МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ДІЛАЙ Сергій Ярославович

Алгоритми доказу з нульовим знанням для забезпечення конфіденційності даних / Zero- Knowledge Proof Algorithms for Data Confidentiality

спеціальність: 125 – Кібербезпека

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -22
С.Я. Ділай

Науковий керівник
д.т.н., професор В.В.Яцків

Кваліфікаційну роботу
допущено до захисту:

«____» _____ ___2023 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2023

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми доказу з нульовим знанням для забезпечення конфіденційності даних» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека» освітньо-професійної програми «Кібербезпека» написана обсягом 82 сторінки і містить 9 ілюстрації, 3 таблиці, 2 додатки та 26 джерел за переліком посилань.

Метою кваліфікаційної роботи є розробки алгоритмів та схеми доказу з нульовим знанням для захисту конфіденційності даних.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: методи забезпечення цілісності та конфіденційності даних, методи доказу з нульовим знанням, методи поліноміальних зобов'язань.

Результати дослідження: розроблено алгоритм та схему доказу з нульовим знанням на основі хеш-значень, яка підвищує надійність перевірки даних без розкриття жодної додаткової інформації.

Результати роботи можуть бути застосовані при створенні системи автентифікації користувачів на основі доказу з нульовим розголошенням.

Ключові слова: КОНФІДЕНЦІЙНІСТЬ ДАНИХ, ДОКАЗ З НУЛЬОВИМ ЗНАННЯМ, ПРУВЕР, ВЕРИФІКАТОР, CIRCOM.

ABSTRACT

Qualification work on "Zero- Knowledge Proof Algorithms for Data Confidentiality" for the degree of "Master" in the specialty 125 "Cybersecurity" educational and professional program "Cybersecurity" is written in 82 pages and contains 9 illustrations, 3 table, 2 appendices and 26 source according to the list of links.

The purpose of the qualification work is the development of algorithms and proof schemes with zero knowledge for data confidentiality protection.

Research methods: To solve the set tasks in this qualification work, the following methods were used: methods ensuring data integrity and confidentiality, zero knowledge proof methods, and polynomial commitment methods.

Research results: An algorithm and proof scheme with zero knowledge based on hash values have been developed, which enhances the reliability of data verification without disclosing any additional information. The results of the work can be applied in the creation of a user authentication system based on zero-knowledge proof.

Keywords: DATA CONFIDENTIALITY, ZERO-KNOWLEDGE PROOF, PROVER, VERIFIER, CIRCOM.

ЗМІСТ

ВСТУП

Актуальність роботи. У світі швидкого розвитку технологій та зростаючої цифрової взаємодії, захист приватності та забезпечення безпеки інформації стають ключовими аспектами сучасного цифрового життя. Однак із загостренням цих викликів з'являються іноваційні підходи, серед яких важливу роль відіграє концепція доказу із нульовим розголошенням. Даний криптографічний підхід, заснований на принципі доказу із нульовим розголошенням, відкриває нові можливості для забезпечення конфіденційності та довіри в цифровому середовищі [1].

Zero-Knowledge Proof (ZKP) – це криптографічний принцип, який дозволяє довести правильність певних висловлювань чи фактів, не розкриваючи при цьому жодної конкретної інформації, що підтверджує це висловлювання. У світі, де обмін конфіденційною інформацією стає все складнішим та частіше стикається з ризиками порушення, ZKP стає важливим інструментом для забезпечення безпеки та захисту особистої інформації [2].

Докази з нульовим знанням також можна розглядати як форму технології стиснення знань, оскільки вони дозволяють перевіряючому переконати верифікатора в істинності твердження, не передаючи всього свідчення. Це призводить до меншого розміру перевірки та зменшення накладних витрат на зв’язок, що робить докази з нульовим знанням придатними для додатків з обмеженнями пропускної здатності або там, де конфіденційність має першорядне значення.

Принципи ZKP дозволяють доводити достовірність без розголошення конкретних даних, що робить його ефективним інструментом для розв’язання сучасних викликів у сфері кібербезпеки, фінансів, медицини та багатьох інших галузях.

Мета і завдання дослідження. Метою роботи є розробки алгоритмів та схеми доказу з нульовим знанням для захисту конфіденційності даних.

Досягнення визначеної мети передбачає вирішення таких завдань:

– визначити основні властивості доказу з нульовим знанням;

– провести аналіз методів доказу з нульовим знанням;

– провести порівняння схем поліноміальних зобов’язань;

– розробити алгоритм побудови SNARK;

– розробити доказ з нульовим знанням на основі хеш-значення.

Об’єкт дослідження – процеси доказу з нульовим знанням для захисту конфіденційності даних.

Предмет дослідження – алгоритми та схеми доказу з нульовим знанням.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: методи забезпечення цілісності та конфіденційності даних, методи доказу з нульовим знанням, методи поліноміальних зобов’язань.

Наукова новизна одержаних результатів. Розроблено алгоритм та схему доказу з нульовим знанням на основі хеш-значень, яка підвищує надійність перевірки даних без розкриття жодної додаткової інформації.

Практичне значення отриманих результатів. Розроблено програму на мові circom та класу доказів з нульовим розголошенням SNARK для доведення певної дати без її розголошення.

Публікації та апробація КР.

1. Басістий В.П., Ділай С.Я., Помогаєв С.О. Алгоритми доказу з нульовим знанням. Матеріали науково-практична конференція молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ – 2023), Тернопіль, 2023. – С.14-16.

2. Ділай С.Я., Кондратюк В.М., Помогаєв С.О. Використання доказів із нульовим розголошенням. Матеріали науково-практичного симпозіуму «Захист інформації», Тернопіль, 2023. – С. 48-49.

1 АНАЛІЗ МЕТОДІВ ДОКАЗУ З НУЛЬОВИМ ЗНАННЯМ

1.1 Основи доказів із нульовим знанням

Доказ нульового знання вперше був представлений у 1985 році С. Голдвассером, С. Мікалі та К. Ракоффом. З появою ZK-Rollups у 2022 році він став ключовим інструментом для масштабованості Ethereum. ZKP математично вирішує задачу доведення правильності тверджень, зберігаючи знання прихованими. Його можна використовувати в різних сценаріях, які вимагають конфіденційності та безпеки, наприклад самоідентифікації та автентифікації прав, без розкриття приватної інформації [2].

Він підпадає під категорію захисту конфіденційності, і в поєднанні з блокчейном в основному використовується для приватних транзакцій (наприклад, zcash і tornado.cash). Однак із появою web3 ZKP став одним із наріжних каменів технології завдяки своїм ефективним можливостям стиснення та перевірки в децентралізованій перевірці даних, багатоланцюжкових транзакціях взаємодії та транзакціях рівня 2.

ZKP є значним прогресом у галузі криптографії та обчислень із збереженням конфіденційності, і деякі люди порівнюють ZKP із створенням нової форми обчислень у тому сенсі, що вони впроваджують нові способи представлення, маніпулювання та міркування щодо інформації. ZKP розширюють спектр проблем, які можна вирішити, і методи, за допомогою яких можна підійти до них, запроваджуючи нові математичні абстракції та інструменти для міркування про інформацію та маніпулювання нею.

Підтвердження з нульовим знанням дозволяють використовувати більш виразні та потужні протоколи збереження конфіденційності, оскільки основна увага зосереджена на взаємозв’язках і перетвореннях між різними частинами інформації, а не на самих конкретних значеннях. Це означає, що ZKP пропонують унікальну обчислювальну парадигму, яка зосереджена на обчисленнях із збереженням конфіденційності. Це дозволяє перевіряти обчислення без розкриття базових даних або специфіки самого обчислення.

Спочатку докази з нульовим знанням були просто математичними концепціями без реальних застосувань чи методології. Однак розвиток технології блокчейн і розвиток методології Грота забезпечили практичні застосування та основу для їх використання.

Визначення: доказ з нульовим знанням – це метод, за допомогою якого особа, що доводить, може переконати верифікатор у правильності певного твердження, не розкриваючи жодної додаткової інформації верифікатору.

Приклад 1. Інтерактивний доказ із нульовим знанням.

Аліса дальтонік, Боб не дальтонік. У Аліси є дві кулі однакового розміру та форми, але різного кольору, і Бобу потрібно довести Алісі, що дві кульки не одного кольору (рисунок 1.1).

У цьому сценарії Аліса є верифікатором і повинна перевірити твердження Боба, тоді як Боб є перевіряльником і повинен довести, що дві кульки різного кольору.

Аліса: Я тримаю червону кульку і синю кульку. Я зараз закладу руки за спину, щоб ти не бачив. Протягом цього часу я можу або не можу перекладати м’ячі між руками. Потім я ще раз покажу тобі кулі.

Боб: Гаразд, ти хочеш, щоб я тобі сказав, чи ти поміняла м’ячі, чи ні.

Аліса: Правильно, лише той, хто бачить кольори, зможе послідовно розповісти мені, що я зробила.

Аліса: Я перемішала кульки?

Боб: Так, м'ячі в різних руках.

Аліса: Я перемішала кульки?

Боб: Ні, м’ячі в одній руці.

У цьому доказі Аліса та Боб повторюватимуть цю гру багато разів, доки Аліса не переконається. Через 20 ітерацій Аліса переконана, що існує лише одна з 1 048 576-ї ймовірності того, що Бобу просто пощастило в кожній здогадці. Вона обґрунтовано переконана, що Боб бачить кольори.

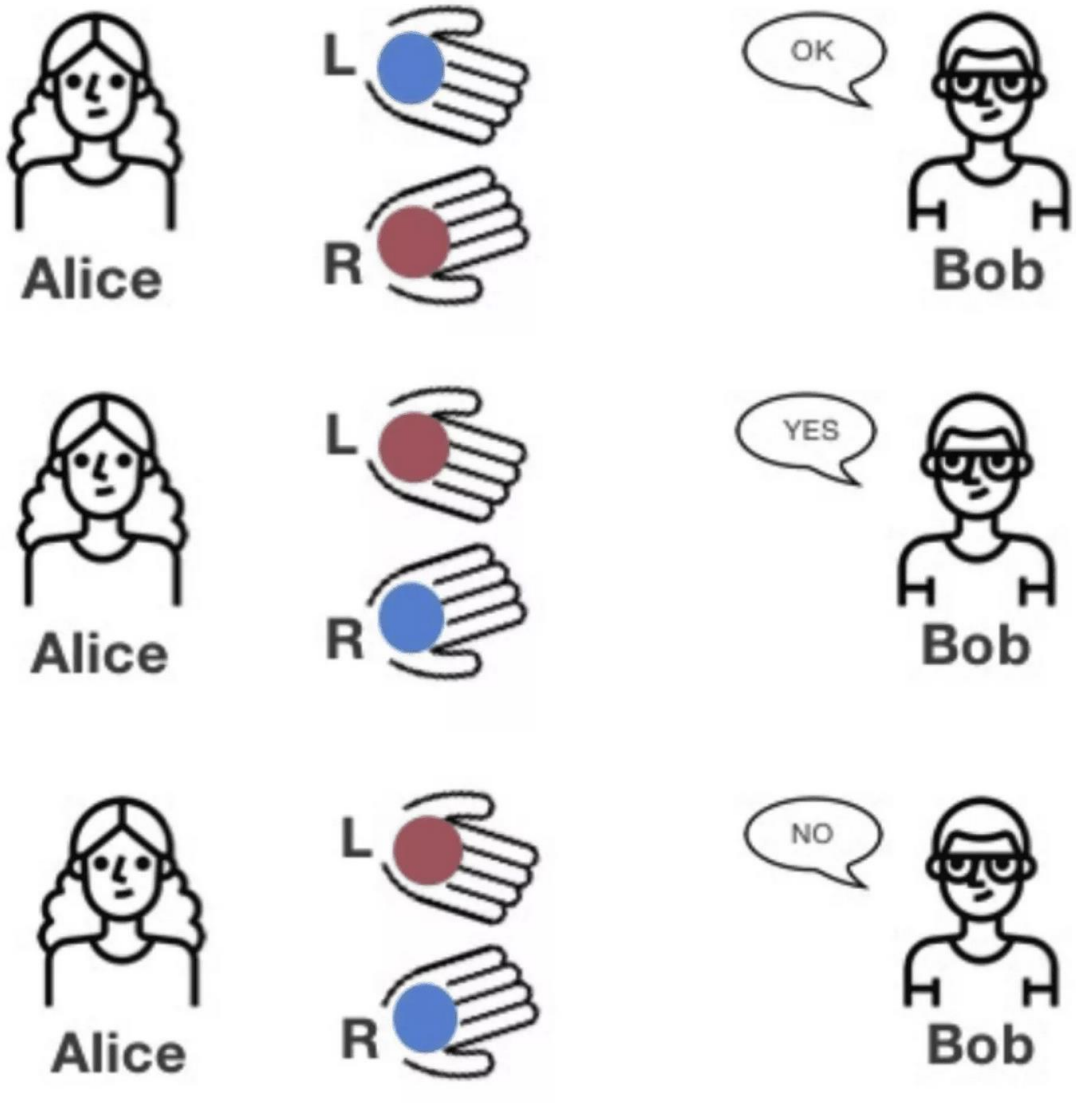

Рисунок 1.1 – Інтерактивний доказ із нульовим знанням

Цей приклад запиту-відповіді показує, що системи підтвердження з нульовим знанням є лише статистичними гарантіями того, що щось може бути правдою.

1.2 Основні властивості доказів із нульовим знанням

Докази з нульовим знанням існують уже багато років, але їхню популярність можна пояснити кількома факторами [3, 4].

1. Узагальнення доказів. Ранні докази з нульовим знанням були спеціалізованими та пристосованими до конкретних проблем, що обмежувало їхню застосовність. Розробка систем доказів більш загального призначення, таких як zk-SNARKs і zk-STARKs, зробила докази з нульовим знанням доступнішими та універсальнішими, створюючи ширший діапазон застосувань.

2. Покращення продуктивності. протягом багатьох років у системах Zero Knowledge Proof значно покращилася продуктивність, зокрема щодо ефективності перевірки та верифікації, розміру перевірки та вимог до налаштування. Ці вдосконалення зробили докази з нульовим знанням більш практичними та масштабованими, відкриваючи нові випадки використання та роблячи їх більш привабливими для реальних додатків.

3. Рекурсивна композиція доказів**.** Можливість рекурсивно складати докази з нульовими знаннями змінила правила гри для багатьох програм, особливо в контексті масштабування блокчейну. Рекурсивна композиція доказів дозволяє ефективно перевіряти велику кількість доказів, що робить можливим створення більш масштабованих і конфіденційних блокчейн-систем.

4. Прийняття блокчейну. Розвиток технології блокчейн створив попит на рішення, які можуть підвищити конфіденційність, безпеку та масштабованість. Докази з нульовим знанням добре вписуються в цей контекст, оскільки їх можна використовувати для створення конфіденційних транзакцій, захисту позаланцюжкових обчислень і створення ефективніших механізмів консенсусу. Як наслідок, зростаючий інтерес до технології блокчейн та її впровадження сприяли зростанню популярності доказів із нульовим знанням.

Доказ із нульовим знанням має три основні властивості [3].

1. Повнота: якщо твердження вірне, чесний доказ може переконати чесного верифікатора.

2. Обґрунтованість: якщо твердження хибне, жоден нечесний доказ не зможе переконати чесного верифікатора.

3. Нульове знання: якщо твердження правдиве, верифікатор не дізнається нічого, окрім факту, що твердження правдиве.

Для розуміння процесу ZKP, важливо знати його основні складові:

1. Твердження. Твердження у правдивості яких перевіряючий хоче переконати верифікатор.

2. Свідок. частина інформації, яка підтверджує правдивість твердження.

3. Доказ. Сторона, яка намагається довести правдивість твердження, не відкриваючи свідка.

4. Верифікатор. сторона, яка перевіряє правдивість твердження без вивчення свідка.

Важливими складовими доказів із нульовим знанням є: свідок, доказувач і верифікатор.

