Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Порівняльний аналіз та дослідження еліптичних кривих та схем генерації доказів із нульовим розголошенням у рамках протоколу zkSNARK
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-3

Прядко В.С.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Тип програми освітньо-наукова

Керівник доц. Голян Н.В.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Прядко Владиславу Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Порівняльний аналіз та дослідження еліптичних кривих та схем генерації доказів із нульовим розголошенням у рамках протоколу zkSNARK»
Затверджена наказом по університету від 29.03.2024р. № 250 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 20.06.2024
3. Вихідні дані до роботи: стандарт протоколу zkSNARK, бібліотека ZoKrates, методології порівняння та збору метрик обчислень, платформа Node.js, мови програмування TypeScript, Python та допоміжна DSL Zokrates, середовище розробки VSCode.
4. Перелік питань, що потрібно опрацювати в роботі: аналіз та порівняння існуючих схем, кривих та бекендів в рамках протоколу zkSnark; вимірювання та естимація головних метрик роботи із QAP; вибір комбінації режимів враховуючи передбачені метрики метрики; написання програмних рішень; проведення експериментів та аналіз отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз предметної області	25.03.2024	*виконано*
2	Постановка задачі	10.04.2024	*виконано*
3	Проведення дослідження	01.05.2024	*виконано*
4	Підготовка пояснювальної записки	15.05.2024	*виконано*
5	Підготовка презентації та доповіді	18.05.2024	*виконано*
7	Перевірка на академічний плагіат	12.06.2024	*виконано*
8	Нормоконтроль	12.06.2024	*виконано*
9	Рецензування	13.06.2024	*виконано*
10	Знесення диплома в електронний архів	13.06.2024	*виконано*
11	Допуск до захисту у зав. кафедри	14.06.2024	*виконано*

Дата видачі завдання: 25.03.2024 р.

Студент (ка) _____ Прядко В.С.
(підпис)

Керівник роботи _____ доц. Голян Н.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 55 с., 7 рис., 2 табл., 18 джерел, 5 додатків.

ZK-SNARK, БЛОКЧЕЙН, ДОКАЗИ ІЗ НУЛЬВИМ РОЗГОЛОШЕННЯМ, ЕЛІПТИЧНА КРИПТОГРАФІЯ, КВАДРАТИЧНІ АРИФМЕТИЧНІ ПРОГРАМИ, ПРОТОКОЛ.

Об’єктом дослідження є протокол генерації доказів із нульовим розголошенням zkSNARK.

Мета роботи становить собою дослідження сфери доказів із нульовим розголошенням, протоколу zkSNARK та його схемам генерації, практичне дослідження схем доказу в протоколі, вимірювання метрик етапів роботи із різними комбінаціями параметрів.

Результатом роботи є заключення про оптимальний вибір комбінації схеми, еліптичної кривої та бекенду для ефективного впровадження протоколу у системи зі складними обчислювальними задачами.

ZK-SNARK, BLOCKCHAIN, ZERO-KNOWLEDGE PROOFS, ELLIPTIC CRYPTOGRAPHY, QUADRATIC ARITHMETIC PROGRAMS, PROTOCOL.

The object of study is the protocol for generating zero-disclosure proofs zkSNARK.

The purpose of the study is to investigate the field of zero-knowledge proofs, the zkSNARK protocol and its generation schemes, to conduct a practical study of the proof schemes in the protocol, and to measure the metrics of the stages of work with different combinations of parameters.

The result of the work is a conclusion about the optimal choice of the combination of scheme, elliptic curve, and backend for the effective implementation of the protocol in systems with complex computational tasks.

