлі голосів.[24]”

Розділ 4. Методика

Ця робота аналізує різні виборчі системи, а також проектує і будує виборчу систему та модель автентифікації виборців. Реалізація надається як смарт-контракти які протестовані на їх надійність використовуються, а також аналізується доцільність їх використання як інструмента у процесі прийняття рішень. Безперечно, що інфраструктура та її властивості, притаманні технології блокчейн, створюють ризики та недоліки щодо голосування, однак ці властивості також надають унікальні можливості досліджувати нові підходи до моделей виборчої системи. Це дослідження має за мету вивчити переваги властивостей блокчейну у моделях виборчої системи, а також мінімізувати представлені ризики та виявлені недоліки, які випливають разом із ними.

4.1 Вимоги

Дивлячись на вимоги, запроваджені у роботі “The Future of Voting: End-to-End Verifiable Internet Voting” [25] то можна виділити 2 основні групи вимог: технічні та нефункціональні. Ці вимоги створюють основу для аналізу отриманих результатів дослідження. Серед вимог які будуть розглядатись або частково розглядатись можна виділити технічні: аутентифікацію користувача, надійність, безпеку, доступність до аудиту, функціональність, а також частково розглянуто нефункціональні вимоги: обслуговування, еволюція системи та точність.

Вимоги, визначені в “The Future of Voting: End-to-End Verifiable Internet Voting” [25] мають по праву вимогливі та складні до виконання відповідні критерії, однак, проектування системи яка б могла покрити всі критерії виходять за межі цього дослідження. Таким чином, визначено підмножину технічних і нефункціональних вимог, які вважаються досяжними та актуальним в рамках цього дослідження. Кілька з визначених вимог частково або повністю задоволені завдяки властивостям технології блокчейн.

4.1.1 Вимоги до надійності та безпеки

Вимоги надійності та безпеки вважаються в основному задовільними завдяки властивостям блокчейн Ethereum та його децентралізованій структурі. Однак, є певні можливі атаки на мережу які будуть розглянути далі.

4.1.2 Вимоги до можливості аудиту

Ethereum пропонує детальний доступ до інформації через свою прозору структуру що надає широкий спектр можливостей для детального аудиту та перевірок. Проте, через прозорість систему також не можна вважати такою яка повністю зберігає конфіденційність і тому система не забезпечує цю вимогу.

4.1.3 Функціональні вимоги

Перелік функціональних вимог які мають відношення до цього дослідження:

- Забезпечення того, щоб “виборчий бюлетень та акт голосування були зафіксовані і збережено, як очікувалося” можна вважати забезпечений в основному завдяки внутрішнім властивостям блокчейну Ethereum. Етап на якому ця властивість може зазнати невдачі це коли виборець віддає свій голос, відправляє транзакцію в мережу, проте вузол не записує транзакцію до блоку. Цю проблему ще буде висвітлено далі.
- Конфіденційність виборця можна вважати частково забезпеченою завдяки властивостям блокчейну Ethereum. Хоча технічно це можливо транслювати транзакції анонімно в мережі Ethereum , це, ймовірно, виходить за межі можливостей більшості учасників голосування і збільшує ризики не фіксації голосу. Псевдо-анонімність це поки що найвищий рівень конфіденційності який можна запровадити.

4.1.3 Нефункціональні вимоги

Дані вимоги більше відносяться до структур які будуть використовувати систему, та ці вимоги переважно виходять за рамки та контекст цього дослідження.

4.2 Технології

4.2.1 Мережа

Як вже розглядалось в роботі Ethereum пропонує середовище виконання коду через EVM. EVM відповідає за виконання набору інструкцій низького рівня, які можна використовувати для оновлення стану мережі. Існує декілька мов вищого рівня,які можна компілювати у байт-коду необхідного для EVM. У цьому дослідження буде використана мова Solidity. Для того, щоб EVM міг виконати байт-код, він повинен спочатку потрапити на блокчейн. Розгортання байт-коду в Ethereum виконується шляхом розгортання приватної мережі Ethereum, надсилання транзакції створення облікового запису(адреси), яка включає байт-код EVM. Після отримання транзакції EVM виконає створення внутрішнього облікового запису. Цей внутрішній обліковий запис зі збереженим байт-кодом і називають смарт-контрактом. Після того, як смарт-контракт починає існувати в блокчейні, байт-код, що зберігається в ньому, може бути активований шляхом створення та надсилання іншої транзакції в мережу цього разу з інструкцією виконання байт-коду який знаходиться в смарт-контракті. Окрім байт-коду в смарт-контракті ще зберігається внутрішній змінні смарт-контракту які можна використовувати для збереження інформації.

4.2.2 Мова програмування

Дві найпопулярніші мови високого рівня для розробки на EVM є Solidity та Vyper. Для цього дослідження було обрано Solidity через низку переваг[26]:

- Об’єктно-орієнтована мова високого рівня для реалізації смарт-контрактів.
- Мова фігурних дужок, яка зазнала найбільшого впливу C++.