Свідок – це частина інформації, яка служить доказом для підтвердження істинності твердження, оскільки вона задовольняє певне математичне співвідношення або обмеження, визначене твердженням, що доводиться.

У ZKP довідник використовує свідка для створення доказу, який демонструє істинність твердження, не розкриваючи жодної інформації про свідка. Потім верифікатор перевіряє доказ і підтверджує, що твердження правдиве, навіть не вивчаючи зміст свідка.

Оскільки схема використовується для представлення математичного відношення або обмеження, яке має бути виконано, щоб твердження було істинним, свідок зазвичай береться як вхідне значення. Вихідні дані схеми зазвичай мають вигляд «1» (істина) або «0» (хибність), залежно від того, чи задовольняє вхідний свідок обмеженням, визначеним схемою..

1.2.1 Інтерактивне доказування Oracle

Інтерактивний Oracle Proof (IOP) – це процес зв’язку, який використовується в сфері криптографії, зокрема в системах підтвердження, де перевіряльник і верифікатор обмінюються інформацією за допомогою серії запитів до оракула. Мета полягає в тому, щоб довідник переконав верифікатор у істинності твердження, не розкриваючи жодної додаткової інформації, окрім дійсності цього твердження.

У IOP перевіряльник і верифікатор беруть участь у кількох раундах взаємодії, під час яких верифікатор надсилає запити (питання) перевіряючому, який відповідає відповідями, які зазвичай включають зобов’язання щодо конкретних значень. Потім верифікатор перевіряє ці відповіді на оракул, який зберігає попередньо обчислені відповіді або секрети, щоб перевірити послідовність і правильність тверджень перевіряючого.

У контексті доказів із нульовим знанням IOP часто поєднуються з іншими криптографічними методами, такими як поліноміальні зобов’язання, для створення ефективних і безпечних систем доказів.

1.2.2 Схеми зобов'язань

Схема зобов’язань є ключовим компонентом ZKP. Це дозволяє одній стороні, перевіряючому, прийняти значення, не відкриваючи значення іншій стороні, верифікатору. Це дозволяє доказувачу продемонструвати, що вони знають певне значення, не розкриваючи самого значення. Схема зобов’язань має бути як обов’язковою (перевірник не може змінити прийняте значення), так і прихованою (верифікатор не може дізнатися прийняте значення) [5].

Ключові властивості схеми зобов’язань полягають у тому, що для перевіряючого має бути неможливо обчислювально змінити прийняте значення після того, як зобов’язання було зроблено, і для верифікатора повинно бути обчислювально неможливо визначити прийняте значення до того, як перевірник розкриє його.

Ось кілька прикладів схем зобов’язань, які можна використовувати в ZKP:

1. Зобов'язання на основі хешування: перевіряючий генерує криптографічний хеш значення, яке вони бажають прийняти, і надсилає хеш верифікатору. Потім перевіряльник може виявити значення пізніше, а верифікатор може перевірити, чи виявлене значення відповідає хешу.

2. Зобов'язання Педерсена: перевірка генерує зобов'язання шляхом множення випадкового значення на генератор і додавання значення, яке вони бажають взяти на себе зобов'язання. Проверювач може розкрити значення пізніше, відкривши випадкове значення та генератор, а верифікатор може перевірити, чи виявлене значення відповідає зобов’язанню.

3. Поліноміальні зобов’язання: у поліноміальному зобов’язанні перевіряльник зобов’язується полінома, надаючи зобов’язання щодо його коефіцієнтів. Пізніше верифікатор може перевірити зобов’язання, перевіривши, що поліном оцінює конкретні значення в певних точках. Використання зобов’язань полінома в ZKP забезпечує конфіденційність, приховуючи значення коефіцієнтів полінома від верифікатора, і можливість перевірки, гарантуючи, що зобов’язання може зробити лише особа, яка знає значення коефіцієнтів. Поліноміальні зобов’язання є важливим будівельним блоком у різних системах підтвердження з нульовим знанням, включаючи zk-SNARK і zk-STARK, що дозволяє їм досягти стислості.

1.2.3 PLONK: конструкція SNARK із нульовим знанням

PLONK (Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge) це конструкція доказу з нульовим знанням, яка представила кілька нововведень, однією з яких є перехід від лінійної до сублінійної арифметизації [6, 7].

Це нововведення підвищило ефективність процесу побудови доказів, скоротивши як час створення доказів, так і час перевірки порівняно з попередніми конструкціями zk-SNARK.

PLONK поєднує поліноміальні зобов’язання та IOP для створення неінтерактивної системи перевірки. У PLONK перевіряльник фіксує поліноми, що представляють свідка, використовуючи схему поліноміального зобов’язання. Потім верифікатор запитує оракул у певних точках, щоб перевірити узгодженість свідка із заявою.

Конструкція PLONK використовує арифметизацію, щоб представити обчислення у вигляді набору поліноміальних рівнянь. Потім ці рівняння використовуються для створення доказу з нульовим знанням, яке перевіряє правильність обчислень, не розкриваючи жодної інформації про вхідні дані, окрім того, що обчислення правильне.

Зауважте, що «арифметизація» – це процес перетворення проблеми в еквівалентне представлення за допомогою поліномів над скінченним полем, яке потім можна використовувати як основу для створення доказу з нульовим знанням.

PLONK, Plonky, Plonky2, PlonkUp, UltraPLONK і HyperPLONK – це всі варіанти системи доказів нульового знання PLONK. Вони розроблені для покращення оригінального PLONK шляхом вирішення конкретних вимог, оптимізації продуктивності та надання додаткових функцій.

Fiat-Shamir: перетворення інтерактивних доказів у неінтерактивні. Докази з нульовим знанням можна класифікувати на інтерактивні та неінтерактивні системи доказів. Інтерактивні системи перевірки вимагають кількох раундів зв’язку між перевірювачем і верифікатором, тоді як неінтерактивні системи перевірки передбачають єдине повідомлення від перевірювача верифікатору. Евристика Фіата-Шаміра — це техніка, яка використовується для перетворення інтерактивних доказів у неінтерактивні. Він замінює випадкові виклики верифікатора детермінованою хеш-функцією, роблячи доказ неінтерактивним, але водночас забезпечує ефективну перевірку.

Rust: формально перевірена мова для доказів із нульовим знанням. Rust – це мова програмування з сильною системою типів і функціями

безпеки, що робить її ідеальною для впровадження доказів із нульовим знанням. Використовуючи Rust для створення систем ZKP, розробники можуть використовувати його формальну можливість перевірки та надійність для забезпечення коректності та безпеки.

Послідовність доказу нульового знання. Приклад роботи ZKP:

1. Той, хто доводить (прувер), хоче переконати верифікатора в істинності твердження, не відкриваючи свідка.

2. Прувер самостійно кодує свідок у поліноміальне зобов'язання, яке є криптографічним інструментом, що дозволяє перевіряючому закріпити поліном.

3. Підтверджувач і верифікатор беруть участь у Interactive Oracle Proof (IOP), системі перевірки, де вони взаємодіють через запити оракула. Під час цього інтерактивного етапу верифікатор запитує оракул у певних точках, щоб перевірити узгодженість свідка із заявою.

4. Пристрій перевірки генерує доказ за допомогою конструкції zk-SNARK, наприклад PLONK, яка поєднує поліноміальні зобов'язання та IOP для створення неінтерактивної системи доказу. Це підтвердження виконується незалежно, без участі верифікатора.

5. Евристика Фіата-Шаміра застосована для перетворення інтерактивного доказу в неінтерактивний доказ. Це досягається шляхом заміни випадкових викликів верифікатора детермінованою хеш-функцією.

6. Пристрій перевірки надсилає неінтерактивний доказ верифікатору, який тепер може ефективно перевірити доказ без будь-якої подальшої взаємодії з перевірячем.

7. Верифікатор перевіряє доказ і, якщо він дійсний, переконується, що твердження правдиве, не вивчаючи жодної додаткової інформації про свідка.

Розширені концепції в доказах із нульовим знанням. Кодування свідків і доказових елементів.

Ефективне кодування свідків і елементів доказів має вирішальне значення для компактних і ефективних доказів з нульовим знанням. Завдяки

перетворенню компонентів свідків і доказів у байтові рядки та використання стандартних типів даних і операцій можна оптимізувати створення, передачу та перевірку доказів. Це також спрощує реалізацію доказів із нульовим знанням у мові програмування

Скінченні поля та алгебраїчні структури. Кінцеві поля є важливими компонентами для побудови ефективних доказів з нульовим знанням. Вони складаються з кінцевої кількості елементів із чітко визначеними арифметичними операціями. Операції, що виконуються в скінченних полях, є критичними для процесу побудови та перевірки доказів, оскільки вони забезпечують необхідну алгебраїчну структуру для підтримки поліноміальних зобов’язань. Скінченні поля використовуються завдяки їхнім бажаним властивостям, таким як існування унікальних обернених і замикання відносно додавання, віднімання, множення та ділення.

Перевірка за допомогою коефіцієнтів у кінцевих полях. Перевірка доказів із нульовим знанням базується на перевірці узгодженості свідка з твердженням. Це робиться шляхом оцінки поліномів, введених перевірником, і перевірки того, що вони задовольняють певні умови. Ці оцінки включають коефіцієнти в скінченних полях, що забезпечує правильність доказу та гарантує, що жоден нечесний доказ не зможе переконати перевіряючого в хибному твердженні.

Булева алгебра та модуль. Булева алгебра має справу з маніпулюванням двійковими значеннями (0 і 1) і використовується в доказах з нульовим знанням для представлення логічних воріт у схемах. Модульні операції використовуються, щоб гарантувати, що всі обчислення залишаються в межах кінцевого поля, що має вирішальне значення для ефективності та безпеки доказів із нульовим знанням.

Перетворення Фур'є та багаточлени, що звертаються в нуль. Перетворення Фур’є використовуються в доказах з нульовим знанням для перетворення поліномів між різними представленнями, такими як представлення коефіцієнтів і оцінки. Це перетворення дозволяє виконувати

ефективні поліноміальні операції та оцінки. Поліноми, що зникають, є ключовим поняттям у побудові доказів із нульовим знанням, оскільки вони допомагають підтвердити послідовність і правильність зобов'язань перевіряючого.

Міркування щодо складності: засоби перевірки та верифікації. Обчислювальна складність доказів з нульовим знанням для перевірників і верифікаторів може відрізнятися залежно від конкретної системи доказів з нульовим знанням, що використовується. Однак, як правило, бажано мати ефективну перевірку, навіть для великомасштабних проблем. З іншого боку, перевіряльники зазвичай мають більше обчислювальних ресурсів і мотивовані створювати докази, тому прийнятна лінійна складність або краща.

Складність. Згортання, у контексті доказів із нульовим знанням, відноситься до техніки, яка використовується для зменшення розміру та складності проблеми чи обчислення, щоб зробити її ефективнішою для доведення та перевірки. Термін «згортання» використовується тому, що він передбачає комбінування або злиття частин проблеми або обчислення таким чином, що вихідну проблему все ще можна довести або перевірити, але з меншим представленням.

Фільтри Блума та криптографічні накопичувачі. Фільтри Блума та криптографічні акумулятори – це структури даних, які можна використовувати в контексті доказів із нульовим знанням для ефективного представлення та перевірки доказів приналежності до набору. Незважаючи на те, що ці структури даних не є безпосередньою частиною основних конструкцій доказу нульового знання, вони можуть покращити продуктивність і застосовність ZKP у різних сценаріях, коли членство в наборі є релевантним аспектом.

Важливість випадковості в доказах із нульовим знанням. Випадковість відіграє вирішальну роль у доказах з нульовим знанням, оскільки вона допомагає підтримувати конфіденційність і запобігати витоку інформації.

Випадковість часто включається в процес створення доказів і перевірки за допомогою криптографічних методів, таких як хеш-функції та генератори випадкових чисел. Забезпечення належної випадковості має важливе значення для безпеки систем доказу нульового знання.

Дерева Меркла та докази Меркла. Дерева Merkle – це структури даних, які використовуються в доказах із нульовим знанням, щоб ефективно підтверджувати включення або невключення елементів у великий набір даних із збереженням конфіденційності. Докази Merkle генеруються з дерев Merkle, що дозволяє отримати компактні докази членства, які можна перевірити. Ці структури особливо корисні в блокчейн-додатках, де вони допомагають забезпечити послідовність і цілісність даних.

Докази. Рекурсивні докази з нульовим знанням – це техніка, яка дозволяє об’єднати кілька доказів з нульовим знанням в одне доказування. Це може допомогти зменшити загальний розмір перевірки та час перевірки в певних програмах. Рекурсивні докази дозволяють ефективно перевіряти великі та складні твердження, зберігаючи конфіденційність і гарантії безпеки доказів з нульовим знанням.

1.2.4 Протоколи та примітиви ZKP

Протоколи Sigma. Протоколи Sigma – це клас інтерактивних систем підтвердження з нульовим знанням, які часто використовуються для автентифікації та цифрових підписів. Вони характеризуються ефективними процесами генерації доказів і перевірки, а також забезпечують надійну основу для різноманітних криптографічних програм. Протоколи Sigma особливо корисні для підтвердження знання дискретних логарифмів або інших алгебраїчних співвідношень без розкриття будь-якої інформації про секрети [8].

Groth16. Groth16 – популярна та ефективна конструкція zk-SNARK, яка вимагає надійного налаштування. Він широко використовується в різних додатках, таких як криптовалюти, що зберігають конфіденційність, і

обчислення поза мережею. Groth16 пропонує невеликі розміри доказів і швидкий час перевірки, що робить його привабливим вибором для впровадження доказів з нульовим знанням у середовищах з обмеженими ресурсами.

Bulletproofs. Bulletproofs – це тип неінтерактивної системи підтвердження з нульовим знанням, яка не потребує довіреного налаштування. Вони мають програми для захисту конфіденційності криптовалют і конфіденційних транзакцій, пропонуючи ефективне створення та перевірку доказів. Bulletproofs можна використовувати для створення доказів діапазону, які гарантують, що секретне значення лежить у певному діапазоні, не розкриваючи фактичне значення.

Halo / Halo 2. Halo та його наступник, Halo 2, є неінтерактивними системами з нульовим знанням, які не потребують довіреного налаштування. Вони були розроблені компанією Electric Coin і використовуються в криптовалютах, які зберігають конфіденційність, наприклад Zcash. Halo 2 покращує оригінальний протокол Halo, пропонуючи більшу ефективність і продуктивність.

Halo та Halo 2 відрізняються можливостями рекурсивної композиції. Ця функція дозволяє ефективно об’єднувати кілька підтверджень із нульовим знанням в одне доказування, зменшуючи загальний розмір перевірки та час перевірки. Це особливо корисно в блокчейн-додатках, де масштабованість і ефективність є критичними факторами.

Halo та Halo 2 використовують поєднання еліптичних кривих і поліноміальні зобов’язання [9, 10].

SNARK і STARK: масштабовані докази з нульовим знанням. SNARK і STARK – це дві популярні сімейства доказів із нульовим знанням, які забезпечують високоефективні та масштабовані рішення. Хоча обидва пропонують стислі й неінтерактивні системи перевірки, вони мають різні припущення безпеки та характеристики продуктивності. Розуміння

відмінностей між SNARK і STARK є важливим для вибору правильного підходу для конкретного застосування.

SNARKs: стислі неінтерактивні аргументи знань. SNARK — це сімейство доказів із нульовим знанням, які забезпечують невеликі розміри доказів і швидку перевірку. Вони покладаються на довірене налаштування, яке передбачає генерацію набору загальнодоступних параметрів, які повинні залишатися в таємниці для забезпечення безпеки системи. Це довірене налаштування можна розглядати як недолік, оскільки воно наражає систему на потенційні ризики, якщо секретні параметри скомпрометовані. Щоб вирішити цю проблему, часто використовується багатостороннє обчислення (MPC). SNARK використовують криптографічні методи, засновані на поєднанні еліптичних кривих, які можуть бути чутливими до майбутніх атак квантових обчислень.