Я, Прядко Владислав Сергійович, студент гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Порівняльний аналіз та дослідження еліптичних кривих та схем генерації доказів із нульовим розголошенням у рамках протоколу zkSNARK», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ZKP(s) – Zero Knowledge Proof(s)

QAP – Quadratic Arithmetic Program

SNARK - Succinct Non-Interactive Argument of Knowledge

r1cs - Rank-One Constraint System

DSL – Domain Specific Language

ВСТУП

У сучасному світі стрімкого технологічного прогресу з'являються нові виклики та можливості для забезпечення конфіденційності та безпеки даних. Особливу увагу привертають протоколи доказів із нульовим розголошенням, зокрема zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge), які дозволяють підтвердити певну інформацію без розкриття її змісту. Це створює нові можливості для практичного застосування та водночас ставить перед дослідниками завдання щодо їх удосконалення.

Робота зосереджена на практичному аналізі та порівнянні різних схем генерації доказів у протоколах zk-SNARK з метою оцінки їх ефективності, надійності та придатності для використання в різних сферах. Висновки роботи зможуть допомогти інтеграції протоколу в таких обчислювально об’ємних сценаріях, як докази отримання тендеру, підтвердження цільового використання коштів та запобігання корупції в оборонних підприємствах. Ці приклади відображають важливість застосування zk-SNARK у реальному світі, де прозорість та конфіденційність є критично важливими.

Спеціально уникаємо фокусування на блокчейн або криптовалютах, прагнучи продемонструвати ширші можливості використання цих технологій. Натомість, увага зосереджена на реальних практичних сценаріях, які відкривають нові горизонти для застосування доказів із нульовим розголошенням. Це дослідження має на меті розширити розуміння потенціалу протоколів zk-SNARK та внести свій вклад у розвиток концепцій конфіденційності та безпеки в інформаційних технологіях.

Таким чином, дослідження є важливим кроком на шляху до більш прозорого та безпечного світу, де технології конфіденційності можуть бути застосовані в найрізноманітніших галузях.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі дослідження

zkSNARKs [1] дозволяють створювати короткі докази без необхідності взаємодії між актором, що доводить та перевіряючим. Вони використовують квадратичні арифметичні програми (QAP) [2] для перетворення верифікації обчислень на перевірку поліномів, з використанням еліптичних кривих для генерації компактних доказів. "Початкове налаштування" [3] у zkSNARKs вимагає генерації секретного ключа, який потім використовується для створення публічного ключа верифікації. Це налаштування вимагає високого рівня довіри до виконавців процесу, оскільки компрометація може загрожувати всій системі, але забезпечує високу ефективність обчислень. Використання zkSNARKs вже було реалізовано в ряді відомих проектів, зокрема в криптовалюті Zcash [4], де вони використовуються для забезпечення приватності транзакцій. Попри свої переваги, такі як ефективність і конфіденційність, zkSNARKs все ще мають певні обмеження, зокрема в контексті масштабування та потреби в "початковому налаштуванні", що вимагає подальших досліджень та розвитку для усунення цих викликів. Хоч існують і інші способи прискорення оффчейн криптографічних операцій [5,6], докази із нульовим розголошенням все ще є незамінним інструментом при ончейн-обчисленнях, оптимізація яких не є легкою задачею.

1.2 Постановка задачі

1.2.1 Визначення класу даних для проведення теоретичного та практичного експериментів

Дані для задачі формального доказу із нульовим розголошенням зазвичай мають просту структуру і невеликий обсяг, часто це число або значення типу "так" чи "ні". У такому випадку, метрики схем доказу не завжди релевантні для нестандартних класів даних. Оскільки дослідження спрямоване на інтеграцію ZKp-протоколів у сучасні соціально-прикладні задачі, необхідно визначити оптимальну схему для цього класу даних.

Класичні дані для zkSNARK зазвичай охоплюють прості числові значення або логічні вирази. Ці дані часто можуть бути відкрито подані без ризику витоку конфіденційної інформації, що дозволяє забезпечувати прозорість і точність у таких застосуваннях, як фінансові додатки та блокчейн-транзакції.