- Статично типізована.
- Підтримує
    - Успадкування
    - Бібліотеки
    - Комплексні типи створені користувачем
- Доступні гарні інструменти для розробника
- Solidity має велику спільноту розробників

Загалом, Solidity є найпопулярнішою мовою розробки смарт-контрактів та має переваги які значно полегшують розробку розробнику такі як: комплексні типи, успадкування, бібліотеки. Vyper є пітоністичною мовою програмування ідея якою є код легкий до читання та аудиту, проте можливості які є у Solidity.

4.2.3 Фреймворк

Враховуючи складний процес, пов’язаний з компіляцією, розгортанням смарт-контрактів та комунікацією зі смарт-контрактом у мережі Ethereum. Зазвичай використовується фреймворк для спрощення розробки та керування життєвим циклом смарт-контрактів.
Hardhat — це середовище розробки для компіляції, розгортання, тестування та налагодження програмного забезпечення Ethereum. Hardhat допомагає розробникам керувати та автоматизувати повторювані завдання, які притаманні процесу створення розумних контрактів і dApps, а також легко впроваджувати більше функціональних можливостей навколо цього робочого процесу.[27] Hardhat дає комфортний досвід розробки смарт-контрактів та їх розгортання на потрібних мережах, також фреймворк підтримує виконання процесів та тестів написах на JavaScript або TypeScript.

4.2.4 Архітектура

Архітектура системи складається з двох елементів: авторизації та виборів. Кожна фаза розглядає різні компоненти загальної архітектури системи, і кожен компонент відповідає за управління різних обов’язків у архітектурі.

4.2.5 Авторизація

Компоненти авторизації призначені для забезпечення доступу до системи: підтримок списку актуальних виборців, адміністраторів та будь-яких інших ролей необхідних для виборчої системи. Загалом, компоненти авторизації відповідають за обмеження доступу до закритих викликів функцій смарт-контракту які можуть змінювати стан у контракті, наприклад, проголосувати. Конструкція має бути достатньо гнучкою, щоб задовольнити різноманітні потреби, що постійно змінюються, але достатньо послідовною, щоб вони всі підходили під загальний інтерфейс. Основні функції аутентифікації відбуваються через асиметричну криптографія, що забезпечується інфраструктурою блокчейну.

4.2.5.1 Архітектура

Основний функціонал доступний смарт-контрактам, — це виклики функцій, таким чином, моделі контролю доступу є обов’язковою для смарт-контракта в цьому дослідженні. Ми визначаємо інтерфейс Authority, який визначає функцію canCall. Будь-який інший смарт-контракт, який бажає реалізувати доступу контролю до функції, може використовувати контракт який реалізує інтерфейс Authority для авторизації доступу до цих функцій.

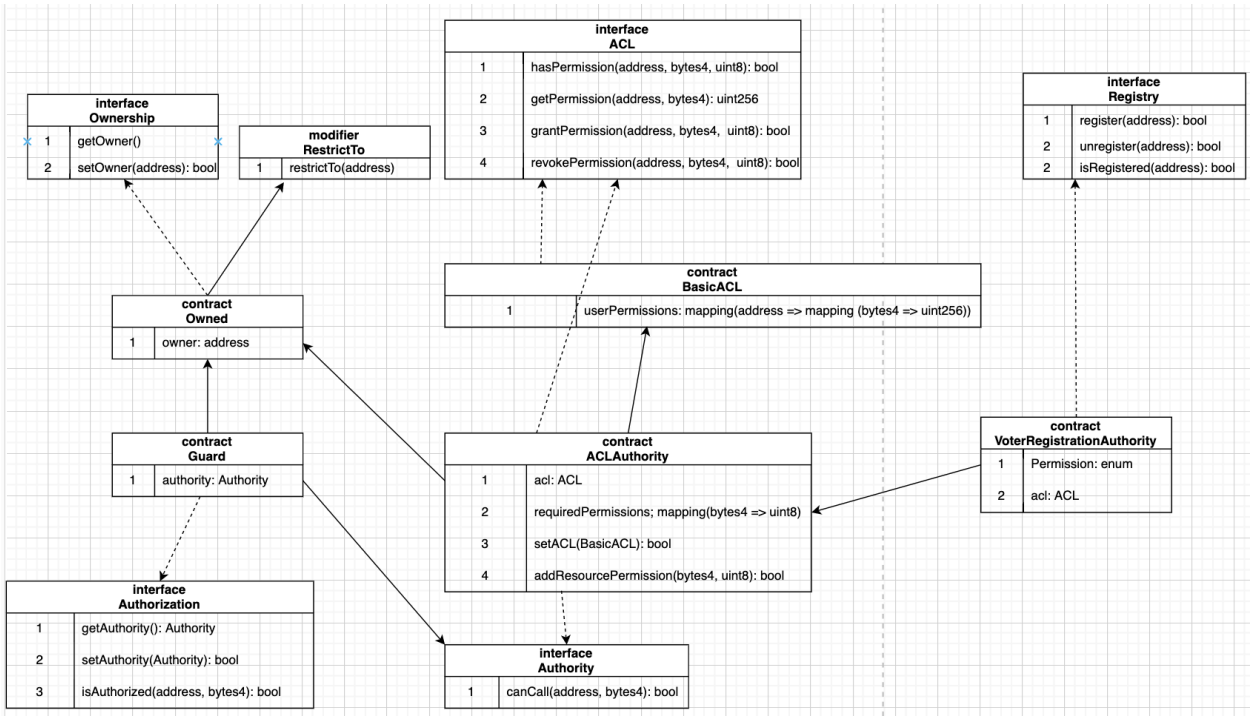

Рис. 4.1 Граф залежностей моделі авторизації

Ми визначаємо інтерфейс Authorization, який взаємодіє з інтерфейсом Authority. Цей інтерфейс визначає getAuthority, setAuthority і, що найважливіше, isAuthorized.