STARKs: масштабовані прозорі аргументи знань. STARKs (Scalable Transparent Arguments of Knowledge) – це клас доказів з нульовим знанням, які не вимагають надійного налаштування, пропонуючи прозорі та безпечні системи доказів.

Хоча STARK забезпечують рівень ефективності, який можна порівняти з SNARK, вони зазвичай генерують більші розміри проб, що може вплинути на їх придатність для певних застосувань. Однак STARK покладаються на криптографічні припущення, які вважаються постквантовими безпечними.

Припущення безпеки в протоколах з нульовим знанням. Різні системи перевірки нульового знання покладаються на різні припущення щодо безпеки, щоб гарантувати свої властивості конфіденційності та надійності. Деякі з найпоширеніших припущень безпеки включають:

1. Проблема дискретного логарифму (DLP): DLP базується на припущенні, що неможливо з точки зору обчислень обчислити логарифм випадкового елемента відносно генератора в кінцевій циклічній групі. Багато конструкцій zk-SNARK і деякі zk-STARK спираються на це припущення.

2. Обчислювальна проблема Діффі-Хеллмана (CDH): припущення CDH стверджує, що обчислити секрет Діффі-Хеллмана з двох відкритих ключів, не знаючи закритих ключів, неможливо. Деякі zk-SNARK, особливо ті, що базуються на парах, покладаються на це припущення для своєї безпеки.

3. Проблеми короткого цілого розв’язку (SIS) і навчання з помилками (LWE): SIS і LWE є проблемами, пов’язаними з криптографією на основі решітки, і вони припускають, що обчислювально неможливо знайти короткий вектор у решітці або розв’язати певні лінійні системи. із зашумленими входами відповідно. zk-STARK та інші постквантові системи доказів нульового знання часто покладаються на ці припущення для своєї безпеки.

4. Програми квадратичного інтервалу (QSP) і програми квадратичної арифметики (QAP): QSP і QAP є алгебраїчними представленнями схем, які є основою для багатьох конструкцій zk-SNARK. Безпека цих систем ґрунтується на припущенні, що обчислювально неможливо знайти дійсний доказ, не знаючи дійсного свідка для схеми.

5. Модель випадкового оракула: деякі системи доказів із нульовим знанням, особливо ті, що використовують евристику Фіата-Шаміра для перетворення інтерактивних доказів у неінтерактивні, покладаються на модель випадкового оракула. Ця модель припускає існування справді випадкової функції (випадкового оракула), до якої можуть звертатися всі сторони. На практиці криптографічні хеш-функції використовуються як заміна ідеального випадкового оракула.

1.2.5 Пов'язані технології

Багатостороннє обчислення (MPC) – це криптографічний метод, який дозволяє кільком сторонам спільно обчислювати функцію на основі своїх приватних вхідних даних, не розкриваючи самих вхідних даних. MPC було інтегровано в різні конструкції з нульовим знанням для підвищення конфіденційності та безпеки. У контексті ZKP MPC зазвичай

використовується на етапі надійного налаштування для генерації параметрів, таких як загальний еталонний рядок (CRS). Використовуючи MPC у процесі генерації параметрів, можна пом’якшити проблеми довіри, оскільки кілька сторін спільно генерують CRS. Це гарантує, що до тих пір, поки хоча б одна сторона поводиться чесно, CRS залишається в безпеці. У деяких системах з підтвердженням нульового знання етап налаштування може бути розроблений як універсальний або оновлюваний. Універсальне налаштування дозволяє кільком примірникам системи перевірки використовувати однакові параметри, підвищуючи ефективність і зменшуючи складність налаштування. Налаштування, які можна оновлювати, дають змогу оновлювати існуючі параметри кількома сторонами, зменшуючи потребу в довірі до початкового налаштування та підвищуючи загальну безпеку системи.

Крім того, MPC можна використовувати як в інтерактивних, так і в неінтерактивних системах підтвердження нульового знання, а в поєднанні з іншими криптографічними примітивами, такими як порогові підписи, він може додатково підвищити конфіденційність і безпеку.

Різниця між доказами з нульовим знанням і гомоморфним шифруванням [11].

Докази з нульовим знанням і гомоморфне шифрування є криптографічними методами, які використовуються для збереження конфіденційності, але вони служать різним цілям і мають відмінні властивості.

Докази з нульовим знанням дозволяють перевіряючому переконати верифікатор у істинності твердження, не розкриваючи жодної додаткової інформації про твердження чи сам доказ. Докази з нульовим знанням в основному використовуються для забезпечення дійсності твердження без витоку конфіденційної інформації.

З іншого боку, гомоморфне шифрування – це схема шифрування, яка дозволяє виконувати обчислення безпосередньо із зашифрованими даними

без їх попереднього розшифровування. Це дозволяє безпечно обчислювати конфіденційні дані, зберігаючи конфіденційність. Гомоморфне шифрування в основному використовується для безпечних обчислень і обробки даних у сценаріях, коли кільком сторонам потрібно співпрацювати над конфіденційною інформацією, не розкриваючи їхні окремі вхідні дані. Він дозволяє аналізувати та приймати рішення із збереженням конфіденційності, що робить його важливим інструментом для безпечних хмарних обчислень, машинного навчання із збереженням конфіденційності та інших програм, які потребують конфіденційності даних.

1.3 Приклади використання доказів із нульовим знанням

ZKP – це перспективна технологія, і вона вже знайшла ряд успішних використань.

Анонімні облікові дані. Для реалізації анонімних систем облікових даних можна використовувати підтвердження з нульовим знанням. У такій системі користувач може довести володіння певними атрибутами (наприклад, старше певного віку або дійсне членство), не розкриваючи свою особу. Це дозволяє постачальникам послуг перевіряти відповідність вимогам, не отримуючи доступу до конфіденційної інформації, зберігаючи конфіденційність користувачів. У цьому сценарії докази з нульовим знанням служать технологією стиснення знань, оскільки вони дозволяють користувачеві підтверджувати твердження, не розкриваючи базові дані.

Розумні контракти, що зберігають конфіденційність. У контексті технології блокчейн докази з нульовим знанням можна використовувати для створення розумних контрактів, які зберігають конфіденційність. Наприклад, додаток для децентралізованих фінансів (DeFi) може дозволити користувачам підтверджувати свою платоспроможність або наявність коштів як технологія стиснення знань (дозволяє виборцям підтверджувати свою

правомочність, не розкриваючи особисту інформацію), і перевірена технологія обчислень (перевірка правильність підрахунку голосів без викриття окремих голосів). не розкриваючи точні суми, які зберігаються на їхніх рахунках. У цьому випадку докази з нульовим знанням служать як стисненням знань, так і перевіреними обчислювальними технологіями, що дозволяє користувачам підтверджувати твердження, пов’язані з їхніми активами, зберігаючи при цьому конфіденційність інформації..

Безпечне голосування. Для реалізації безпечних систем голосування, що зберігають конфіденційність, можна застосувати докази з нульовим знанням. У таких системах виборці можуть довести, що їхні голоси дійсні, не розкриваючи своїх виборчих уподобань. Це забезпечує цілісність процесу голосування при збереженні анонімності виборця. Докази з нульовим знанням відіграють у цьому прикладі подвійну роль як технологія стиснення знань (дозволяє виборцям підтверджувати свою правомочність, не розкриваючи особисту інформацію), і технологія обчислення, яку можна перевірити (перевірка правильності підрахунку голосів без розкриття окремих голосів).

Служби визначення місцезнаходження, що зберігають конфіденційність. Служби визначення місцезнаходження часто вимагають від користувачів повідомляти своє точне місцезнаходження, що потенційно порушує конфіденційність. Підтвердження з нульовим знанням можна використовувати для створення конфіденційних служб визначення місцезнаходження, які дозволяють користувачам доводити свою присутність у певній області чи поблизу, не розкриваючи своїх точних координат. У цьому сценарії докази з нульовим знанням діють як технологія стиснення знань, що дозволяє користувачам підтверджувати твердження на основі місцезнаходження, не розголошуючи своє точне місцезнаходження. Крім того, як перевірена обчислювальна технологія, докази з нульовим знанням можуть гарантувати правильність заяв про місцезнаходження без розкриття базових даних.

2 АЛГОРИТМИ ДОКАЗУ З НУЛЬОВИМ РОЗГОЛОШЕННЯМ

2.1 Неінтерактивний випадковий доказ з використання хеш-значення

Одним із найбільш широко використовуваних методів для доказів із нульовим знанням є «неінтерактивний випадковий доказ до оракула» який визначається як евристика Фіата-Шаміра [10].

«Оракули – це об'єкти, які отримують та захищають зовнішні дані для блокчейнів, дозволяючи смарт-контрактам взаємодіяти із зовнішніми системами» [11].

Розглянемо використання хеш-значення для неінтерактивної випадкової частини оракула. Зазвичай Аліса передає випадкове значення тому, хто хоче підтвердити свою особу, але в цьому випадку вона використовуватиме хеш-значення для рандомізації головоломки та відповіді.

Алгоритм роботи наступний:

1. Спочатку всі складають головоломку і мають секрет (x).

В даному випадку головоломкою буде вираз:

$$y = g^{\mathrm{x}}$$

де $g,$ – спільно погоджене значення; x – секрет, який Аліса доводить, що вона його знає.

Наприклад: при g дорівнює 13, а x дорівнює 11, тому $g^{\mathrm{x}} = 13^{11} = 1792160394037.$

2. Далі генерується випадкове число (v) і обчислюється t:

$$t = g^{\mathrm{v}}.$$

Припустимо, що значення v дорівнює 8. Тоді $t = 815730721.$

3. Обчислюємо хеш-значення (c), створене з $g,\ y$ і t:

$$c = MD5(g + y + t).$$

Припустимо, це дає нам 12 (як правило діапазон значення обмежується).

4. Обчислюємо r :

$$r = v - c \times t(mod\ p = -124.$$

5. Надсилаємо t і r, щоб підтвердити свою особу: $[815730721, -124]$

6. Кожен, хто знає, хто хоче підтвердити свій ідентифікатор, потім обчислює (де c можна обчислити як хеш g, y і t):

$$t = g^{\mathrm{r}} \times y - c.$$

У цьому випадку розрахунок дає 815730721, що збігається зі значенням t, яке надіслала Аліса, тому вони підтвердили свій дентифікатор (рисунок 2.1).

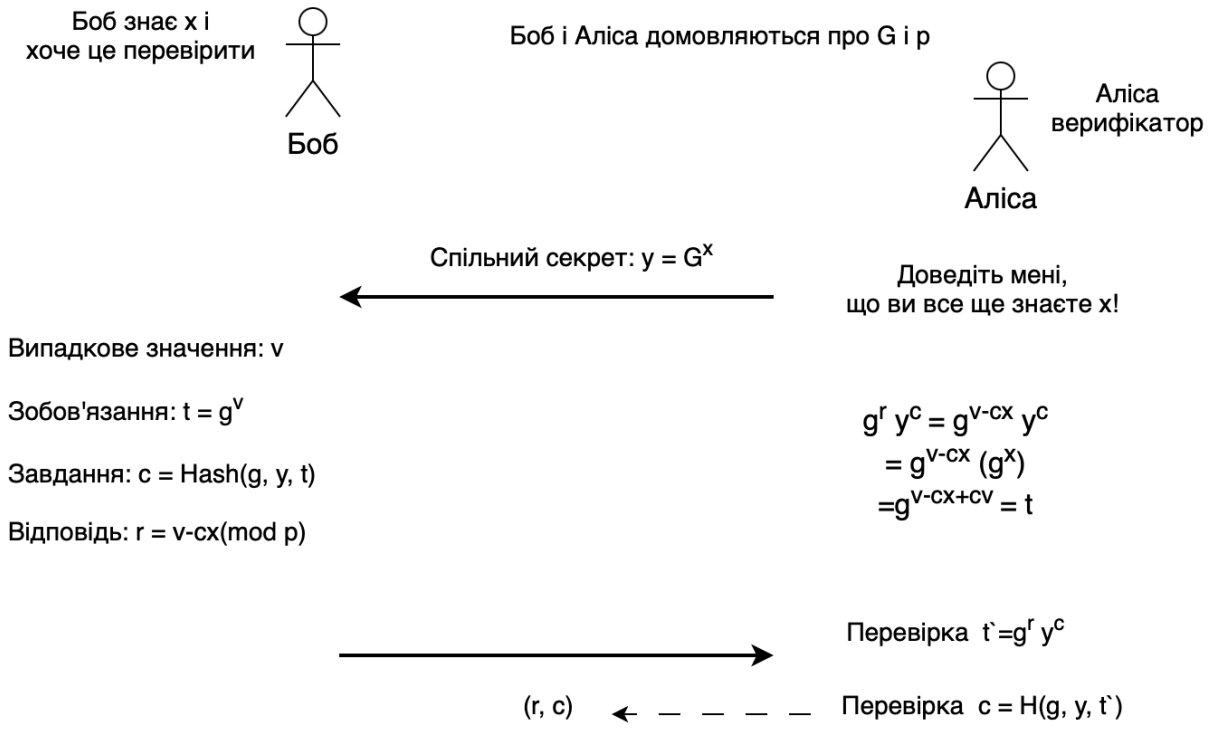

Рисунок 2.1 – Графічне представлення неінтерактивного випадкового доказу

Щоразу Аліса генерує нове випадкове число і щоразу доводить, що вона знає значення t.

Приклад. Нехай генератор $g = 41$;

секрет: x= 55;

випадкове число: $v = 8$;

обчислюємо t:

$t = 41^8 = 7984925229121$;

$y = 41^{55} =$

$= 504791600208990440968953111006108435519750467636784622211$
$35273981\ 904186426760632708618201$;

Хеш значення: $c = 14$.

Аліса відправляє: $(t,\ r) = (\ 7984925229121, -762\ )$.

У цьому випадку неінтерактивний оракул є хеш-функцією [12].

2.2 Доказу із нульовим розголошенням з використання дискретних логарифмів

Неінтерактивний доказ із нульовим розголошенням (Zero Knowledge Proof, NI-ZKP) дозволяє доводити знання цінності, не видаючи його.

У цьому прикладі використовується перетворення Фіата-Шаміра, де Аліса знає секретне значення x. Потім Аліса може показати загальнодоступне значення $g^x(mod\ p)$. Тоді Боб може довести, що Аліса знає значення x за допомогою NI-ZKP.

У цьому випадку Аліса згенерує випадкове значення (v, а потім надсилає Бобу значення $g^v(mod\ p)$, r і c, і тоді Боб зможе сказати, що Аліса знає значення x.

Використаємо метод Фіата-Шаміра, щоб довести, що Аліса знає x. Оскільки ми використовуємо неінтерактивний метод, Аліса створює власне завдання за допомогою формули:

$$c = Hash((G)||(x)||(g^x(mod\ p)))$$

і Аліса обчислює:

$$X = g^x(modp)$$

Потім Аліса створює випадкове значення (v) і обчислює:

$$V = g^v(modp)$$

і

$$r = (v - cx1)(modp - 1).$$

Вона надсилає значення $V,\ r$ і c Бобу, який обчислює:

$$Ch = (g^r(mod\ p) \times X^c(mod\ p))(mod\ p)$$

Якщо V дорівнює Ch, Аліса довела, що вона знає x.

Приклад. Тепер доведемо, що Аліса знає x за допомогою методу Фіата-Шаміра.

Нехай значення, яке потрібно довести – 51, а секрет X= 48; p= 15827311950925515283;

$v =$ 92105861389274556 31;

$V =$ 48885127180097411 01;

$r = 11001259345952756056;$

$c = 32914993906046767535445950367324706554 9;$

$Ch = 48885127180097411 01.$

Отже, знання доведено.

2.2 Автентифікація на основі доказу із нульовим розголошенням

Розглянемо спосіб, за допомогою якого Аліса могла б створити власний пароль, а потім зареєструвати щось у Боба, щоб він міг створити для неї виклик, щоб довести, що вона все ще знає пароль. Для цього нам потрібні докази з нульовими розголошенням (ZKP), і ми гарантуємо, що процес реєстрації виконується у спосіб, який має високий рівень довіри.