З іншого боку, дані у прикладних задачах державного або корпоративного рівня є значно складнішими. Вони включають багатовимірні набори даних, які часто мають високий ступінь конфіденційності. Наприклад, це можуть бути дані про об'єми виробництва оборонних виробів або деталі про тендери, які не повинні бути відомі публіці чи неуповноваженим особам. Використання таких даних актуальне для урядових, оборонних та соціально-орієнтованих систем, де важливо забезпечити конфіденційність і безпеку критично важливої інформації. Таким чином, при інтеграції zkSNARK-протоколів у ці системи необхідно враховувати їхню складність і високий рівень конфіденційності.

1.2.2 Теоретичний вибір схем генерації доказу

Для стандартних даних оптимальна схема вибирається за допомогою лінійної адитивної згортки з ваговими коефіцієнтами. Протокол zkSNARK може бути побудований на різних частинах, таких як еліптична крива, схема генерації доказу та r1cs бекенд [7]. Основні схеми, які розглядаються у цьому контексті, включають G16 [8], GM17 та Marlin.

Критерії вибору оптимальної схеми визначаються за кількома параметрами. Перший з них — це підтримка EVM-сумісних еліптичних кривих [9], що має коефіцієнт ваги 0.4. Різні криві мають різні оцінки: ALT_BN128 отримує +1, BLS12_377 або BW6_761 — +0.5, а BLS12_381 — +0. Цей параметр є важливим, оскільки забезпечує сумісність із популярними криптографічними стандартами і впливає на загальну ефективність протоколу.

Другий критерій — це універсальний сетап [10], що має ваговий коефіцієнт 0.3. Цей параметр визначає можливість перевикористання trusted setup, що є важливим для зниження складності і вартості розгортання системи. Якщо схема підтримує універсальний сетап, їй присвоюється 1 бал, якщо ні — 0.

Третій критерій стосується вразливості до атак відтворення і має коефіцієнт 0.2. Деякі схеми генерують докази, які можуть бути змінені, але залишатися валідними з точки зору перевіряючого. Стійкі до таких атак схеми отримують 1 бал, а ті, що не мають цього захисту — 0. Цей критерій є важливим для забезпечення надійності та безпеки системи.

Останній критерій — це підтримка бекендів, з ваговим коефіцієнтом 0.1. r1cs (рангова система обмежень) вимагає специфічної побудови системи генерації, і тут враховується, чи підтримує схема один або обидва основних бекенди: Bellman та Ark. Схема, яка підтримує один із бекендів, отримує 1 бал, а та, що підтримує обидва — 2.

Отже, вибір оптимальної схеми генерації доказів залежить від детального аналізу всіх зазначених критеріїв з урахуванням їх вагових коефіцієнтів. Це дозволяє визначити найбільш ефективну і надійну схему для використання у конкретному контексті (див. табл. 1).

Таблиця 1.1 – Значення критеріїв вибору схем генерації доказів (таблицю виконано самостійно)

	EVM-comp	Universal setup	Replay attack resistant	Backend
G16	1	0	0	2
GM17	1,5	0	1	2
Marlin	1,5	1	1	1

Після детального опису шкал оцінок за критеріями та відповідними коефіцієнтами можна визначити, яка зі схем є найбільш підходящою. Для цього необхідно виконати обчислення за наступною формулою:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \qquad (1.1)$$

Розрахунок за цією формулою був проведений у середовищі Excel з використанням відповідних функцій та інструментів для автоматизації процесу. Отримані

результати були ретельно проаналізовані та відображені на рисунку 1.1 розуміння.

	EVM-comp	Universal setup	Replay attack resistant	Backend		Z*
G16	1.000	0	0	2		0.140
GM17	1.500	0	1	2		0.290
Marlin	1.500	1	1	1		0.570
β	0.4	0.3	0.2	0.1		
α	0.25	1	0.5	0.2		

$$Z = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}$$

Рисунок 1.1 – Результати розрахунку (рисунок виконано самостійно)

Отримані дані після розрахунку та перетворені критерії додаємо до таблиці 1.2.