Контракт Guard реалізує інтерфейс Authorization та надає функції модифікатора restrictTo. Цей модифікатор можна застосувати до інших функцій та обмежити доступ до викликів функцій обліковими записами, які схвалив адміністратор.

Ця реалізація відповідає ряду принципів проектування: поділ відповідальності, принцип інверсії залежності, принцип відкритості/закритості, принцип розділу інтерфейсу, принцип заміщення та принцип єдиної відповідальності.

Контроль доступу часто починається ще до реалізації авторизації та авторизації. Більшість контрактів пропонують примітивну форму контролю доступу через Ownership. Смарт-контракти та більшість контрактних функцій є відкритими та публічними, таким чином, важливо заблокувати чутливі функції контракту від виконання. Поширеним шаблоном є збереження адреси автора контракту як власника в змінній контракту та використовувати цю адресу для обмеження доступу до закритих функцій.

Також нам потрібно забезпечити контроль доступу за межами інтерфейсів і не обмежуватись адресою єдиного облікового запису. Реалізація контролю доступу використовує список контролю доступу, який можна розширити, щоб забезпечити інший контроль доступу. За допомогою чого ми можемо надавати доступ по ролям. Також смарт-контракт може реалізувати тільки інтерфейс Authority якщо немає потреби у доступі по ролям.

За допомогою інтерфейса ACL ми можемо підтримувати до 256 унікальних дозволів у смарт-контракті.

Далі ми реалізуємо контракт BasicACL який використовуємо як основнону базу даних, який створює маппінг від адреси облікового запису до ще одного маппінгу функцій(зберігається перші 4 байти їх сигнатури) до 256-бітного цілого числа без знаку, яке представляє дозволи. Дозволи працюють за допомогою побітових операцій NOT, OR та BUT щоб модифікувати 256-бітне ціле число без знаку, що представляє дозволи для певної пари (адреса , функція).

Маючи таку структуру ми легко можемо імплементувати ACLAuthority використовуючи BasicACL. Реалізуючи інтерфейс Authority та ACL ми поєднуємо наші функції canCall та hasPermission які будуть діставати інформацію з BasicACL.

Також нам треба максимально спростити використання цієї структури для інших смарт-контрактів. Тому ми створюємо VoterRegistrationAuthority який реалізує інтерфейси Registry та Authority. Це дозволяє нам створити абстракцію з простими для виклику функціями для подальшої роботи з авторизацією користувачів.

4.2.6 Вибори

Виборчі компоненти відповідають за побудову та проведення виборчих процесів. Ці компоненти повинні підтримувати цілісність та безпеку виборів, обробки голосів, підрахунок бюлетенів та визначення переможців виборів

4.2.6.1 Архітектура

Існують важливі критерії, які варто враховувати під час аналізу доцільність впровадження виборчих систем через інтернет, а саме, критерії підрахунку бюлетенів: сума, поліноміальний час і розв’язність. Ці виборчі критерії є суттєві, оскільки вони впливають на витрати на газ, пов’язані з роботою системи яка працює на блокчейні. Звичайну систему відносної більшості можна підрахувати використовуючи лінійну кількість простору відносно числа доступних варіантів без втрати інформації необхідної для завершення процесу підрахунку. Якщо враховувати що виборець може віддати один голос за кандидата, то складність підрахунку голосів буде $O(n^2)$ Під час впровадження, розгортання та експлуатації необхідно враховувати витрати на газ у EVM. Якщо витрати газу занадто великі то використання такої системи може бути занадто догорим. . Найдорожчими операціями, з немалою перевагою, є операціями, що включають зберігання. Тому важливо мінімізувати використання пам’яті системи.

Щоб чітко розділяти стадії голосування було виділено такі стадії: налаштування, заморожування, голосування та підрахунок.Ці фази можна змоделювати як скінченний автомат, де виклики різноманітних функції запускають переходи до різних етапів. Можна використовувати модифікатори щоб обмежити доступ до функцій які можна викликати лише під час відповідного стану. Більшість переходів із стану в стан можна спроектувати як часові переходи, які відбуватися ліниво, перевірка, щоб підтвердити, що настав час переходу з одного стану в інший виникає під час виклику функції. Обґрунтування такого підходу обумовлено той факт, що Ethereum не пропонує механізму запуску коду зсередини самого блокчейну, тому всі виклики функцій, навіть якщо вони викликані з іншої функції або навіть функції іншого контракту, зрештою, почалась з транзакції і створена зовнішнім обліковим записом.

Стадія налаштування як випливає з назви, існує щоб надати адміністраторам можливість налаштувати вибори: варіанти вибору, час початку/закінчення виборів і т.д. Після завершення налаштування адміністратори можуть заморозити контракт, запобігаючи його подальшим змінам.