Тепер Аліса може створити початкове реєстраційне значення, яке Боб може зберегти. Важливим моментом є те, що визначити її оригінальний пароль за реєстраційним значенням, яке ми використовуємо, майже неможливо. Тепер, коли Аліса хоче ввійти в систему Боба, вона надсилає йому випадковий виклик, і вона повинна дати правильну відповідь, щоб показати, що вона все ще знає свій пароль. Якщо вона подає правильну відповідь, Боб дозволяє їй увійти. Кожного разу, коли Аліса хоче увійти, Віктор надсилає їй інший виклик (рисунок 2.2).

Існує багато способів вирішення проблеми, але одним із найбільш широко використовуваних методів є «неінтерактивний довільний доступ до оракула» для доказів із нульовим розголошенням. Він визначається як евристика Фіата-Шаміра [11]. Він використовує дискретні логарифми, щоб створити складну головоломку для Єви, легку для Аліси, щоб довести, і для Боба, щоб перевірити. Якщо ми зробимо головоломку настільки складною, що Єві доведеться споживати масу обчислювальної потужності лише для вирішення одного елемента головоломки.

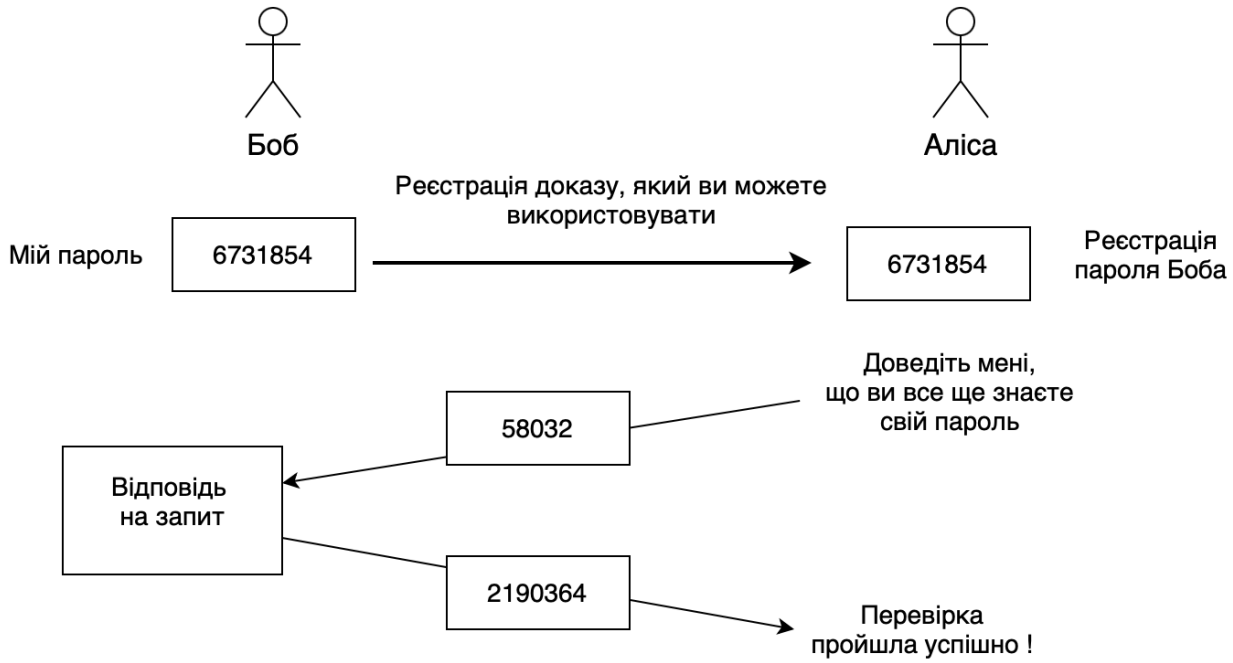

Рисунок 2.2 – Схема автентифікація на основі доказу із нульовим розголошенням

Спочатку Боб і Аліса узгоджують значення генератора (g) і просте число (p).

Алгоритм автентифікації складається з наступних кроків:

1. Спочатку Аліса вибирає свій пароль (pass), обчислює його хеш, а потім перетворює його на ціле число (x).

$$x = Hash(pass),$$

$$y = g^x$$

і щоб ускладнити ситуацію та дозволити створити головоломку, потрібно вибрати просте число (p) і виконати операцію:

$$y = g^x(modp).$$

Аліса надсилає Бобу значення y, і він зберігає його. У коді це виглядає так (рисунок 2.3):

x = int(hashlib.sha256(passw).hexdigest()[:8], 16) % p);

y= g**x % p.

2. Тепер Аліса хоче увійти, тому вона генерує нове випадкове число та надсилає Бобу значення t:

$$t = g^v.$$

3. Тоді Боб надсилає їй виклик (c), який є випадковим значенням, яке він використовував раніше.

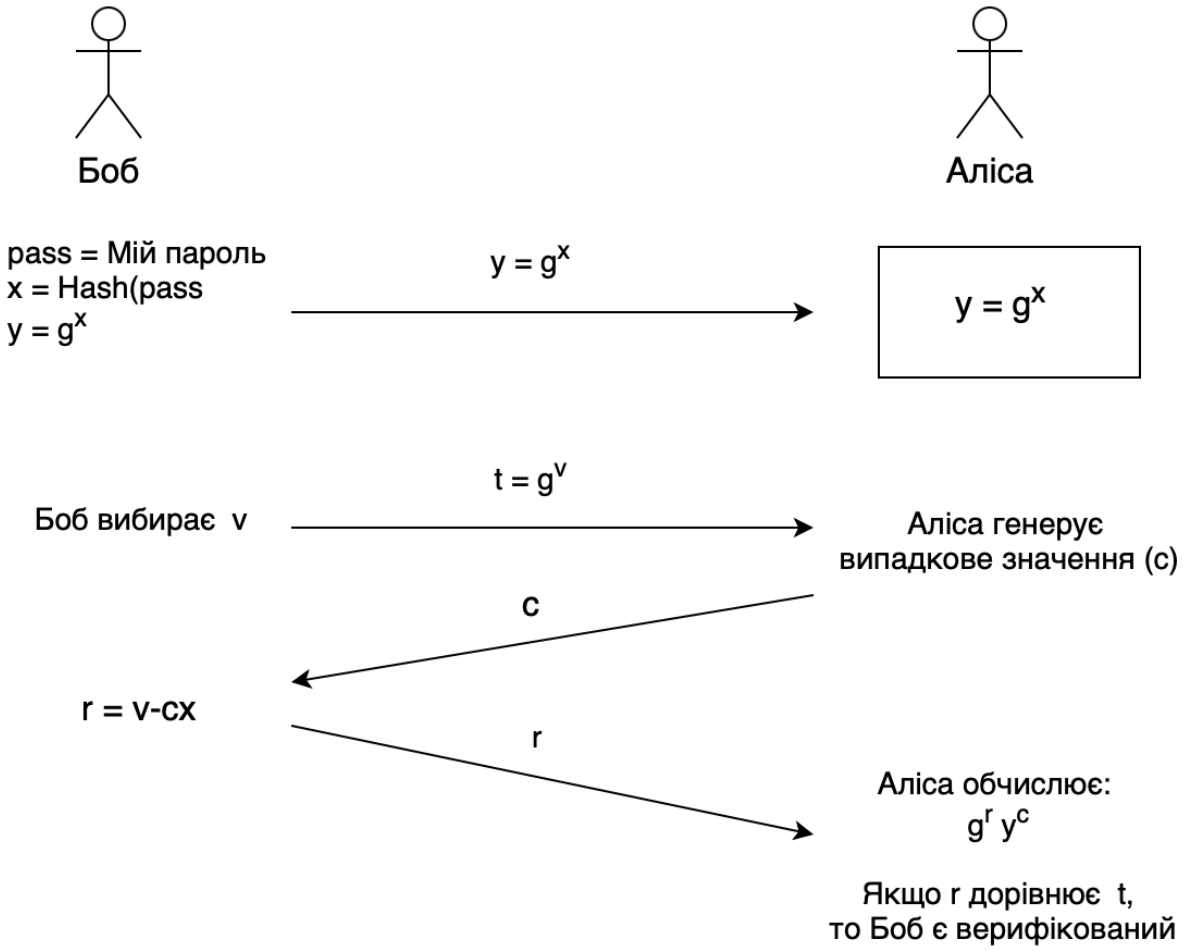

Рисунок 2.3 – Графічне представлення алгоритму автентифікації

4. Аліса генерує інше випадкове значення (v), а тепер бере значення c і обчислює:

$$r = v - c \times t.$$

Потім вона надсилає Бобу обчислене нею значення r.

5. Потім він обчислює:

$$val = g^c\, y^c,$$

і перевіряє, чи результат дорівнює t дорівнює значенню, і якщо вони однакові, Аліса довела свою ідентичність, і Єва залишається без нічого.

Операції виконуються за модулем $(mod\, p)$, і це працює завдяки властивості логарифмів:

$$g^r \times y^c = g^{v-cx} \times g^{xc} = g^{v-cx+cx} = g^v.$$

Щоб скинути пароль, Аліса просто зв’язується з Бобом і виконує багатофакторну автентифікацію, яку вона зробила під час реєстрації свого секрету, а потім реєструє новий секрет.

Приклад. Нехай пароль: hello555.

Узгоджені параметри.

P= 997 (просте число).

G= 7 (генератор).

Секрет Аліси: x= 485.

Випадкові значення:

$c = 571$ (випадкове значення Боба).

$v =$ 213 (випадкове значення Аліси).

Спільна цінність:

$$g^x\, mod\, p = 633;$$

$$r = -276722.$$

Результати обчислень:

$$t = g ** v \,\%\, n \; = 18;$$

((g**r) * (y**c))=18.

Отже, Аліса довела, що знає пароль.

Єва не має обчислювальної потужності, щоб розгадати пароль Аліси, і Боб також його не знає.

Доказ із нульовим розголошенням є перспективним напрямом розробки систем автентифікації, який не потребує зберігання паролів.

2.3 Неінтерактивна схема доказу з нульовим розголошенням на основі еліптичних кривих

За допомогою схеми Фіата-Шаміра ми можемо довести, що знаємо значення, фактично не розкриваючи початкове значення. Далі Боб доведе Алісі, що він усе ще знає свій пароль. Боб генерує випадкове число (v), обчислює значення виклику (c) і надсилає доказ того, що він усе ще знає x.

Для того щоб не розголошувати конфіденційну інформацію можна зареєструвати версію свого секрету, а потім довести, що ви його все ще знаєте. Для цього використаємоо метод Фіата-Шаміра. Один із способів, розглянутий раніше, реалізувати це – дискретні логарифми, але можна використати потужнішу математику, наприклад, криптографію еліптичної кривої. Спочатку Боб і Аліса узгоджують дві точки на вибраній еліптичній кривій (G і H). Далі Боб бере свій секрет і хешує його, щоб отримати секретне значення (x). Потім він обчислює xG і xH (тобто точки G і H додають x разів) і надсилає їх Алісі. Потім Аліса реєструє їх (рисунок 2.4).

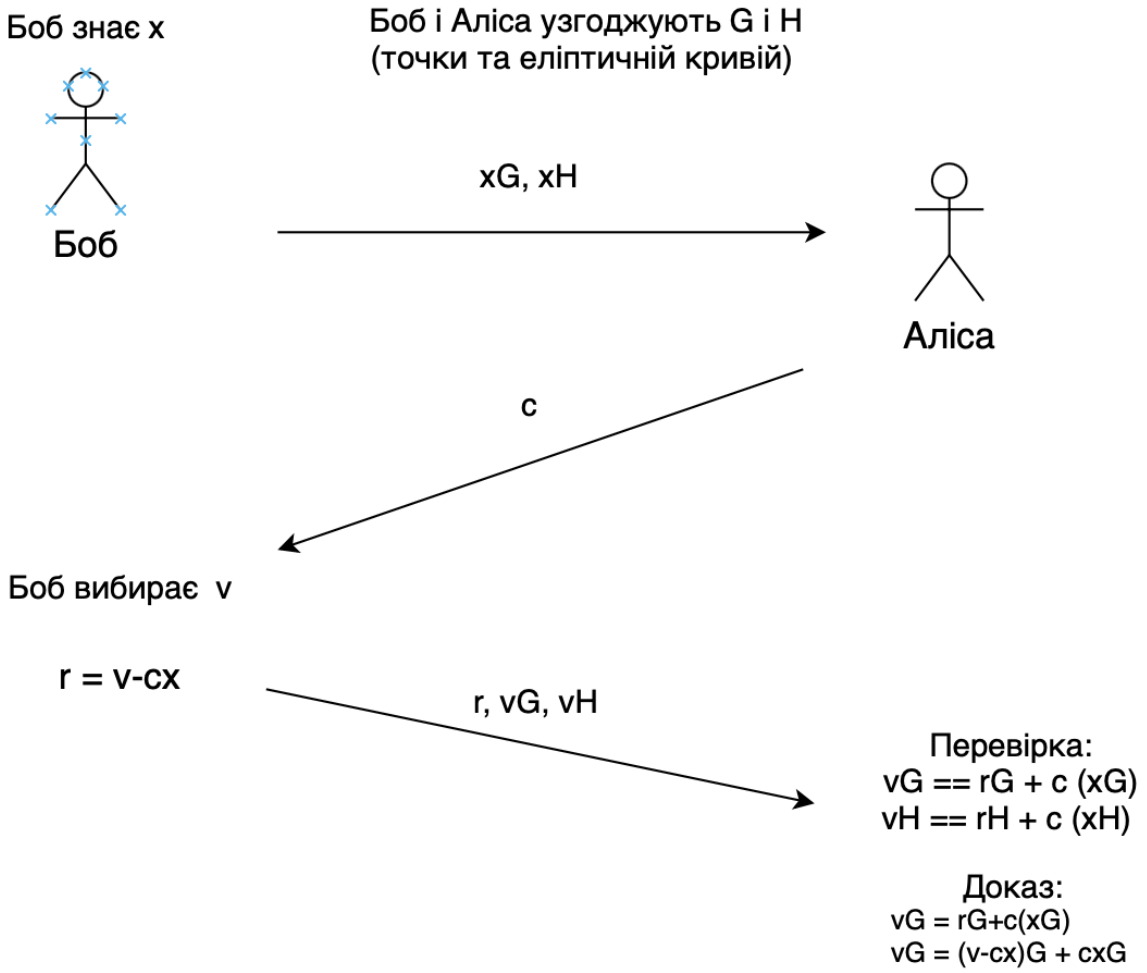

Рисунок 2.4 – Схема доказу з нульовим розголошенням на основі еліптичних кривих

Потім Аліса надсилає випадкове значення виклику (c) і надсилає його Бобу. Потім Боб генерує власне випадкове значення (v) і обчислює r:

$$r = v - cx.$$

Це робиться у вигляді множення (cx) і віднімання $(v - cx)$ еліптичної кривої. Він також обчислює vG і vH і надсилає r, vG і vH назад Алісі. Потім Аліса перевіряє, що:

$$vG = rG + c(xG)$$

і

$$vH = rH + c(xH).$$

Якщо вони однакові, відповідно, Боб перевірив свій пароль.

Приклад перевірки:

Боб і Аліса погоджую G і H:

G:
a9cdf48cf6ab65512c54afeb39fa52b783d956f5c67397b39e73b82f84a2a0ac

H:
e2ba8b9ffd9b6828465349e7b502feb0e46331cc5f60b9f31b6c20d5f89ea6ff

Пароль Боба: hello555

Секрет Боба (x):
fd6c6f8815debe9c054de2b385c8a909c70bb5906bcc7a6d92e 5501c6c6eef0c

Боб надсилає наступні значення:

xG:
7d53df06726a647f0a9a14d32718c9657499694bd6551e3617b2e57ff79ff1c5

xH: 2aa13b8c21b721eb868c77273a249ee36111cb07839403b68374adf782ff
87d4

Аліса надсилає виклик:

c: 365454a65d92d888183277635fc62c888b74df560ccd17042ca356
00aff60308

Боб обчислює:

v: 85333cf96092e1f4b7e379d1a07ee0c687d507bee1e2cbd89142d315
21e4d20c

r: 1f8aa280c1084a3d2a83dd4be38d9e8a8d274e392696ef0e955f743bf
656b207

Отже, доказ правильний.