Таблиця 1.2 – Результати розрахунку (таблицю виконано самостійно)

	EVM-comp	Universal setup	Replay attack resistant	Backend	Z*
G16	1	0	0	2	0.140
GM17	1,5	0	1	2	0.290
Marlin	1,5	1	1	1	0.570

Таким чином, теоретично, для стандартних даних найкраще підходить Marlin [11], оскільки має максимальне значення Z*.

2 ОПИС ПРИЙНЯТИХ ПРОЄКТНИХ РІШЕНЬ

2.1 Аналіз методів роботи з ZoKrates

Продукт ZoKrates [12] працює на основі бібліотеки snark.js, яка відповідає за криптографічну частину протоколу zkSNARK. ZoKrates надає API для компіляції програм на мовах DSL ZoKrates (.zok), а також пропонує зручні методи параметризації протоколу та генерації trusted setup. Це робить його зручним інструментом для розробників, які бажають інтегрувати zkSNARK у свої проекти.

Для практичного експерименту доцільним буде використання програмного API ZoKrates, яке працює з WASM-байндингами до оригінального коду бібліотеки, написаного на Rust. Такий підхід із використанням посередника впливає на значення метрик, збільшуючи їх. Однак, оскільки метою роботи є порівняння різних методів, затримка, що виникає через використання посередника, не залежить від обраних опцій протоколу. Це означає, що її вплив є константним і однаковим для всіх варіантів, тому її можна не враховувати при аналізі результатів.

Таким чином, ZoKrates, з його API та підтримкою snark.js, є важливим інструментом для дослідження та порівняння різних схем zkSNARK у практичних умовах, забезпечуючи зручність та гнучкість у використанні.

2.2 Аналіз та вибір складності задач

Експеримент буде реалізований у формі ітеративного тестування різних комбінацій опцій протоколу на задачах різної складності та різного обсягу даних. Для забезпечення репрезентативності отриманих результатів було прийнято рішення створити чотири задачі, які відрізняються своєю складністю з точки зору протоколу zkSNARK. Складність задач визначатиметься не лише класичними обчисленнями, але й обчисленнями, що верифікуються, адже для цих двох класів задачі різна складність може бути спричинена різними аспектами.

Зокрема, zkSNARKs використовують поліноміальне представлення програм, тому на швидкість виконання таких програм сильно впливає розмірність даних. Крім того, просторову складність задачі суттєво визначає її ітеративність, що пов'язано з особливостями етапу генерації доказу в протоколі. Наприклад,

алгоритми хешування значно впливають на використання пам'яті, тоді як побудова дерева, яка включає рекурсію, матиме менший вплив на ресурси.

2.3 Опис задач

Легка задача. Першою задачею буде проста задача, яка включає всього два числові входи: число та його квадрат. У контексті нульового розголошення, елементом буде число, квадрат якого має співпасти із заданим значенням. Генерація доказу полягатиме в підтвердженні, що зазначене число дійсно є квадратним коренем даного числа. Ця задача є базовою та має мінімальні вимоги до обчислювальних ресурсів.

Середня задача. Задача середньої важкості включає побудову дерева Меркла з рандомізованою індексацією [13]. У вузлах дерева будуть зберігатися блоки даних, а генерація псевдовипадкових чисел реалізуватиметься за допомогою алгоритму xorwow [14]. Ця задача є типовою для згорток блокчейн-мереж другого рівня. Незважаючи на об'ємність реалізації, вона не вважається складною за умови правильного підходу до хешування. Завдання включає кілька етапів, де потрібно забезпечити цілісність даних у вузлах дерева. Важка задача

Третя задача включає хешування чотирьох чисел з послідовною перевіркою двох складових хешу (першої та другої половини байтів) з еталонним значенням. Крім важкості процесу хешування [15], ця задача ускладнюється необхідністю порівняння двох великих чисел, що є складовими хешу у десятковій системі. Таке порівняння значно збільшує вимоги до обчислювальних ресурсів та ускладнює задачу.