Стадія заморожування, може бути просто імплементована булевим значенням. Булеве значення треба перевіряти перед виконанням кожної адміністративної функції та припиняти виконання коду, якщо значення істинне. Це зроблено для потенційного запобігання маніпуляцій від адміністратора після початку виборів.

Стадія голосування. Для цієї стадії ми повинні визначити функцію голосування. Для гнучкості системи, ми можемо передавати дані як дві розрідженні матриці, де в першій буде передаватися індекс вибору, а у другій значення вибору. Це можна використати для незначного зменшення розміру транзакцій.

Фаза підрахунку голосів – це завершальна фаза виборчого процесу, яка триває місце після закінчення фази голосування. У цьому дослідженні було імплементовано систему відносної більшості, проте функціонал підрахунку голосів буде залежати від типу виборів. На цій стадії можна запустити тільки функцію підрахунку голосів. Функція підрахунку, у цьому випадку, перевірить кількість голосів за кожного кандидата і визначить переможцем вибір, який отримав найбільшу підтримку.

Розділ 5. Результати

Тестування контрактів Ethereum можна по різному але два основних параметри які потрібно обрати для тестування це: агент тестування та мережа тестування.

Агент тестування:

- Юніт тест від контракту до контракту - використовується ще один смарт-контракт для виконання процессу тестування. Ці тести забезпечують впевненість у тому, що міжконтрактний зв’язок функціонуватиме належним чином.
- Юніт тест від клієнта до контракту - використовують зовнішні облікові записи для проведення тестування. Ці тести дають впевненість, що зовнішні облікові записи можуть виконувати код контракту належним чином

Мережа тестування:

- Hardhat network, ganache - локальні тестові ефіріум клієнти які симулюють реальний вузол. Безкоштовний та швидкий варіант, не може повністю симулювати повноцінний вузол.
- Локальний вузол Ethereum - Це безкоштовно та точно симулює вузол Ethereum, але не має складності реальної мережі.

- Тестнети - тестові мережі розроблені для тестування нового функціоналу Ethereum та розгортання смарт-контрактів. Тестнети забезпечують мережі, наближені до реальності, водночас забезпечуючи безкоштовні засоби отримання коштів для тестування.
- Найточнішим середовищем для тестування є реальна мережа Ethereum, проте це не безкоштовним, а можливо і доволі дорогим засобом тестування.

Тести у цьому дослідженні проводились за допомогою hardhat network яку можна запустити за допомогою команди `npx hardhat node`. Тести були написані за допомогою JavaScript, зв'язок з контрактами відбувався за допомогою фреймворка hardhat.

5.1 Вибори

Кілька виборчих контрактів було реалізовано з декількома змінами. Щоб оцінити вартість виконання та необхідного зберігання було змодельовано кілька наборів виборів використання різних конфігурацій виконання договору, наявності вибору, голосування структура та відбір виборців. Загальний процес вимагає створення мінімальної реалізації виборчої системи та модифікації за потреби. Виборчи системи мали тести, написані для підтвердження базової функціональності, після чого відбувається симуляція процесу виборів. Кожна симуляція використовувала сотню облікових записів, створених для нашої тестової мережі за допомогою hardhat. Кожен обліковий запис зробив випадкове вибір у голосування за допомогою генерування псевдовипадкових чисел.

Контракт	Функція	Виклик	Користувачів	Витрачено Газу	Середнє значення газу
OneVoteSmp	vote	1	100	8007700	80077
MultiVoteSmp	vote	1	100	7447632	74476
MultiVoteSmp	vote	2	100	3208496	32084
OneVoteSmp	tally	1	1	0	0
MultiVoteSmp	tally	1	1	917790	917790
OneVoteSmp	computeResult	1	1	360516	360516
MultiVoteSmp	computeResult	1	1	47438	47438

Таб. 5.1 Витрати газу у контракті на функцію

Таблиця 5.1 показує результати симуляції роботи контракту, відображаючи споживання газу за функцію виборчого контракту. Ці значення були отримані шляхом вимірювання загального споживання газу, що спостерігалося протягом кількох прогонів симуляції та обчислення середньої вартість однієї контрактної функції. Симуляції були налаштовані з використанням 100 виборців кожен пробіг, з 10 доступними варіантами для вибору на кожних виборах. Випадковий процес вибору використовувався для проведення голосування. Процесс підрахунку голосів та підрахунку результату виборів перевірялись незалежно і порівнювались з результатами контракту. Смарт-контракт OneVoteSmp імплементує виборчу систему відносної більшості в якій виборці мають право віддати один голос. Ця реалізація використовує переваги суми, щоб уникнути зберігання бюлетеня кожного окремого виборця і замість цього підтримує постійний підрахунок бюлетенів.