2.4 Система обмежень першого рангу

Основною концепцією в zk-SNARK є використання R1CS (система обмежень рангу 1), яка підтримує формат, який легко перевірити, для доказів з нульовим знанням. У цьому випадку ми використаємо практичний приклад, використаний для пояснення zk-SNARK в [13].

Спочатку починаємо з доказу того, що Боб знає розв'язок кубічного рівняння. Розглянемо наступний такий приклад:

$$x^3 + x + 5 == 35.$$

Розв'язком цього рівняння є $x = 3$. Спростимо дане рівняння, щоб воно включало операції додавання, віднімання, множення та ділення:

```
«def qevl(x):
    y = x**3
    return x + y + 5»
```

Наступним кроком є приведення рівняння до вигляду схеми, де ми отримуємо серію тверджень, які приймають форму [13]:

$$x = y(op)z,$$

де op може бути «+», «–», «*» або «/». В основному це можна розглядати як створення логічної схеми з елементами І (*) і АБО (+). В результаті отримаємо [13]:

$$sym\_1 = x * x;$$
$$y = sym\_1 * x;$$
$$sym\_2 = y + x;$$
$$\sim out = sym\_2 + 5.$$

Це можна розглядати як наявність двох вентилів I (множників), а потім двох вентилів АБО (суматорів). Далі ми перетворюємо у формат R1CS (Rank-1 Constrain System), який визначає три вектори: A, B і C.

Розв'язком цього є вектор (S),

де:

$$\boldsymbol{S}.A \cdot S.B \cdot S - C \cdot S = [0].$$

У даному випадку операція «.» – скалярний добутком.

Отже, результат обробки кубічного рівняння, яке ми визначили вище має вигляд:

R1CS з коду:

a: [[0 0 1 0 0 0 0 0] [0 0 1 0 0 0 0 0] [0 0 1 0 1 0 0 0] [5 0 0 0 0 1 0 0]
[0 0 0 0 0 0 1 0] [0 1 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]];

b: [[0 0 1 0 0 0 0 0] [0 0 0 1 0 0 0 0] [1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]];

c: [[0 0 0 1 0 0 0 0] [0 0 0 0 1 0 0 0] [0 0 0 0 0 1 0 0] [0 0 0 0 0 0 1 0]
[0 1 0 0 0 0 0 0] [0 0 0 0 0 0 1 0] [0 0 0 0 0 0 0 1]].

У даному випадку Свідок (Боб) доводить, що він знає розв'язок, видавши:

доказ [1 35 3 9 27 30 35 1].

Відповідно, скалярний добуток для свідка, з використанням матриці A, дорівнює:

[**0 0 1 0 0 0 0 0**] [1
[0 0 1 0 0 0 0 0] 35
[0 0 1 0 1 0 0 0] 3
[5 0 0 0 0 1 0 0] 9
[0 0 0 0 0 0 1 0] 27
[0 1 0 0 0 0 0 0] 30
[1 0 0 0 0 0 0 0] 35
1]

Результат обчислення:

$$0 \times 1 + 0 \times 35 + 1 \times 3 + 0 \times 9 + 0 \times 27 + 0 \times 30 + 0 \times 35 + 0 \times 1 = 3.$$

Для матриці B:

$$\begin{matrix}[\mathbf{0\ 0\ 1\ 0\ 0\ 0\ 0\ 0}] & [1 \\ [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0] & 35 \\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] & 3 \\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] & 9 \\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] & 27 \\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] & 30 \\ [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0] & 35 \\ & 1]\end{matrix}$$

Результат обчислення:

$$0 \times 1 + 0 \times 35 + 1 \times 3 + 0 \times 9 + 0 \times 27 + 0 \times 30 + 0 \times 35 + 0 \times 1 = 3.$$

І далі для матриці C:

$$\begin{matrix}[\mathbf{0\ 0\ 0\ 1\ 0\ 0\ 0\ 0}] & [1 \\ [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0] & 35 \\ [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0] & 3 \\ [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0] & 9 \\ [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0] & 27 \\ [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0] & 30 \\ [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1] & 35 \\ & 1]\end{matrix}$$

Результат обчислення:

$$0 \times 1 + 0 \times 35 + 0 \times 3 + 1 \times 9 + 0 \times 27 + 0 \times 30 + 0 \times 35 + 0 \times 1 = 9.$$

І тепер ми бачимо, що для A.B−C при обчисленні першого значення ми отримуємо:

$$A0 \cdot S \times B0 \cdot S - C0 \cdot S = 3 \times 3 - 9 = 0.$$

Отже, Боб (свідок) надав правильну першу частину доказу. Обчисливши інші сім значень для кожного рядка A, B і C ми побачимо, що результат буде нульовим. Таким чином Боб довів, що він знає розв’язок кубічного рівняння.

Використовуючи бібліотеку numpy мови Python покажемо, що в цьому випадку A.S це $[3, 3, 30, 35, 35, 35, 1]$, B.S це $[3, 9, 1, 1, 1, 1, 1]$, а C.S дорівнює $[9, 27, 30, 35, 35, 35, 1]$.

Результатом A.S x B.S є $[9, 27, 30, 35, 35, 35, 1]$, і при відніманні C.S, отримаємо: $[0, 0, 0, 0, 0, 0, 0]$. Це доводить, що Боб знає рішення.

Приклад.

```
x (private input): 5
y (public input): 35
Flat: func exp3(private a):
        b = a * a
        c = a * b
        return c
func main(private s0, public s1):
        s3 = exp3(s0)
        s4 = s3 + s0
        s5 = s4 + 5
        equals(s1, s5)
        out = 1 * 1
```

Відповідно, схема визначається як:

$$\&\{8\ 1\ 8\ [s0]\ [s1]\ [one\ s1\ s0\ b0\ s3\ s4\ s5\ out]\ []\ [\{in\ \ s1\ \ []\ []\}$$
$$\{in\ \ s0\ \ []\ []\}\{*\ s0\ s0\ b0\ b0 = s0 * s0\ []\ []\}\{*\ s0\ b0\ s3\ s3\ =$$
$$s0 * b0\ []\ []\}\{+\ s3\ s0\ s4\ s4 = s3 + s0\ []\ []\}\{+\ s4\ 5\ s5\ s5\ =$$
$$s4 + 5\ []\ []\}\{*\ s5\ 1\ s1\ equals(s1, s5): s1 ==$$
$$s5\ *\ 1\ []\ []\}\{*\ s1\ 1\ s5\ equals(s1, s5): s5 ==$$
$$s1\ *\ 1\ []\ []\}\{*\ 1\ 1\ out\ out = 1 * 1\ []\ []\}]\ \{[]\ []\ []\}\}$$

Доказ дорівнює: [1 135 5 25 125 130 135 1].

R1CS з коду:

a: [[0 0 1 0 0 0 0 0] [0 0 1 0 0 0 0 0] [0 0 1 0 1 0 0 0] [5 0 0 0 0 1 0 0]
[0 0 0 0 0 0 1 0] [0 1 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]];
b: [[0 0 1 0 0 0 0 0] [0 0 0 1 0 0 0 0] [1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]
[1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0] [1 0 0 0 0 0 0 0]];
c: [[0 0 0 1 0 0 0 0] [0 0 0 0 1 0 0 0] [0 0 0 0 0 1 0 0] [0 0 0 0 0 0 1 0]
[0 1 0 0 0 0 0 0] [0 0 0 0 0 0 1 0] [0 0 0 0 0 0 0 1]].
As: [[2 0 5 7 3 5 0] [4 8 1 5 3 0 1] [10 10 10 2 3 8 2] [0 0 0 0 0 0 0] [2 8 8 6 4 2 3]
[9 5 6 0 9 8 7] [10 2 8 8 5 8 3] [0 0 0 0 0 0 0]];
Bs: [[4 9 9 4 1 5 1] [0 0 0 0 0 0 0] [7 7 0 8 4 10 9] [1 6 2 10 6 7 1][0 0 0 0 0 0 0]
[0 0 0 0 0 0 0] [0 0 0 0 0 0 0] [0 0 0 0 0 0 0]];
Cs: [[0 0 0 0 0 0 0] [10 2 8 8 5 8 3] [0 0 0 0 0 0 0] [7 7 0 8 4 10 9] [1 6 2 10 6 7 1]
[2 8 8 6 4 2 3] [2 2 7 5 1 8 8] [1 8 8 7 2 9 9]].

Відповідно, доказ дорівнює: [5 5 9 7 4 6 4 8 6 6 6 10 1].

2.5 Порівняння схем поліноміальних зобов’язань

Алгоритми Zero-Knowledge Proof побудовані з використанням криптографічних примітивів – поліноміальне зобов'язання (Polynomial Commitment, PC) [14 – 16].

Зобов’язання – це загальнодоступне значення, пов’язане з оригінальним повідомленням (обчислювальна прив’язка), надане комітером, який не розкриватиме повідомлення (приховування). Комітер повинен відкрити це зобов’язання та надіслати повідомлення верифікатору, щоб перевірити відповідність між зобов’язанням і повідомленням. PC можна розглядати як зобов’язання щодо певного полінома P, і комітер може довести, що значення полінома в певній точці z задовольняє $P(z) = a$ за допомогою доказу, не розкриваючи поліном P. Існує кілька схем для реалізації поліноміальних зобов’язань: [17-19]

1) зобов’язання KZG10 на основі парної групи [20];

2) зобов’язання IPA: на основі групи дискретних журналів [21];

3) зобов’язання FPI: на основі хеш-функції [22];

4) Зобов’язання DARKS: на основі невідомої групи замовлення [23].

Розглянемо відмінності між наведеними вище поліноміальними схемами з різних точок зору.

Форми обчислень звичайних PC приведені в таблиці 2.1. У різних ZKP алгоритми різні саме тому, що застосовуються різні схеми поліноміальних зобов’язань.

Таблиця 2.1 – Форми обчислень поліноміальних зобов'язань

Схема PC	KZG10	IPA	FRI	DARKS
Техніка на низькому рівні	Парна група	Дискретні логарифми	Хеш функція	Невідома група замовлення
Налаштування	G1, G2 – групи; g1, g2 – генератори; e – парна функція; Sk секретне значення в F;	G – еліптична крива; g^n– незалежні елементи в G;	H – хеш функція; w – одиничний корінь;	N – невідомий порядок; g – випадкове число вN; q – велике ціле число;
Зобов’язання	$(a_0 s^0 + \cdots + a_n s^n) g_1$	$a_0 g_0 + \cdots + a_n g_n$	$H(f(w^0), \cdots, f(w^n))$	$(a_0 q^0 + \cdots + a_d q^d) g$

На рисунку 2.5 показано зв'язок між алгоритмами ZKP і наведеними вище схемами поліноміальних зобов'язань.

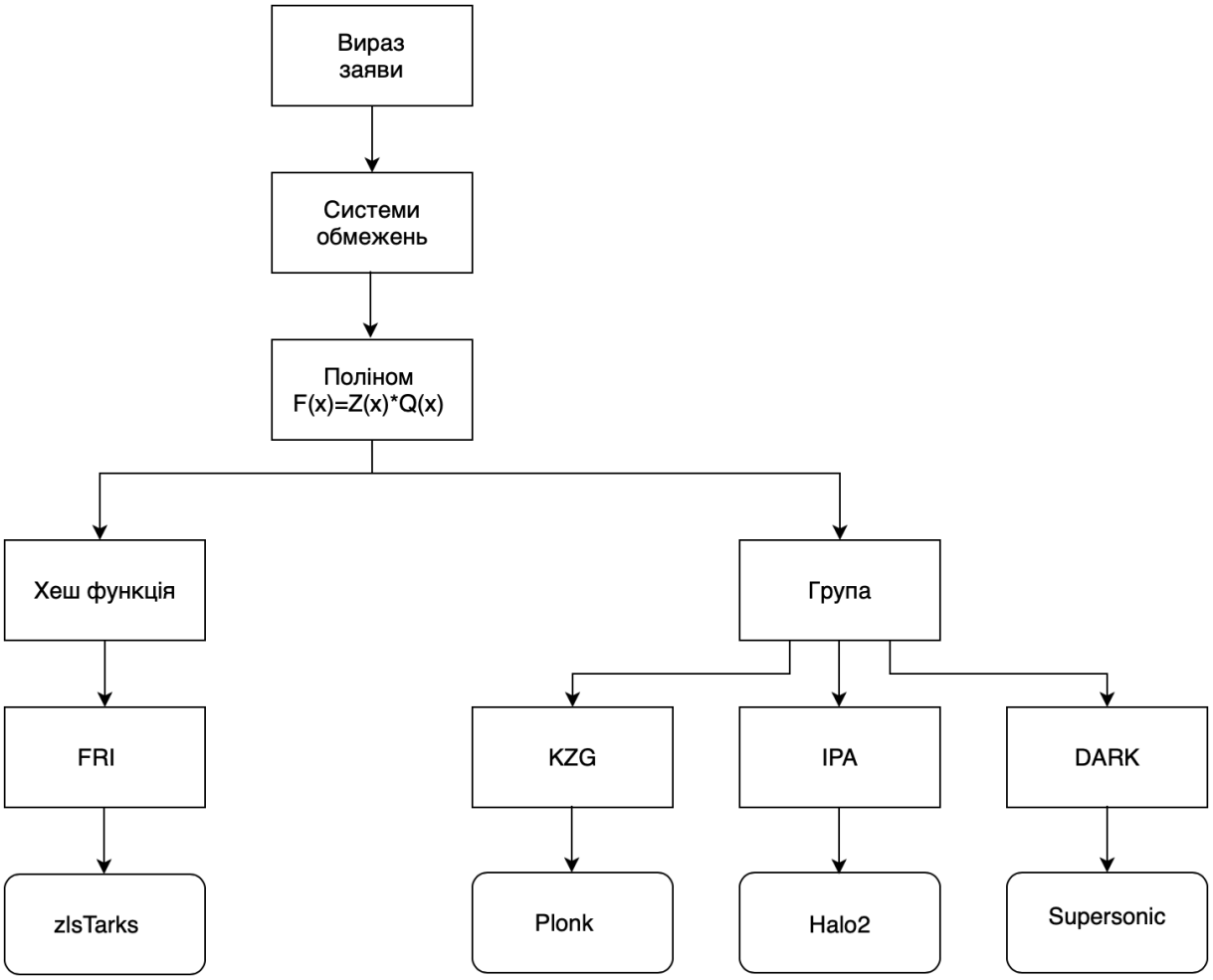

Рисунок 2.5 – Зв'язок між алгоритмами ZKP і схемами поліноміальних зобов'язань

Різні схеми поліноміальних зобов’язань призводять до різних алгоритмів ZKP з різними властивостями, ефективністю та безпекою. Наприклад, алгоритм ZK-STARKs на основі FRI є квантово стійким і не вимагає надійних налаштувань, оскільки він спирається на кілька математичних припущень безпеки. Крім того, для алгоритму Supersonic на основі DARK, якщо невідомою порядковою групою є група RSA, для нього потрібні надійні налаштування, оскільки він покладається на припущення проблеми цілочисельної факторизації (IFP); якщо невідома група порядку є групою класу, вона не потребує довірених налаштувань, оскільки вона покладається на складність обчислення кількості елементів групи класу. У

таблиці 2.2 наведено переваги та недоліки кожного поліноміального зобов’язання.

Таблиця 2.2 – Переваги та недоліки кожного поліноміального зобов’язання

	FRI	KZG	IPA	DARK (RSA)	DARK (Class)
Прозорий	так[1]	ні[3]	так[1]	ні[3]	так[1]
Стислий	так[2]	так[1]	ні[3]	так[2]	так[2]
Пост-квантовий	так[1]	ні[3]	ні[3]	ні[3]	ні[3]

де 1– відповідає відмінному; 2 – середньому; 3 – звичайному.