Надважка задача. Найскладніша задача включає хешування, порівняння хешів з параметрами, побітові операції над великими числами та розгалуження. Оскільки обчислення повинні бути представлені у вигляді безперервного арифметичного контуру, кожна з гілок умовних операторів виконується в системах обчислень, що верифікуються, включаючи zkSNARK. Це значно збільшує розмір обчислень та витрати ресурсів. Складність задачі зростає через необхідність одночасного виконання кількох складних обчислювальних процесів.

Таким чином, експеримент має на меті детально дослідити ефективність різних опцій протоколу zkSNARK при вирішенні задач різного рівня складності та з різними обсягами даних. Це дослідження дозволить отримати глибше розуміння того, як різні параметри протоколу впливають на продуктивність та ефективність обчислень у реальних умовах. Під час експерименту будуть розглянуті різноманітні варіанти конфігурацій та налаштувань протоколу, що дасть можливість виявити найоптимальніші рішення для конкретних задач. Це, у свою чергу, сприятиме розробці більш ефективних та продуктивних систем, які зможуть працювати з великими обсягами даних і виконувати складні обчислення швидше і з меншою витратою ресурсів.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Реалізація тестового стенду для проведення експерименту включає різні компоненти, які разом забезпечують всебічне тестування та аналіз продуктивності zkSNARK протоколів. Цей стенд складається з кількох основних елементів, кожен з яких виконує певну роль у загальному процесі експерименту.

Першим компонентом є набір імплементацій zokrates-програм різної складності. Це дозволяє тестувати протокол на різних задачах, починаючи від простих числових операцій до складних багатовимірних обчислень. Кожна програма була спеціально розроблена для оцінки різних аспектів продуктивності zkSNARK.

Допоміжні модулі для заміру часу, використаної пам'яті та розмірів виходу є наступним важливим компонентом стенду. Ці модулі забезпечують точний моніторинг і реєстрацію ресурсів, що використовуються під час виконання програм. Вони дозволяють отримувати об'єктивні дані про ефективність кожної програми, зокрема вимірюючи час виконання, обсяг споживаної пам'яті та розміри вихідних даних.

Методи створення синтетичних даних для виконання програм є невід'ємною частиною експерименту. Вони забезпечують генерацію наборів даних, необхідних для тестування різних сценаріїв. Ці дані можуть варіюватися за обсягом та структурою, що дозволяє детально вивчити вплив різних факторів на продуктивність zkSNARK протоколів.

Модуль нормалізації та ранжування використовується для обробки отриманих результатів. Він перетворює сирі дані у стандартизований формат, що дозволяє порівнювати результати між собою. Ранжування допомагає визначити найбільш ефективні конфігурації протоколу, спираючись на певні критерії.

Модуль візуалізації відіграє ключову роль у представленні результатів експерименту. Він забезпечує створення графіків, діаграм та інших візуальних матеріалів, які наочно демонструють продуктивність різних імплементацій zkSNARK. Це дозволяє легко інтерпретувати дані та робити висновки про ефективність протоколу.

Модуль роботи із ZoKrates забезпечує інтеграцію з цією платформою, дозволяючи автоматизувати процес компіляції та виконання zokrates-програм. Це значно спрощує експеримент, дозволяючи зосередитися на аналізі результатів, а не на технічних деталях реалізації.

Розглянемо деякі частини цього тестового стенду докладніше.

3.1 Zokrates програми

Проста задача реалізована як маленька функція порівняння чисел:

```
def main(private field a, field b) {
    assert(a * a == b);
    return;
}
```

Задача середньої складності має об’ємну реалізацію, тому наведемо основну її частину