Просторова складність тому є постійна $O(1)$ щодо кількості виборців та лінійною $O(n)$ відносно кількості кандидатів. Реалізація продемонструвала дуже послідовне споживання газу під час викликів функцій у симуляції.
Смарт-контракт MultiVoteSmp імплементує так само систему відносної більшості, проте виборці мають право віддати більше одного голосу, проте виборець так само може віддати 1 голос. Це означає що будь-який бюлетень для голосування, який був поданий і вже був частково підраховано потрібен мати функцію відміни голосу. Щоб відмінити попередній голос і записати нову інформацію нам потрібно зберігати інформацію про обрані виборцем варіанти. Тому останній бюлетень виборця повинен завжди зберігатись у системі щоб підтримувати операцію відміни голосу якщо виборець хоче проголосувати декілька разів.
Можливість відмінити голос ставить під сумнів використання простої суми для підрахунку та збереження результатів як було використано в попередньому контракті. Постає питання використання жадібного оцінювання або лінивого оцінювання.
У системі з одним голосом можна побачити перевагу та недолік жадібного оцінювання. Виборець повністю бере на себе оплату не тільки за відданий голос, але й за підрахунок голосів у системі, проте ми бачимо що підрахунок голосів викликаний адміністратором після процесу голосування не потребував жодних витрат на газ, оскільки операції зчитування не потребують газу.
Більш традиційний підхід до таблиці результатів
на виборах полягає в тому, щоб відкласти процедуру підрахунку голосів до кінця періоду голосування та
змусити адміністратора виборів нести витрати та відповідальність за підрахунок голосів

Більш традиційний підхід до підрахунку результатів на виборах полягає в тому, щоб відкласти процедуру підрахунку голосів до кінця періоду голосування та змусити адміністратора виборів нести витрати та відповідальність за підрахунок голосів. Цей метод і був використаний під час голосування з кількома голосами, і за результатом таблиці ми можемо побачити що виборці витратили значно менше на надсилання голосів, проте операція підрахунку голосів була значно дороще ніж у першому варіанті. Можна побачити також один цікавий результат у таблиці 5.1, другий виклик функції голосування у варіанті коли виборець може віддати кілька голосів є значно дешевшим у витрати газу на відміну від першого. Висока вартість у газі першого виклику відображає високу плату за простір для зберігання в блокчейні. Другий же виклик не потребує додаткового простору для збереження інформації, а тільки перезаписує перший виклик, тим самим витрачаючи меншу кількість газу.

5.2 Авторизація

Існує низка контрактів пов’язаних з авторизацією які потребують перевірки, для перевірки контрактів ми можемо концентруватися на написанні тестів для абстракції VoterRegistrationAuthority, і тим самим можемо перевірити що функціонал інших смарт-контрактів таких як: Guard, ACLAuthority, BasicACL, теж виконується.

Тест для VoterRegistrationAuthority:

1. Створення облікових записів для голосування
2. Розгортання смарт-контракту
3. Реєстрування облікових записів
4. Перевірка чи isRegistered працює для зареєстрованих облікових записів.

5. Перевірка чи isRegistered не працює для інших облікових записів.
6. Видалити з реєстру низку зареєстрованих облікових записів.
7. Перевірка чи isRegistered працює для зареєстрованих облікових записів.
8. Перевірка чи isRegistered не працює для видалених з реєстру облікових записів.

Ціль цього дослідження була присвячена побудові архітектурі смарт-контрактів яка б була гнучкою та модульною та могла б використовуватись у багатьох випадках систем авторизації. Система і правда підтримує реалізацію багатьох рівнів доступу та класів користувачів, а також надає можливість лімітування доступу до функціоналу на потрібних рівнях доступу. Проте, система вийшла доволі громіздка, та доволі складна для розуміння, через велику кількість зв’язків між смарт-контрактами, та потребує ретельної документації для роботи з системою.

Альтернативним варіантом могла бути імплементація функціоналу авторизації в одному смарт-контракті що значно б полегшило роботу з ним, проте гнучкість такої системи є сумнівною і такі рішення скоріше можна вважати розробленим під конкретні потреби.

Система завдячує своїй надійності блокчейну Ethereum яка за свій час не мала проблем з даунтаймом та відома за свою децентралізацію. Не дивлячись на те що атаки на мережу можливі, вірогідність цього не така значна, як вірогідність знайдення проблем у смарт-контрактах системи виборів, тому система повинна пройти професійний аудит перед своїм виходом у реальний світ. Найбільшою проблемою з якою зараз стикаються користувачі Ethereum це дороговизна транзакцій у системі, що звісно буде тільки збільшуватись під час активності виборців у системі. Блокчейн Ethereum активно працює над проблемою масштабування, яку хоче вирішити за допомогою переходу на більш економний алгоритм консенсусу, а також додавання шардінгу в систему

мережі, яка може розділяти транзакції та зменшити напруження на мережу. Використання альтернативних рішень на EVM, теж можливо, багато систем пропонують рішення зі значно дешевшими транзакціями ніж Ethereum, проте мають меншу децентралізацію та репутацію надійності.
Також повна конфіденційність користувачів, на даний момент є неможливою, тільки псевдо-конфіденційність використовуючи адресу облікового запису у системі, як і з питанням масштабування, можливі рішення анонімних транзакції є у розробці загалом для екосистеми EVM, проте ще не існує достатньо надійних рішень для побудови системи прийняття важливих рішень.

5.3 Результати тестів

Це дослідження вивчило та розробило гнучку систему авторизації виборців. Однак, як зазначалося раніше, комунікацією між контрактами важко керувати. В додаток до цього, проблема масштабування мережі Ethereum може призвести до нерентабельності проведення масштабних виборів у системі. Нажаль, створення повноцінної системи голосування на мережі Ethereum на даний момент не є реальним, і потребують пошуку альтернативної мережі або вирішення нинішніх проблем проблем командою Ethereum. Звичайно, організація, яка проводить вибори, може запустити особисту мережу блокчейн, і це б вирішило питання рентабельності проведення виборів, проте це не вирішує ціль дослідження, а тобто повне перенесення довіри на блокчейн з організації.