В алгоритмі ZKP стислий, як правило, означає: стисле доведення, стисла перевірка і стислий зв’язок. Тому у другому рядку наведеної таблиці 2.2 позначено плюси та мінуси кожного алгоритму відповідно до відмінностей у трьох наведених вище показниках. Далі розглянемо показники продуктивності, що відповідають кожному PC.

Таблиця 2.3 – Показники продуктивності поліноміального зобов’язання

	FRI	KZG	IPA	DARK
Доказ розміру	$O(\log_2(d))$	$O(1)$	$O(\log (d))$	$O(\log (d))$
Перевірка часу	$O(\log_2(d))$	$O(1)$	$O(\log (d))$	$O (d)$
Доведення часу	$O(d \cdot \log_2(d))$	$O(d)$	$O(d)$	$O (d)$

З таблиці 2.3 можна зробити висновок, що KZG є найкращою схемою з точки зору лаконічності, оскільки її розмір перевірки та час перевірки постійні, а це означає, що збільшення розміру схеми не призведе до збільшення розміру перевірки. Для інших схем поліноміального зобов’язання

розмір пов’язаний із масштабом схеми, що означає, що розмір буде збільшуватися зі збільшенням масштабу схеми.

Однак з точки зору безпеки схема KZG є слабшою порівняно з іншими схемами, оскільки для неї потрібне довірене налаштування третьої сторони (див. таблицю 2.1).

Проведено порівняння основних схем поліноміального зобов’язання з різних точок зору. Однак при реальному застосування розробник повинен провести комплексну оцінку сценаріїв застосування. При цьому, вища безпека або краща ефективність завжди є компромісом.

3 РОЗРОБКА ТА ТЕСТУВАННЯ СХЕМ ДОКАЗУ З НУЛЬОВИМ ЗНАННЯМ

3.1 Алгоритм побудови SNARK

Алгоритм побудови SNARK складається з наступних кроків [24].

1. Вибираємо кінцеве поле для деякого простого числа. Тобто, фіксуємо набір функцій.

2. Описуємо арифметичну схему (використовуючи вибране кінцеве поле). Використовуючи доменно-специфічну мову (DSL), наприклад Circom, Leo, Cairo, Noir...). Схема повинна враховувати публічну заяву та приватного свідка. Схема виводить деякі елементи в кінцеве поле. Арифметична схема має вентилі додавання та множення (mod p), і схема приймає як вхідні дані оператор x і свідок w. Таким чином ми можемо скомпілювати наш код у зручний для SNARK формат (R1CS, Plonk, AIR..). Тепер у нас є система аргументів.

3. Попереднья обробка схеми за допомогою алгоритму налаштування. Алгоритм налаштування приймає опис схеми як вхідні дані та виводить загальнодоступні параметри для Прувера та Верифікатора. Це дозволяє попередньо обробити схему. Тут використовується трансформація Фіата-Шаміра: це крок, який дозволяє мати справу з неінтерактивними системами доказів замість інтерактивних. Алгоритм налаштування виводить деякі глобальні параметри, які використовуються верифікатором і перевірником. Для неінтерактивних систем етап попередньої обробки необхідний, оскільки це дає змогу перевіряльнику просто надіслати підтвердження без будь-яких додаткових взаємодій.

4. Після того, як ми обираємо алгоритми Prover і Verifier, ми передаємо загальнодоступні параметри в серверну систему перевірки snark (наприклад, перевірка запускатиме щось на зразок Groth16, Plonk, Bulletproofs...).

5. Перевіряючий (Прувер, Prover) хоче довести, що він може обчислювати f(x) = y. Перевіряючий також хоче довести, що функція f

включена в певну область (набір функцій, який був вибраний раніше). Іншими словами, Перевіряючий хоче довести, що для схеми C і оператора s він знає свідка w, наприклад, що схема дорівнює 0, що коротко означає $C(s, w) = 0$. Свідок у цьому випадку є описом функції f разом із випадковістю r. Як правило, доказ досить короткий (наприклад, кілька КБ). Перше повідомлення в протоколі - це зобов'язання щодо полінома, надіслане перевірячем верифікатору. Щоб надіслати перше повідомлення, Prover вибирає функцію в наборі та фіксує її, запускаючи алгоритм фіксації. Алгоритм фіксації виводить рядок зобов’язання, який потім використовується як зобов’язання щодо глобальних параметрів, вибраної функції та випадковості. Це перше повідомлення важливе, оскільки воно визначає поліном, який перевірив. Після цього першого повідомлення починається інтерактивна процедура. Інтерактивна процедура відповідає протоколу оцінки, який дозволяє взаємодіяти між Прувером і верифікатором. У цій інтерактивній процедурі повідомлення від Прувера є короткими та читаються Верифікатором повністю. Ця процедура включає серію раундів, зокрема:

6.1) Верифікатор вибирає випадкову точку в скінченному полі та просить Прувера оцінити поліном у цій випадковій точці. Іншими словами, Верифікатор просить Прувера «відкрити» вибраний поліном у певній точці.

6.2) Прувер повинен оцінити поліном у точці, вибраній Верифікатор, щоб він міг довести, що він знає, як оцінити функцію, яку він взяв на себе. Прувер надсилає оцінку разом із доказом правильності оцінки.

6.3) Прувер фіксує інший поліном і надсилає зобов’язання верифікатору.

6.4) Верифікатор вибирає інше випадкове значення та надсилає його Пруверу.

6.5) Прувер обчислює поліном у новій точці та надає Верифікатору як нові оцінки, так і новий доказ того, що оцінка правильна.

7. Етапи пункту 6 повторюються протягом кількох раундів. Це безперервна взаємодія між перевіряльником і верифікатором. Оскільки використовується перетворення Фіата-Шаміра, виконавши попередню обробку, можна змоделювати цю взаємодію. Верифікатор може попросити Прувера відкрити будь-який із цих поліномів у будь-якій точці за вибором верифікатора.

8. Прувер фіксує останній поліном і надсилає фіксацію. Це означає, що прувер виконує функцію, щоб пізніше він міг створити стислий доказ того, що прийнята функція задовольняє $f(x) = y$. Прувер доводить, що знає свідка (мається на увазі опис функції та випадковість), такий що $f(x) = y$.

9. Після певної кількості взаємодій верифікатор вирішує, прийняти чи відхилити доказ. Верифікатор виконує цю перевірку, беручи як вхідні дані твердження, випадкові значення та оцінки поліномів у всіх вибраних ним точках.

Таким чином, щоб побудувати SNARK, ми параметризуємо поліноміальний IOP двома параметрами: кількість поліноміальних зобов'язань; кількість запитів на оцінку від алгоритму перевірки (це кількість точок, за якою прувер повинен відкрити оцінюваний поліном).

Остаточний SNARK складається з:

– Провер надсилає поліноміальні зобов’язання верифікатору.

– Верифікатор виконує оціночні запити кілька разів.

– Остаточний SNARK закінчується кількома поліноміальними зобов’язаннями + кількома доказами оцінки.

3.2 Структура доказу з нульовим знанням

Нещодавно набір криптографічних примітивів, які називаються доказами з нульовим знанням (ZKP), сколихнув світ публічних блокчейнів і розподілених реєстрів. Спочатку ZKP з’явилися як рішення проблем

конфіденційності, але останнім часом вони стали також ідеальним рішенням проблем масштабованості. У результаті ці криптографічні докази стали дуже привабливими інструментами для блокчейн-спільноти, а найефективніші алгоритми вже розгорнуто та інтегровано в деякі програми.

Основним типом ZKP, що використовується в контексті блокчейну, є zk-SNARK (або «короткі неінтерактивні аргументи знань з нульовим знанням»). Ці докази додають традиційним ЗКП 2 додаткові властивості: лаконічність та неінтерактивність. Лаконічність означає, що розмір перевірки невеликий, часто лише кілька сотень байт, і може бути швидко перевірений верифікатором. Неінтерактивність означає, що верифікатор і верифікатор не повинні взаємодіяти; самого доказу достатньо. Старіші ZKP вимагали перевірників і верифікаторів надсилати повідомлення туди й назад для створення доказу.

Стисліcть робить ZKP швидкими та дешевими для перевірки. Це робить їх чудовою технологією масштабування. У зведеннях валідності (тобто zkRollups) потужні верифікатори можуть обчислювати результати багатьох тисяч транзакцій, створювати стислі докази їх правильності та надсилати їх до базового ланцюжка. Там верифікатори можуть перевірити підтвердження та негайно прийняти результати всіх включених транзакцій, не обчислюючи їх самостійно. Таким чином, мережа може масштабуватися, в той час як базовий ланцюг залишається децентралізованою.

Об’єднання тисяч транзакцій в одне підтвердження вимагає величезної кількості обчислень і призводить до тривалого часу підтвердження. Довгий час перевірки створює більший час для завершення, оскільки транзакції користувачів не мають повної остаточності, доки їх транзакція та її підтвердження не будуть подані в базовий ланцюжок. Це може зайняти деякий час. Наприклад, у Starknet ми очікуємо, що спочатку час перевірки триватиме кілька годин. Доказ з нульовим знанням потрібно прискорити для кращої продуктивності, безпеки та UX.

Система доказу з нульовим знанням має три етапи: налаштування, створення доказів і перевірка доказів (рисунок 3.1) [24].

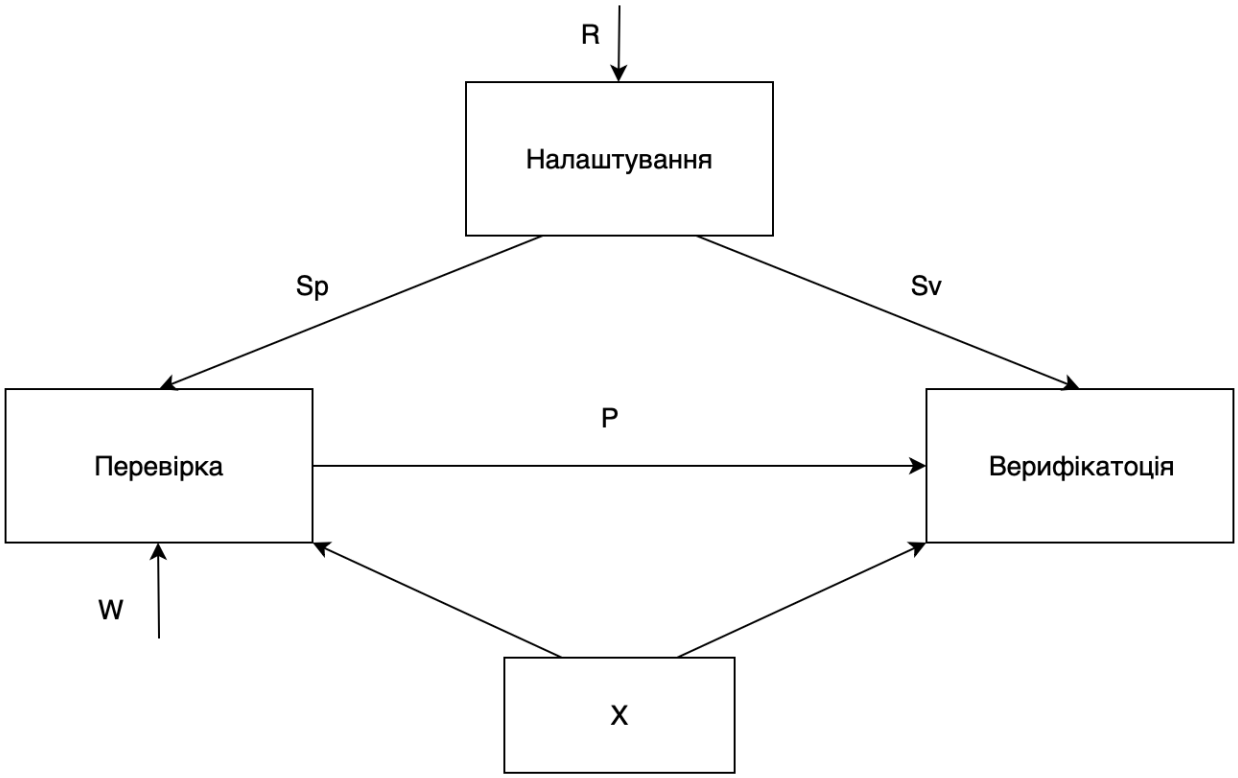

Рисунок 3.1 – Система доказу з нульовим знанням

У схемі доказу використовується 6 значень:

R – випадкове число: для налаштування системи ZKP необхідне одноразове таємне випадкове число. Якщо будь-яка сторона знає випадкове число, вона може зламати код і ідентифікувати секретне вхідне значення, усуваючи властивість нульового знання. Звідси походить ідея «довірених установок». У надійних налаштуваннях багато сторін збираються разом, щоб колективно генерувати випадкове число, щоб ніхто не дізнався секрет. Як користувач, ви повинні вірити, що налаштування виконано правильно (тобто ніхто не знає R), щоб ваша інформація залишалася конфіденційною. Зверніть увагу, що для STARK не потрібні надійні налаштування.

S_p – параметри налаштування перевіряльника: параметри, надані перевіряльнику після налаштування, які дозволяють йому згенерувати дійсний доказ.

S_v – параметри налаштування верифікатора: параметри, надані верифікатору після налаштування, які дозволяють йому перевірити дійсне підтвердження.

X – загальнодоступні вхідні значення: вхідні значення, які ми використовуємо для виконання обчислень. Вони надаються як перевіряючому, так і верифікуючому і не є секретом.

W – приватне вхідне значення: це таємне вхідне значення, яке називається свідком і надається лише перевіряючому. Зверніть увагу на діаграму вище, що свідок не передається верифікатору. Суть ЗК полягає в тому, що він дає змогу довести твердження про свідка без необхідності його розкривати.

P – доказ: це доказ, створений перевіряючим і надісланий верифікатору.

Отже, доказ із нульовим знанням – це протокол, який дозволяє одній стороні, яка хоче перевірити твердження, переконати іншу сторону (верифікатор), у тому, що твердження правдиве, не розкриваючи при цьому жодної інформації, окрім правдивості твердження. Наприклад, довідник може створювати докази для таких тверджень:

«Я знаю приватний ключ, який відповідає цьому відкритому ключу»: у цьому випадку підтвердження не розкриє жодної інформації про приватний ключ.

«Я знаю приватний ключ, який відповідає відкритому ключу з цього списку»: як і раніше, доказ не відкриває інформацію про приватний ключ, але в цьому випадку пов’язаний відкритий ключ також залишатиметься закритим.

«Я знаю прообраз цього хеш-значення»: у цьому випадку підтвердження покаже, що перевіряльник знає прообраз, але не розкриє жодної інформації про значення цього прообразу.

«Це хеш блоку блокчейну, який не створює від'ємного балансу»: у цьому випадку підтвердження не розкриє жодної інформації про суму, походження або призначення транзакцій, включених до блоку.

Неінтерактивні докази з нульовим знанням (NIZK) – це окремий тип доказів з нульовим знанням, у якому перевіряльник може створити доказ без взаємодії з верифікатором. Протоколи NIZK добре підходять для блокчейн-додатків Ethereum, оскільки дозволяють смарт-контрактам виконувати роль верифікатора. Таким чином, будь-хто може створити доказ і надіслати його як частину транзакції в смарт-контракт, який може виконувати певні дії залежно від того, чи є доказ дійсним чи ні.

У цьому контексті найбільш бажаними доказами NIZK є докази zk-SNARK (короткий неінтерактивний аргумент знань з нульовим знанням), набір неінтерактивних протоколів з нульовим знанням, які мають стислий розмір доказу та сублінійний час перевірки. Важливість цих протоколів подвійна: з одного боку, вони допомагають покращити гарантії конфіденційності, а з іншого, їхній невеликий розмір перевірки використовується в рішеннях для масштабування.