Алгоритм генерації псевдовипадкових чисел xorwow:

```
struct XORWOW_STATE {
    u32[5] x;
    u32 counter;
    u32 current;
}
def xorwow(XORWOW_STATE mut state) -> XORWOW_STATE {

 /* Algorithm "xorwow" from p. 5 of Marsaglia, "Xorshift RNGs" */
    u32 mut t = state.x[4];

    u32 s = state.x[0]; /* Perform a contrived 32-bit shift. */
    state.x[4] = state.x[3];
    state.x[3] = state.x[2];
    state.x[2] = state.x[1];
    state.x[1] = s;

    t = t ^ (t >> 2);
    t =  t ^ (t << 1);
    t = t ^ s ^ (s << 4);

 state.x[0] = t;
    state.counter = state.counter + 362437;
    state.current = t + state.counter;
    return state;
}
```

Головна функція побудови дерева:

```
import "utils/pack/u32/nonStrictUnpack256.zok";
import "utils/pack/u32/pack256.zok";
import "utils/casts/u32_to_field.zok";

const u32 C32T256 = 8;
const u32 NUM_NODES = 11;
const u32 NUM_BLOCKS_PER_NODE = 10;

struct BLOCK_HEADER {
    field prevHash;
    u32 nonce;
    u32 timestamp;
    field uncleHash;
    field dataHash;
}

def main(private field encKeySeed,  public BLOCK_HEADER prevBH,
public BLOCK_HEADER[NUM_NODES][NUM_BLOCKS_PER_NODE] block_headers) ->
u32[8] {

    u32[C32T256] encKeySeed32 = nonStrictUnpack256(encKeySeed);
    // set up the random generator seed for xorwow function
     XORWOW_STATE mut state = XORWOW_STATE { x: [
        encKeySeed32[0] ^ encKeySeed32[1], encKeySeed32[2] ^
encKeySeed32[3], encKeySeed32[4] ^ encKeySeed32[5], encKeySeed32[6] ^
encKeySeed32[7],
        0xdeadc0de
     ], counter: 0xdeadbeef, current: 0 };

    // get randomly selected verifier block
    state = xorwow(state);
    u32 verifier_node_index = state.current % NUM_NODES;
    //debugInfo = [state.x[0], state.x[1], state.x[2], state.x[3],
state.x[4], state.counter, verifier_node_index, 0]
    // get the first node index % 11, could be verifier node too
    state = xorwow(state);
    u32 mut cur_node_index = state.current % NUM_NODES;
    cur_node_index = (cur_node_index + ((cur_node_index ==
verifier_node_index) ? 1 : 0)) % NUM_NODES;
    // get the random first block inside the nodes processed blocks
    state = xorwow(state);
    u32 mut block_index = state.current % NUM_BLOCKS_PER_NODE;
    // get a random stride to access the nodes processed blocks
    state = xorwow(state);
    u32 verifier_stride = state.current % NUM_BLOCKS_PER_NODE;
    // if any matches fail return 0 has the hash
    u32 mut verified_node_count = 0;

    u32[8] mut debugInfo = [verifier_node_index, cur_node_index,
block_index, verifier_stride, state.x[4], state.counter,
state.current, 0];

    // starting hash
    u32[8] mut xorHash = [0, 0, 0, 0, 0, 0, 0, 0];
    // loop through all 11 nodes
    for u32 i in 0..NUM_NODES-1 {
```

Продовження коду:

```
        u32[8] verifierHash =
nonStrictUnpack256(block_headers[verifier_node_index][verified_node_c
ount].dataHash);
        u32[8] nodeHash =
nonStrictUnpack256(block_headers[cur_node_index][block_index].dataHas
h);

        log("verifierHash is {}", verifierHash);
        log("iTH nodeindex/blockindex is {} - {},{} with nodeHash
{}", i, cur_node_index, block_index, nodeHash);
        u32 mut dataHashMatches = 0;
        // xor 256 bit hashes from field parameters
        for u32 j in 0..C32T256 {
            xorHash[j] = xorHash[j] ^ nodeHash[j];
            dataHashMatches = dataHashMatches + ((nodeHash[j] ==
verifierHash[j]) ? 1 : 0);
        }

        //log("dataHashMatches is {} for nth iteration {}",
dataHashMatches, i);

        // add +1 if this is not the verifier node and the node hash
matched with verifier node hash
        verified_node_count = verified_node_count + 1;

        // check next node skipping verifier node, wrap (modulo)
NUM_NODES
        cur_node_index = (cur_node_index + ((cur_node_index ==
verifier_node_index-1) ? 2 : 1)) % NUM_NODES;

        block_index = (block_index + verifier_stride) %
NUM_BLOCKS_PER_NODE;
    }

    debugInfo[7] = verified_node_count;

    // return if NUM_NODES - verfier (1 node) have been verified.
    //return encKeySeed
    //return (verified_node_count == NUM_NODES-1) ? pack256(xorHash)
: pack256(debugInfo)
    //assert (verified_node_count == NUM_NODES-1)
    return (verified_node_count == NUM_NODES-1) ? xorHash :
debugInfo;
}
```