5.4 Майбутня робота та напрямки вдосконалення

- Якщо проблеми масштабування у мережі Ethereum не будуть вирішені то існує декілька варіантів дій:
    - Пошук альтернативної мережі яка б забезпечувала дешеві транзакції при тому не втрачали свої якості децентралізації
    - Проектування та запуск приватного блокчейну конкретно запроектованого під вирішення питань виборчої системи, проблема з приватними блокчейнами це, звичайно, проблема централізації, тому це також потребуватиме вирішення легальних питань, хто може брати участь у запуску вузлів у системи та на яких умовах буде визначено що система децентралізована. Ці питання є важливими для вирішення, оскільки інакше немає сенсу імплементація рішення у блокчейні.
- Побудова інтерфейсу користувача, побудова інтерфейсу має бути модульною щоб адаптуватися під модифікацію системи та додавання додаткових виборчих систем.
- Додавання різних виборчих систем, у цьому дослідженні було розроблено виборчу систему відносної більшості, проте широко використовувана пропорційна виборча система, була б чудовим додатком.
- Об’єднання системи у бібліотеку та написання документації використання системи, для вирішення питання роботи з системою, та для контролю версій системи.

Розділ 6. Висновок

Зараз, все частіше постають питання які потребують системи честного голосування, яке б можна було провести дистанційно, це пов’язанно як і з безпекою виборців так і з неможливістю деяких виборців проголосувати фізично. Концепція децентралізованих організацій, що працюють демократично та існують у мережі, пропонує а провокаційний і привабливий альтернативний підхід, який може мати потенціал підтримувати більш демократичні соціальні структури та моделі управління.

Було розглянути різноманітні виборчих систем та ознайомлення з блокчейн рішеннями які надають нам високий ступінь доступу до аудиту та впевненості у відсутності маніпуляцій у системі. Ознайомлення з проблемами побудови систем Інтернет голосування.

Було побудовано низку смарт-контрактів які б вирішували питання авторизації виборців у блокчейні та надання їм певних рівнів доступу, при цьому захист системи від маніпуляції під час виборів. Не дивлячись на те що результати тестування показують що, загалом, створення повноцінної системи голосування можливе, мережа Ethereum використана в дослідженні ще потребує вдосконалення та вирішенні їх актуальних проблем, таких як анонімність транзакцій та масштабування.

Випливаючи з проблем масштабування, вимоги до обчислення та зберігання функціонування виборчої системи є надто високими, щоб зробити рентабельної для використання в більшості сценаріїв. За найщедрішими підрахунками одна операція голосування буде вартувати близько 5 доларів, це при врахування що операції проводились на мережі яка виконувала операції тільки пов’язані з голосуванням, на повноцінній мережі, ціна за один голос буде зростати значно швидше, оскілька мережа завантажена ще іншими

транзакціями не пов’язаними з процессом голосуванням. Ціна за виконання функції підрахунку голосів буде в районі 1000 доларів і це тільки за 100 користувачів. Все це робить систему неймовірно дорогою для використання, при тому не враховуючи всі додаткові витрати на адміністративну роботу та інші процеси пов’язані з виборами.

Проте, з вирішенням проблем масштабування у мережі використання для системи виборів, використання таких систем можливе у майбутньому і значно покращить демократичні цінності організацій та урядів які будуть використовувати такі системи, а також вирішать проблеми централізованих систем Інтернет голосування.

Список літератури

1. Madise Ü. Martens T. E-voting in Estonia 2005. The first Practice of Country-wide binding Internet Voting in the World. 2006.
2. Gerlach J., Grasser U. Three Case Studies from Switzerland: E-voting. Berkman Center Research Publication. 2009.
3. Stenerud. I. S. G., Bull C. When reality comes knocking: Norwegian experiences with verifiable electronic voting. 5th International Conference on Electronic Voting 2012 (EVOTE2012), 2012
4. Springall D., Finkenauer T., Durumeric Z., Kitcat J., Hursti H., MacAlpine M., Halderman A. Security Analysis of the Estonian Internet Voting System. CCS '14: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, 2014
5. Osgood R. The Future of Democracy: Blockchain Voting. COMP116: Information Security 2016
6. Andrew Reynolds. 2005. Electoral system design: The New International IDEA Handbook. International Institute for Democracy and Electoral Assistance, Stockholm, Sweden
7. Satoshi Nakamoto. 2009. Bitcoin: a peer-to-peer electronic cash system.
8. A.M. Antonopoulos. 2014. Mastering Bitcoin: Unlocking Digital Cryptocurrencies.
9. Adam Back. 2002. Hashcash - A Denial of Service Counter-Measure. Technical report.
10. Gavin Wood et al. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. Technical report 2014, 1–32.
11. Vitalik Buterin et al. 2014. A Next-Generation Smart Contract and Decentralized Application Platform. Technical report 37