3.2.1Арифметичні схеми

Як і більшість ZKP, zk-SNARK дозволяють перевіряти обчислювальні твердження, але вони не можуть бути застосовані до обчислювальної задачі безпосередньо, спочатку потрібно перетворити твердження в правильну форму. Зокрема, zk-SNARK вимагають, щоб оператор обчислення моделювався за допомогою арифметичної схеми. Хоча не завжди очевидно, як виконати це перетворення, більшість обчислювальних задач можна легко перетворити на арифметичні схеми.

F(p) – арифметична схема це схема, що складається з набору проводів, які передають значення з поля F(p) і з'єднують їх з елементами додавання та множення за модулем p.

Необхідно пам'ятайти, що задане просте число p кінцевого поля F(p) складається з набору чисел $\{0,\ldots,p-1\}$, на які ми можемо додавати та множити ці числа за модулем p.

Наприклад, кінцеве поле F(7) складається з набору чисел $\{0,\ldots,6\}$, на які ми можемо додавати та множити числа за модулем 7. Простий приклад, як виконується операція за модулем – це годинник, який працює за модулем 12, нам не важливо, скільки разів стрілки годинника перейшли через 12, а лише те, який час вони показують. Іншими словами, нас цікавить лише залишок від ділення на 12.

Наприклад:

16 modulo 7 = 2, оскільки 16 = 7 + 7 + 2;

7 modulo 7 = 0;

3*6 modulo 7 = 4, оскільки 3*6 = 18 = 7+7 + 4.

3.2.2 Сигнали схеми

Отже, арифметична схема приймає деякі вхідні сигнали, які мають значення між $0,\ldots,p-1$, і виконує додавання та множення між ними за модулем простого p. Вихід кожного вентиля додавання та множення вважається проміжним сигналом, за винятком останнього вентиля схеми, вихід якого є вихідним сигналом схеми.

Щоб згенерувати та перевірити докази zk-SNARK в Ethereum, нам потрібно працювати з F(p) – арифметичними схемами, взявши просте p:

p = 21888242871839275222246405745257275088548364400416034343698204186575808495617

Просте число p є порядком скалярного поля кривої BN254 (також відомої як крива ALT_BN128), як визначено в EIP 196.

Circom 2.1.6 представляє два нових простих числа для роботи, а саме порядок скалярного поля BLS12-381.

p = 52435875175126190479447740508185965837690552500527637822603658699938581184513,

і просте число 18446744069414584321, яке спочатку використовувалося в Plonky2.

На рисунку 3.1 приведено арифметичну схему F(7), яка виконує операцію: $y = a * b + c$. Схема має 5 сигналів: сигнали a, b і c є вхідними сигналами, d є проміжним сигналом і вихідний сигнал є вихідним сигналом схеми.

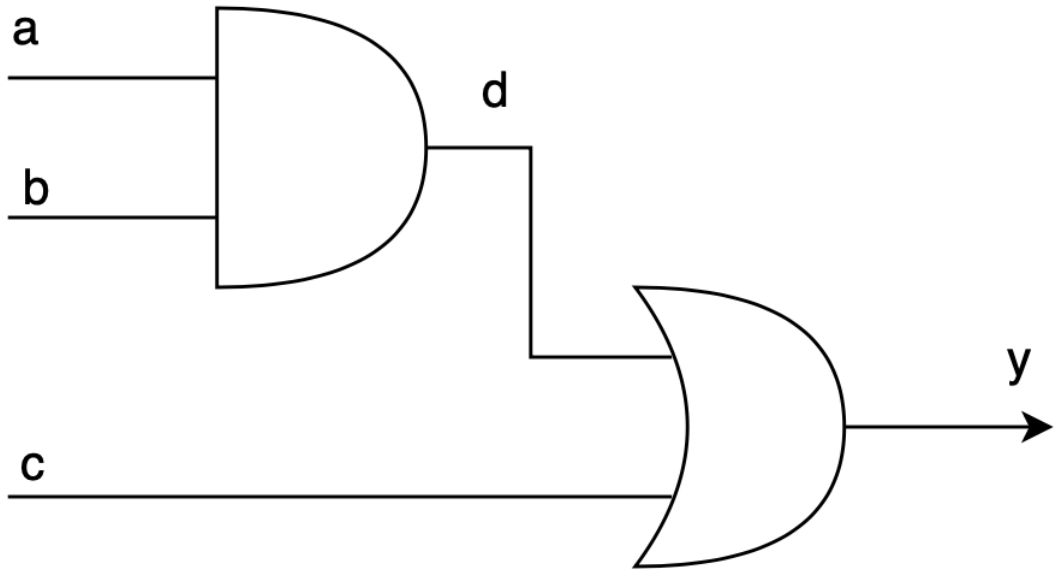

Рисунок 3.1 – Логічна схема

Щоб використовувати протоколи zk-SNARK, нам потрібно описати зв’язок між сигналами як систему рівнянь, які зв’язують змінні з вентилями. Відтепер рівняння, які описують схему, називатимуться обмеженнями, і ви можете думати про них як про умови, яким мають задовольняти сигнали цієї схеми.

3.2.3 Система обмежень рангу 1

Якщо у нас є арифметична схема з сигналами $s\_1, \ldots, s\_n$, тоді ми визначаємо обмеження як рівняння наступної форми:

$$(a\_1 * s\_1 + \ldots + a\_n * s\_n) * (b\_1 * s\_1 + \ldots + b\_n * s\_n) + (c\_1 * s\_1 + \ldots + c\_n * s\_n) = 0.$$

Однак, треба враховувати, що обмеження мають бути квадратичними, лінійними або постійними рівняннями, і іноді, вносячи невеликі модифікації (наприклад, змінюючи змінну або збираючи два обмеження), можна зменшити кількість обмежень або змінних. Загалом схеми матимуть кілька обмежень (як правило, по одному на мультиплікативний вентиль). Набір обмежень, що описує схему, називається системою обмежень рангу 1 (R1CS):

$$(a\_11 * s\_1 + \ldots + a\_1n * s\_n) * (b\_11 * s\_1 + \ldots + b\_1n * s\_n) + (c\_11 * s\_1 + \ldots + c\_1n * s\_n) = 0.$$

$$(a\_21 * s\_1 + \ldots + a\_2n * s\_n) * (b\_21 * s\_1 + \ldots + b\_2n * s\_n) + (c\_21 * s\_1 + \ldots + c\_1n * s\_n) = 0.$$

$$(a\_31 * s\_1 + \ldots + a\_3n * s\_n) * (b\_31 * s\_1 + \ldots + b\_3n * s\_n) + (c\_31 * s\_1 + \ldots + c\_1n * s\_n) = 0.$$

...

...

$$(\mathrm{a\_m1} * \mathrm{s\_1} + \ldots + \mathrm{a\_mn} * \mathrm{s\_n}) * (\mathrm{b\_m1} * \mathrm{s\_1} + \ldots + \mathrm{b\_mn} * \mathrm{s\_n}) + (\mathrm{c\_m1} * \mathrm{s\_1} + \ldots + \mathrm{c\_mn} * \mathrm{s\_n}) = 0.$$

Операції всередині схеми виконуються за модулем певного простого p. Отже, усі рівняння вище визначені за модулем p.

У попередньому прикладі R1CS нашої схеми складається з наступних двох рівнянь:

$$d = a * b \; modulo \; 7$$
$$out = d + c \; modulo \; 7$$

У даному випадку, замінивши безпосередньо змінну d, ми можемо зібрати два рівняння в одне:

$$out = a * b + c \; modulo \; 7.$$

Приємна річ у схемах полягає в тому, що хоча більшість протоколів з нульовим знанням мають притаманну складність, яка може бути надзвичайною для багатьох розробників, дизайн арифметичних схем є чітким і акуратним.

За допомогою Circom можна розробляти власні схеми з власними обмеженнями, а компілятор виводить представлення R1CS, яке знадобиться для доказу нульового знання.

Нульове знання дозволяє підтвердити задовільність схеми. Це означає, що ви можете довести, що знаєте набір сигналів, які задовольняють схему, або іншими словами, що ви знаєте рішення для R1CS. Цей набір сигналів називається свідком.

3.2.4 Свідок

Враховуючи набір вхідних даних, розрахунок проміжних і вихідних сигналів є досить простим. Отже, маючи будь-який набір вхідних даних, ми завжди можемо обчислити решту сигналів. Отже, чому ми повинні говорити про задовільність схеми? Ключовим аспектом доказів із нульовим знанням є те, що вони дозволяють обчислювати ці схеми, не розкриваючи інформацію про сигнали.

Наприклад, представимо, що в попередній схемі вхід a є закритим ключем, а вхід b є відповідним відкритим ключем. Ви можете погодитися з розкриттям b, але ви точно не хочете розкривати a. Якщо ми визначимо a як приватний вхід, b, c як загальнодоступні вхідні дані та out як загальнодоступний вихід, з нульовим знанням ми зможемо довести, не розкриваючи його значення, що ми знаємо приватний вхідний сигнал a, який для певних публічних значення $b,\ c$ і y є розв'язком рівняння:

$$a * b + c = y\ (mod\ 7).$$

При цьому, ми могли б легко вивести значення a, відокремивши його від інших сигналів. Важливо розробити схеми, які зберігають конфіденційність приватних входів і запобігають їх виведенню з R1CS.

Присвоєння сигналів називається свідком.

Наприклад, $\{a = 2, b = 6, c = -1,\ y = 4\}$ буде дійсним свідком для схеми.

Присвоєння $\{a = 1, b = 2, c = 1,\ y = 0\}$ не буде дійсним свідком, оскільки воно не задовольняє рівняння $a * b - c = y\ (mod\ 7)$.

Отже, докази zk-SNARK – це певний тип доказів з нульовим знанням, які дозволяють вам довести, що ви знаєте набір сигналів (свідків), які відповідають усім обмеженням схеми, не розкриваючи жодних сигналів, окрім загальнодоступних входів і виходи.

3.3 Встановлення екосистеми circom

Першим етапом створення доказу є написання програми на мові Circom. Наступним кроком треба згенерувати з програми доказ нульового знання. Для цього використовується компілятор Circom для компіляції обмежень C у систему обмежень рангу 1 (R1CS), яка, по суті, є набором

поліноміальних рівнянь 2-го ступеня над кінцевим полем. Оскільки компілятор Circom накладає обмеження на те, як можна використовувати === і ==, отримані обмеження відносно легко конвертувати в R1CS. Після отримання обмеження R1CS, використовуємо генератор zk-SNARK, такий як (snarkjs), щоб створити прувер P і верифікатор V. Цей процес показаний на рисунку 3.2 [25].

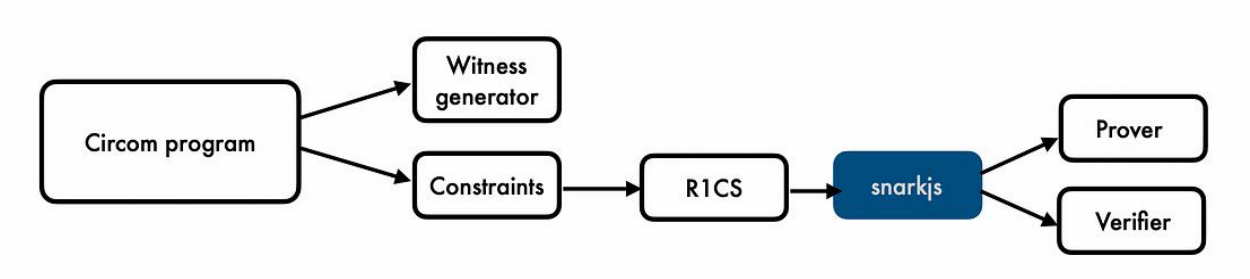

Рисунок 3.2– Процес компіляції програми Circom до доказу з нульовим знанням

Встановлення залежностей. Щоб запустити circom і пов'язані з ним інструменти, у системі потрібно встановити ряд залежностей (рисунок 3.2).

Основним інструментом є компілятор circom, написаний мовою Rust. Щоб у операційній системі був доступний Rust, необхідно встановити rustup. Для Linux або macOS, у терміналі вводимо таку команду:

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

Також має бути доступний менеджер пакетів, наприклад, npm або yarn. Останні версії Node.js включають підтримку великих цілих чисел і компілятори веб-складання, які допомагають виконувати код швидше, тому, щоб отримати кращу продуктивність, необхідно встановити версію 10 або новішу.

Встановлення circom. Щоб встановити circom необхідно клонувати репозиторій circom:

```
git clone https://github.com/iden3/circom.git
```

Переходимо до каталогу circom і використаємо збірку cargo для компіляції:

```
cargo build –release
```

Встановлення займає приблизно 3 хвилини. Коли команда успішно завершується, вона генерує двійковий файл circom у каталозі target/release.

Для встановлення цього бінарного файлу необхідно виконати команду:

```
cargo install --path circom
```

Попередня команда встановить двійковий файл circom у каталозі $HOME/.cargo/bin

Тепер можна побачити всі параметри виконуваного файлу за допомогою команди довідки:

```
circom –help
```

```
circom [input source circuit file] -r [output r1cs file] -c [output c file] -w
[output wasm file] -t [output wat file] -s [output sym file]

Options:
 --version      Show version number
[boolean]
 -h, --help     Show help
[boolean]
 --verbose, -v  Run with verbose logging      [boolean]
 --fast, -f     Do not optimize constraints       [boolean]
Copyright (C) 2018  0kims association
```

This program comes with ABSOLUTELY NO WARRANTY;
This is free software, and you are welcome to redistribute it
under certain conditions; see the COPYING file in the official
repo directory at https://github.com/iden3/circom

Встановлення snarkjs. snarkjs – це пакет npm, який містить код для створення та перевірки доказів ZK з артефактів, створених circom.

Встановити snarkjs можна за допомогою команди:

```
npm install -g snarkjs
```

Розглянемо приклад створення доказу з нульовим знанням за допомогою zkSnarks.

Першою частиною створення доказу з нульовим знанням у zkSnarks є розбиття функції на логічні кроки. Це передбачає створення арифметичної схеми, яка буде складатися з основних арифметичних операцій додавання, віднімання, множення та ділення. У даному прикладі використаємо рівняння:

$$d = a^2 + b$$

де a і b – приватні вхідні значення. Потім ми хочемо довести, що знаємо результат рівняння для вхідних даних a і b, фактично не відкриваючи a або b. Наприклад, якщо a=3 і b=11, відповідь буде такою:

$$d = 3 \times 3 + 11 = 20. \quad (3.1)$$

В даному прикладі доказом буде те, що ми знаємо, що $d = 20$, коли $a = 3$ і $b = 11$ (але не видаючи ці значення).

Опишемо рівняння (3.1) мовою circom [24]:

```
pragma circom 2.0.0;

template Mult() {
    signal input a;
    signal input b;

    signal output d;
    var c=0;

    c = a*a;
    d <== c+b;
 }

 component main = Mult();
```

та збережемо дану програму у файл з назвою mult.circom.