Важка задача, реалізована як порівняння хешів:

```
import "hashes/sha256/512bitPacked" as sha256packed;

def main(private field a, private field b, private field c, private
field d) {
    field[2] h = sha256packed([a, b, c, d]);
    assert(h[0] == 263561599766550617289250058199814760685);
    assert(h[1] == 65303172752238645975888084098459749904);
    return;
}
```

І надважка задача хешування і побітових маніпуляцій:

```
import "hashes/sha256/512bitPacked" as sha256packed;

def main(private field value, private field before, field valueHash,
field beforeHash, field afterHash) -> field {
field[2] priBefore = sha256packed([0, 0, 0, before]);
field[2] priAfter = sha256packed([0, 0, 0, before-value]);
    field result = value > before && priBefore[0] == beforeHash &&
priAfter[0] == afterHash ? 1 : 0;
    return result;
}
```

3.2 Тестовий стенд

Заміри метрик, а саме часу виконання, спожитої пам'яті та об'єму виходу, відбуваються на трьох основних етапах роботи протоколу: обчислення, генерація доказу та верифікація доказу. На першому етапі, обчисленні, вимірюється час, необхідний для виконання вихідної програми, а також обсяг пам'яті, що використовується під час цього процесу. Це дозволяє зрозуміти базовий рівень продуктивності для кожної задачі.

На другому етапі, генерації доказу, заміряється час, необхідний для створення криптографічного доказу, що підтверджує правильність виконання програми, і кількість пам'яті, що використовується. Цей етап є критично важливим, оскільки процес генерації доказу може бути ресурсоємним і вимагати значних обчислювальних потужностей. Вимірювання на цьому етапі допомагають оцінити ефективність протоколу в умовах реальних навантажень.

Третій етап, верифікація доказу, включає перевірку створеного доказу на коректність. Тут заміряється час, необхідний для перевірки доказу, а також обсяг пам'яті, що використовується під час цього процесу. Верифікація є важливою частиною протоколу, оскільки вона підтверджує автентичність і правильність обчислень, забезпечуючи тим самим надійність системи.

Наведемо приклад коду побудови ітеративної задачі генерації доказу:

```
private buildProofGenerationTask = (
    iterations: number,
    program: Uint8Array,
    pk: Uint8Array,
    witnesses: Uint8Array[],
) => {
return async () => {
const proofs = new Array<Proof>();
```

Продовження коду:

```
for (let i = 0; i < iterations; i++) {
              const  proof  =  generateProof(this.provider,  program,
witnesses[i], pk);
        proofs.push(proof);
      }

      return proofs;
    };
  };
```

А також коду заміру метрик цієї генерації:

```
private async testProofGeneration(
    task: ReturnType<typeof this.buildProofGenerationTask>,
    iterations: number,
  ): Promise<Proof[]>
{
    const {
      memoryConsumed: proofGenerationMemoryMetric,
      result: { timeConsumed: proofGenerationTimeMetric, result: proofs
},
    } = await measure(task);

    this.benchmark.proofGeneration = {
      timeMetric: proofGenerationTimeMetric / iterations,
      memoryMetric: proofGenerationMemoryMetric / iterations,
      volumeMetric: getBytesVolume(proofs) / iterations,
    };

    return proofs;
  }
```

Де функція заміру визначається як композитна функція заміру залученої пам’яті та часу:

```
export const measure = (task: () => Promise<any>) =>
  memoryConsumption(() => timeConsumption(task));
```

Кожен набір параметрів перевіряється на кожній із задач, що дозволяє оцінити продуктивність протоколу в різних умовах.