12. Ethereum Foundation. 2018. Ethereum. ethereum.org. https://ethereum.org.
13. A.M. Antonopoulos and G.W.P. D. 2018. Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media
14. Ethereum Foundation. 2021. Ethereum accounts. ethereum.org. https://ethereum. org/en/developers/docs/accounts/.
15. Vitalik Buterin. 2013. Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alternative. Technical report. http://www.hashcash.org/papers/ dagger.html.
16. Vitalik Buterin. 2014. Dagger Hashimoto. eth.wiki. https://eth.wiki/concepts/ dagger-hashimoto
17. Valimised, "Statistics about internet voting in estonia," https://www.valimised.ee/en/archive/statistics-about-internet-voting-estonia, 2019
18. B. Perrigo. (2019) What the U.S. Can Learn About Electronic Voting From This Tiny Eastern European Nation.
19. Valimised. I-Voting. https: //e-estonia.com/solutions/e-governance/i-voting/
20. D. Springall, T. Finkenauer, Z. Durumeric, J. Kitcat, H. Hursti, M. MacAlpine, and J. A. Halderman, "Security analysis of the estonian internet voting system,"
21. T. Finkenauer and D. Springall, "Estoniavoting.org press-release," https://estoniaevoting.org/press-release/, 2014
22. S. Nichter, "Vote buying or turnout buying? machine politics and the secret ballot," American political science review
23. C. Vilalta, "Vote-buying crime reports in mexico: magnitude and correlates," Crime, law and social change
24. S. C. Stokes, "Perverse accountability: A formal model of machine politics with evidence from argentina," American political science review

25. Susan Dzieduszycka-Suinat, Judy Murray, Joseph R Kiniry, Daniel M Zimmerman, Daniel Wagner, Philip Robinson, Adam Foltzer, and Shpatar Morina. 2015. The Future of Voting: End-to-End Verifiable Internet Voting. Specification and Feasibility Assessment Study. U.S. Vote Foundation
26. https://ethereum.org/en/developers/docs/smart-contracts/languages/
27. https://hardhat.org/getting-started
28. https://docs.soliditylang.org/
29. https://geth.ethereum.org/

Додаток 1. Код смарт-контракту VoterRegistrationAuthority

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;
import './Owned.sol';
import './Registry.sol';
import './Authority.sol';
import './AccessControlList.sol';
import './ACLAuthority.sol';

contract VoterRegistrationAuthority is Owned, Registry, Authority {
    enum Permissions {
        Vote
    }

    ACLAuthority acl;

    mapping(bytes32 => bytes4) resourceSignatures;
```

```
constructor() {
    acl = new ACLAuthority(true);
    resourceSignatures['vote'] = bytes4(keccak256('vote()'));
    acl.setRequiredResourcePermission(
        resourceSignatures['vote'],
        uint8(Permissions.Vote)
    );
}

function register(address _voter)
    public
    override
    restrictTo(owner)
    returns (bool _success)
{
    return
        acl.grantPermissions(
            _voter,
            resourceSignatures['vote'],
            uint8(Permissions.Vote)
        );
}

function unregister(address _voter)
    public
    override
    restrictTo(owner)
```

```
    returns (bool _success)
{
    return
        acl.revokePermissions(
            _voter,
            resourceSignatures['vote'],
            uint8(Permissions.Vote)
        );
}

function isRegistered(address _voter)
    public
    view
    override
    returns (bool _isRegistered)
{
    return
        acl.hasPermission(
            _voter,
            resourceSignatures['vote'],
            uint8(Permissions.Vote)
        );
}

function canCall(address _subject, bytes4 _resource)
    public
    view
```

```
        override
        returns (bool _canCall)
    {
        return acl.canCall(_subject, _resource);
    }
}
```

Додаток 2. Код інтерфейсу Registry

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

interface Registry {
    function register(address _subject) external returns (bool _success);

    function unregister(address _subject) external returns (bool _success);

    function isRegistered(address _subject)
        external
        view
        returns (bool _isRegistered);
}
```

Додаток 3. Код смарт-контракту ACLAuthority

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

import './Owned.sol';
```

```
import './Authority.sol';
import './ACL.sol';
import './BasicACL.sol';

contract ACLAuthority is Owned, Authority, AccessControlList {
    AccessControlList acl;

    mapping(bytes4 => uint8) requiredResourcePermission;

    constructor(bool _createACL) {
        if (_createACL) acl = new BasicACL();
    }

    function hasPermission(
        address _subject,
        bytes4 _resource,
        uint8 _permission
    ) public view override returns (bool _hasPermission) {
        return acl.hasPermission(_subject, _resource, _permission);
    }

    function getPermissions(address _subject, bytes4 _resource)
        public
        view
        override
        returns (uint256 _permissions)
    {
```

```
    return acl.getPermissions(_subject, _resource);
}

function grantPermissions(
    address _subject,
    bytes4 _resource,
    uint8 _permission
) public override restrictTo(owner) returns (bool _success) {
    return acl.grantPermissions(_subject, _resource, _permission);
}

function revokePermissions(
    address _subject,
    bytes4 _resource,
    uint8 _permission
) public override restrictTo(owner) returns (bool _success) {
    return acl.revokePermissions(_subject, _resource, _permission);
}

function setPermissions(
    address _subject,
    bytes4 _resource,
    uint256 _permissions
) public override restrictTo(owner) returns (bool _success) {
    return acl.setPermissions(_subject, _resource, _permissions);
}
```