Арифметична схема побудована за рівнянням (3.1) приведена на рисунку 3.3:

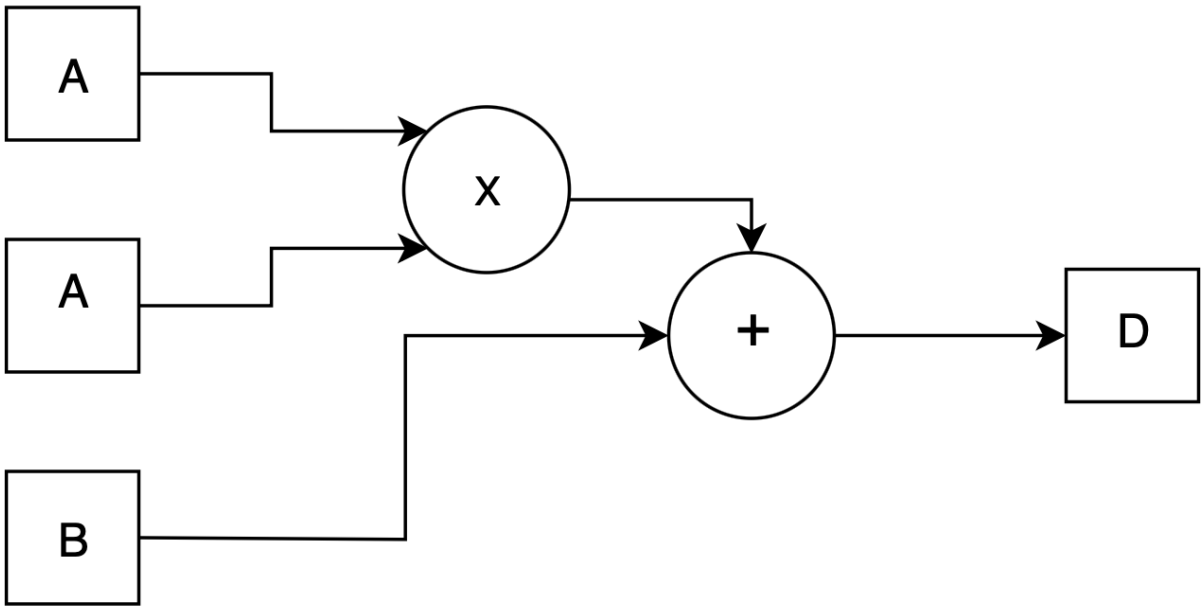

Рисунок 3.3 – Арифметична схема

Для того щоб скомпілювати схему виконаємо команду:

```
circom mult.circom --r1cs --wasm --sym --c
```

Після успішної компіляції файлу в каталозі проєкту буде створено ряд файлів (рисунок 3.4):

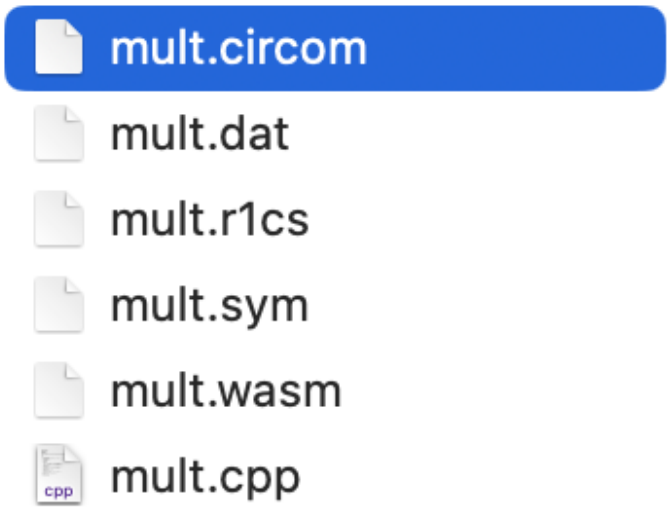

Рисунок 3.4 – Структура каталогу проєкту

В результаті компіляції буде створений файл R1CS (система обмежень рангу 1) (з розширенням -r1cs), а також папку з файлами generate_witness.js, mult.wasm і svjedok_calculator.js (з розширенням – wasm). Формат R1CS використовується для представлення всіх сигналів у схемі та для того, щоб їх можна було перевірити на доказ – це відоме як програма квадратичної арифметики (QAP).

Далі необхідно створити вхідний файл (input.json), щоб визначити доказ, для цього виконуємо команду:

```
echo " {"a": 3, "b": 11}” > input.json
```

Експортуємо файл R1CS у формат JSON за допомогою команди:

`circom1 % snarkjs r1cs export json mult.r1cs mult.json`.

В результаті буде створений файл mult.json з наступним текстом:

```
{
 "n8": 32,
 "prime": "21888242871839275222246405745257275088548364400416034343698204186575808495617",
 "nVars": 4,
 "nOutputs": 1,
 "nPubInputs": 2,
 "nPrvInputs": 0,
 "nLabels": 4,
 "nConstraints": 1,
 "useCustomGates": false,
 "constraints": [
  [
   {
    "2": "21888242871839275222246405745257275088548364400416034343698204186575808495616"
   },
   {
    "2": "1"
   },
   {
    "1": "21888242871839275222246405745257275088548364400416034343698204186575808495616",
    "3": "1"
   }
  ]
 ],
 "map": [
  0,
  3,
  1,
  2
 ],
 "customGates": [
 ],
 "customGatesUses": [
 ]
}
```


Наступним кроком, треба створити свідка для всіх значень у схемі, для заданих входів. Для цього необхідно зайти в папку mult_js і створити файл-свідок:

```
node generate_witness.js mult.wasm input.json witness.wtns
```

Після перерахунку witness json, отримаємо:


```
[
 "1",
 "20",
 "3",
 "11"
]
```


Тепер переходимо до кроку “powers of tau”:

```
snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
```

```
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS:

Blank Contribution Hash:
786a02f7 42015903 c6c6fd85 2552d272
912f4740 e1584761 8a86e217 f71f5419
d25e1031 afee5853 13896444 934eb04b
903a685b 1448b755 d56f701a fe9be2ce
[INFO]  snarkJS: First Contribution Hash:
9e63a5f6 2b96538d aaed2372 481920d1
a40b9195 9ea38ef9 f5f6a303 3b886516
0710d067 c09d0961 5f928ea5 17bcdf49
ad75abd2 c8340b40 0e3b18e9 68b4ffef
```

Дана команда створює файл під назвою pot12_0000.ptau. Створена схема буде застосована на наступному етапі:

```
snarkjs powersoftau prepare phase2 pot12_0001.ptau pot12_final.ptau -v
```

Тепер буде створено ключ перевірки та перевірки, які створюються у файлі zkey:

```
snarkjs groth16 setup mult.r1cs pot12_final.ptau mult_0000.zkey
```

Далі створюємо ключі:

```
snarkjs groth16 setup mult.r1cs pot12_final.ptau mult_0000.zkey
```

Для експортування ключа підтвердження використаємо команду:

```
snarkjs zkey export verificationkey mult_0001.zkey verification_key.json
```

Пробний запуск дає:

```
go-circom-prover-verifier
```

```
zkSNARK Groth16 prover
true
```

3.4 Розробка доказу з нульовим знанням на основі хеш-значення

Припустимо, що потрібно довести знання певної дати, наприклад чийсь день народження, не розкриваючи дату у форматі ДД::ММ::РР.

Однак, отримати значення d, m і y із публічних параметрів a, b і c досить просто. Щоб знайти d, m і y, потрібно лише розв'язати унікальні значення a, b і c, склавши рівняння:

```
a = d*y
b = m*y
c = d*m

=>  m = sqrt((bc)/a)
...
```

Тому кожен спостерігач може легко отримати d, m і y (які, як передбачається, є секретами, розділеними лише між prover і verifier).

Цю задачу можна вирішити, включивши односторонню (хеш-функцію), яка зробить обчислювально складним пошук d, m і y:

$$Hash = h(x),$$

де функція h(x) є односторонньою хеш-функцією, як, наприклад, SHA256. Тепер значення h(x) буде відкритим, але неможливо буде вивести x, знаючи лише h(x).

Iden3 circom надає стандартну бібліотеку, яка включає реалізацію хеш-функції SHA256 мовою circom [26].

Отже, circom SHA256 виконує математичні операції над бітовим масивом, тобто потрібно вказати вхідні дані, а також розмір вихідних даних.

Підтвердження хешу SHA256. Тепер розробимо схему, яка доводить, що ми знаємо деякий цілочисельний масив «date» з хешем SHA256(date):

```
pragma circom 2.0.0;
include "./circomlib/circuits/sha256/sha256.circom";
template Birthday(){
  component SHA = Sha256(6);
  signal input date[6];
  SHA.in <== date;

  signal output date_out[256];
  date_out <== SHA.out;
}

component main = Birthday();
```

Тепер, якщо ми хочемо довести, що знаємо день народження 13.03.2005, наш файл input.json виглядатиме так:

```
{"date":["1","5", "0", "3", "0", "5"]}
```

Ми могли б просто надіслати хеш і показати з високою ймовірністю, що знаємо значення «date».

Однак без доказу, доданого до обчислення, можна було б просто випадково вгадати хеш, і не було б доказу походження для процесу його обчислення. Це стане проблемою, коли кілька сторін надсилатимуть свої відповіді незалежно, оскільки лише одна з них повинна буде знати «date», а інші зможуть скопіювати та надіслати загальнодоступний параметр SHA256(date).

За допомогою zk-proof ми можемо довести, що знаємо деяке значення «date», хеш якого дорівнює SHA256(date), замість того, щоб просто доводити, що ми знаємо хеш SHA256(date).

Якщо припустити, що діапазон можливих значень для дати становить 01.01.1900–01.01.2000, існує приблизно $365 \times 100 = 36500$ можливих комбінацій. Тому шанс правильно вгадати дату з n спроб становить n/36500.

У реальній програмі zk це не вважатиметься безпечним, оскільки зазвичай використовується поріг для правильного вгадування секрету

$$n/2^{128} < n/36500.$$

Обмеження Powersoftau. Для наведеного прикладу «Перевірка хешу SHA256» використано файл, виходячи з його розміру приблизно 30 000 обмежень [25].

Після кроку powersoftau і створення доказу 256-бітний вихід SHA256 можна знайти в файлі public.json.

Щоб згенерувати доказ обчислення, виконуємо інструкцій із попереднього прикладу для ілюстрації роботи із circom, але замість генерації pot12_final.ptau використовуємо попередньо завантажений файл ptau [26].

Якщо запуск перевірки пройшов успішно, появиться повідомлення::ŕ

[INFO] snarkJS: OK!

Це означає, що підтвердження для SHA256(date) успішно створено.

Обмеження та правильне використання. На етапі надійного налаштування рекомендується, щоб багато сторін брали участь у створенні ключа підтвердження. Це необхідно, оскільки circom покладається на високий ступінь випадковості через той факт, що той самий ключ перевірки використовується знову і знову для однієї схеми, як для створення, так і для перевірки доказів. Одного чесного вводу на цьому етапі налаштування достатньо, щоб забезпечити криптографічний захист.

Circom простий у використанні для розробників, а бібліотека circom дозволяє кожному, незалежно від рівня знань у математиці, реалізувати складні схеми.

ВИСНОВКИ

В кваліфікаційній роботі розв’язано актуальну задачу розробки алгоритмів та схеми доказу з нульовим знанням для захисту конфіденційності даних. При цьому отримано наступні результати.

1. Проведено аналіз основні властивості доказу з нульовим знанням. Було детально розглянуто методи інтерактивного та неінтерактивного доказу з нульовим знанням, їхні переваги та обмеження.

2. Порівняння схем поліноміальних зобов'язань виявило, якість кожної з них та ступінь їхньої придатності для практичного використання. Це дало можливість визначити найефективніші рішення в конкретних умовах.

3. Аналіз різних методів доказу з нульовим знанням дозволив виокремити їхні переваги та недоліки в різних сценаріях застосування. Це сприяло більш глибокому розумінню та вибору оптимального методу в залежності від вимог та умов.

4. Розроблено алгоритм побудови SNARK, який дозволяє реалізувати ефективні схеми доказу з нульовим знанням. Це має важливе значення для захисту конфіденційності в різних сферах, від кібербезпеки до фінансових технологій.

5. Розроблено схему доказу з нульовим знанням на основі хеш-значень, яка підвищує надійність перевірки дати без розкриття жодної додаткової інформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sun, X., Yu, F. R., Zhang, P., Sun, Z., Xie, W., & Peng, X. A survey on zero-knowledge proof in blockchain. *IEEE network*, 2021, *35*(4), – pp. 198-205.

2. Morais, E., Koens, T., Van Wijk, C., Koren, A. A survey on zero knowledge range proofs and applications. *SN Applied Sciences*, 2019, *1*, – pp.1-17.

3. Mohr, A. A survey of zero-knowledge proofs with applications to cryptography. *Southern Illinois University, Carbondale*, 2007 –pp.1-12.

4. Yin, W. Zero-Knowledge Proof Intelligent Recommendation System to Protect Students' Data Privacy in the Digital Age. *Applied Artificial Intelligence*, 2023б *37*(1), 2222495.

5. Understanding Zero-knowledge proofs through illustrated examples. Електронний ресурс]. - Режим доступу: https://blog.goodaudience.com/understanding-zero-knowledge-proofs-through-simple-examples-df673f796d99

6. Quadratic Arithmetic Programs: from Zero to Hero. Електронний ресурс]. - Режим доступу: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649

7. Esgin, M. F., Steinfeld, R., Liu, J. K., & Liu, D. (2019, August). Lattice-based zero-knowledge proofs: new techniques for shorter and faster constructions and applications. In *Annual International Cryptology Conference* (pp. 115-146).

8. Bootle, J., Lyubashevsky, V., & Seiler, G. (2019, August). Algebraic techniques for short (er) exact lattice-based zero-knowledge proofs. In *Annual International Cryptology Conference*, – pp. 176-202.

9. Aranha, D. F., El Housni, Y., & Guillevic, A. A survey of elliptic curves for proof systems. *Designs, Codes and Cryptography*, 2023, *91*(11), 3333-3378.

10. Fiege, U., Fiat, A., & Shamir, A. Zero knowledge proofs of identity. In Proceedings of the nineteenth annual ACM symposium on Theory of computing, 1987, – pp. 210-217

11. Що таке Оракули (Oracles)? Електронний ресурс]. - Режим доступу: https://medium.com/@LOFT.education/%D1%89%D0%BE-

%D1%82%D0%B0%D0%BA%D0%B5-%D0%BE%D1%80%D0%B0%D0%BA%D1%83%D0%BB%D0%B8-oracles-6fea6e7f6ac0

12. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. Електронний ресурс]. - Режим доступу: https://link.springer.com/content/pdf/10.1007/3-540-47721-7_12.pdf

13. Quadratic Arithmetic Programs: from Zero to Hero. Електронний ресурс]. - Режим доступу: https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649

14. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., & Virza, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, 2013, – pp. 90-108.

15. Lipmaa, H. Prover-efficient commit-and-prove zero-knowledge SNARKs. In *Progress in Cryptology–AFRICACRYPT 2016: 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings 8*, 2016, – pp. 185-206.

16. Campanelli, M., Fiore, D., & Querol, A. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, – pp. 2075-2092.

17. Pinto, A. M. An introduction to the use of zk-SNARKs in blockchains. In *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*, –2020, pp. 233-249.

18. Partala, J., Nguyen, T. H., & Pirttikangas, S. Non-interactive zero-knowledge for blockchain: A survey. *IEEE Access*, *8*, 2020, – pp. 227945-227961.

19. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., & Wang, Q. Horst meets fluid-SPN: griffin for zero-knowledge applications. In Annual International Cryptology Conference, – 2023, – pp. 573-606.

20. Kate, A., Zaverucha, G. M., & Goldberg, I. Constant-size commitments to polynomials and their applications. In Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16, pp. 177-194

21. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., & Maxwell, G. Bulletproofs: Short proofs for confidential transactions and more. In 2018 IEEE symposium on security and privacy (SP), 2018. – pp. 315-334

22. Ben-Sasson, E., Bentov, I., Horesh, Y., & Riabzev, M. (2018). Fast reed-solomon interactive oracle proofs of proximity. In 45th International colloquium on automata, languages, and programming (icalp 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. [Електронний ресурс]. - Режим доступу: https://drops.dagstuhl.de/storage/00lipics/lipics-vol107-icalp2018/LIPIcs.ICALP.2018.14/LIPIcs.ICALP.2018.14.pdf

23. Bünz, B., Fisch, B., & Szepieniec, A. Transparent SNARKs from DARK compilers. In Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39, 2020, – pp. 677-706.

24. Circom 2 Documentation. [Електронний ресурс]. - Режим доступу: https://docs.circom.io/circom-language/signals/]

25 Powers of Tau. [Електронний ресурс]. - Режим доступу: https://hermez.s3-eu-west-1.amazonaws.com/powersOfTau28_hez_final_15.ptau

26. Неалізацію хеш-функції SHA256 мовою circom. Електронний ресурс]. - Режим доступу: https://github.com/iden3/circomlib/blob/master/circuits/sha256/sha256.circom