```
function setACL(AccessControlList _acl)
    public
    restrictTo(owner)
    returns (bool _success)
{
    assert(owner == address(this));
    acl = _acl;
    return true;
}

function setRequiredResourcePermission(bytes4 _resource, uint8 _permission)
    public
    restrictTo(owner)
    returns (bool _success)
{
    requiredResourcePermission[_resource] = _permission;
    return true;
}

function canCall(address _subject, bytes4 _resource)
    public
    view
    override
    returns (bool _canCall)
{
    return
        hasPermission(
```

```
            _subject,
            _resource,
            requiredResourcePermission[msg.sig]
        );
    }
}
```

Додаток 4. Код смарт-контракту BasicACL

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

import './ACL.sol';
import './Owned.sol';

contract BasicACL is Owned, AccessControlList {
    mapping(address => mapping(bytes4 => uint256)) subjectResourcePermissions;

    function hasPermission(
        address _subject,
        bytes4 _resource,
        uint8 _permission
    ) public view override restrictTo(owner) returns (bool _hasPermission) {
        uint256 result = subjectResourcePermissions[_subject][_resource] &
            (uint256(1) << _permission);
        return (result > 0);
    }
```

```
function getPermissions(address _subject, bytes4 _resource)
    public
    view
    override
    restrictTo(owner)
    returns (uint256 _permissions)
{
    return subjectResourcePermissions[_subject][_resource];
}

function grantPermissions(
    address _subject,
    bytes4 _resource,
    uint8 _permission
) public override restrictTo(owner) returns (bool _success) {
    subjectResourcePermissions[_subject][_resource] |=
        uint256(1) <<
        _permission;
    return true;
}

function revokePermissions(
    address _subject,
    bytes4 _resource,
    uint8 _permission
) public override restrictTo(owner) returns (bool _success) {
    subjectResourcePermissions[_subject][_resource] &= ~(uint256(1) <<
```

```
            _permission);
        return true;
    }

    function setPermissions(
        address _subject,
        bytes4 _resource,
        uint256 _permissions
    ) public override restrictTo(owner) returns (bool _success) {
        subjectResourcePermissions[_subject][_resource] = _permissions;
        return true;
    }
}
```

Додаток 5. Код інтерфейсу ACL

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

interface ACL {
    function hasPermission(
        address _subject,
        bytes4 _resource,
        uint8 _permission
    ) external view returns (bool _hasPermission);

    function getPermissions(address _subject, bytes4 _resource)
        external
```

```
        view
        returns (uint256 _permissions);

    function setPermissions(
        address _subject,
        bytes4 _resource,
        uint256 _permissions
    ) external returns (bool _success);

    function grantPermissions(
        address _subject,
        bytes4 _resource,
        uint8 _permission
    ) external returns (bool _success);

    function revokePermissions(
        address _subject,
        bytes4 _resource,
        uint8 _permission
    ) external returns (bool _success);
}
```

Додаток 6. Код інтерфейсу Authority

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

interface Authority {
    function canCall(address _subject, bytes4 _resource)
        external
        view
        returns (bool _canCall);

}
```

Додаток 7. Код смарт-контракту Owned

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;
import './RestrictTo.sol';
import './Ownership.sol';

contract Owned is RestrictTo, Ownership {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    function getOwner() external view override returns (address _owner) {
```

```
        return owner;
    }

    function setOwner(address _owner)
        external
        override
        restrictTo(owner)
        returns (bool _success)
    {
        owner = _owner;
        return true;
    }
}
```

Додаток 8. Код смарт-контракту Guard

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;
import './Authorization.sol';
import './Authority.sol';
import './Owned.sol';

contract Guard is Owned, Authorization {
    Authority public authority;

    modifier auth() {
        assert(isAuthorized(msg.sender, msg.sig));
        _;
```

```
}

modifier authorized(bytes4 _resource) {
    assert(isAuthorized(msg.sender, _resource));
    _;
}

function getAuthority() public view override returns (address _authority) {
    return address(authority);
}

function setAuthority(address _authority)
    public
    override
    auth
    returns (bool _success)
{
    authority = Authority(_authority);
    return true;
}

function isAuthorized(address _subject, bytes4 _resource)
    public
    override
    view
    returns (bool _isAuthorized)
{
```

```
    if(_subject==address(this)) return true;
    if(authority==Authority(address(0))) return false;
    if(_subject== owner) return true;
    return authority.canCall(_subject, _resource);
  }
}
```

Додаток 9. Код інтерфейсу Ownership

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

interface Ownership {
  function getOwner() external view returns (address _owner);
  function setOwner(address _owner) external returns (bool _success);

}
```

Додаток 10. Код інтерфейсу Authorization

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

interface Authorization {
   function getAuthority() external view returns(address _authority);
   function setAuthority(address _subject) external returns(bool _success);
```

```
   function isAuthorized(address _subject, bytes4 resource) external returns(bool
_isAuthorized);
 }
```

Додаток 11. Код модифікатору RestrictTo

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.4;

contract RestrictTo {
  modifier restrictTo(address _subject) {
    require(msg.sender == _subject, "You are restricted to do that operation");
    _;
  }
}
```