Отримані метрики записуються в JSON файл для подальшої обробки. Це забезпечує структуроване зберігання даних, що спрощує їх подальший аналіз, обробку та візуалізацію. JSON формат також дозволяє легко обмінюватися даними між різними системами та інструментами, що може бути корисним для більш комплексних досліджень.

Для більш детального розуміння процесу формування комбінацій параметрів та запуску експерименту, розглянемо код на TypeScript, що відповідає за ці функції:

```
async function main() {
  const iterations_progression = [10];
```

```
      const schemes: Scheme[] = ['g16', 'gm17', 'marlin'];
      const curves: Curve[] = ['bls12_377', 'bls12_381', 'bn128',
'bw6_761'];
      const backends: Backend[] = ['ark', 'bellman'];

      interface QualifiedProvider {
        provider: ZoKratesProvider;
        options: Options;
      }

      const providers: QualifiedProvider[] = [];

      for (const scheme of schemes) {
        for (const curve of curves) {
          for (const backend of backends) {
            let config: Options = {
              scheme,
              backend,
              curve,
            };
            try {
              const provider = await getZokratesInstance(config);

              providers.push({
                provider,
                options: config,
              });
            } catch (e) {
              console.log(`Options did't fit: ${config}`);
            }
          }
        }
      }

      const zok_sets: MockedSet[] = [
        {
          zok_program_name: ZOK_PROGRAM_SIMPLE,
          mock: mockData_simple_static,
          note: 'static',
        },
        {
          zok_program_name: ZOK_PROGRAM_MIDDLE,
          mock: mockData_middle_static,
          note: 'static',
        },
        {
          zok_program_name: ZOK_PROGRAM_COMPLEX,
          mock: mockData_complex_static,
          note: 'static',
        },
        {
          zok_program_name: ZOK_PROGRAM_ADVANCED,
          mock: mockData_advanced_static,
          note: 'static',
        },
    ];

      const toPath = (name) =>
```

Продовження коду:

```
        join(cwd(), ZOK_PROGRAMS_DIR, name + Extention.Zokrates);

      for (const provider of providers) {
        for await (const zok_set of zok_sets) {
          const { mock, zok_program_name, note } = zok_set;
          const zok_path = toPath(zok_program_name);
          const testBench = new ZkTester(provider.provider, mock,
zok_path);

          for await (const iterations of iterations_progression) {
            const { scheme, curve, backend } = provider.options;
            try {
              await testBench.testIterative(iterations);

              console.log(`Successful:
${JSON.stringify(provider.options)}`);
              console.log(testBench.benchmark);

              await testBench.saveBenchmark({
                descriptor: `${zok_program_name}:${iterations}${note ?
`@${note}` : ''}`,
                curve,
                scheme,
                backend,
                type: zok_program_name,
              });
            } catch (e) {
              console.log(
                `Provider thrown: ${JSON.stringify(provider.options)},
${e}`,
              );
              continue;
            }
          }
        }
      }
```

Цей код використовує generic-клас, який виконує кожну з операцій задану кількість разів. Після виконання операцій, клас обчислює середнє значення отриманих результатів та зберігає їх у файл. Такий підхід дозволяє отримати більш точні та стабільні показники продуктивності, усуваючи можливі випадкові відхилення та забезпечуючи репрезентативність даних.

Важливо зазначити, що результат верифікації не є важливим для отримання замірів. Це пов’язано з тим, що послідовність операцій, яка виконується над валідним або невалідним доказом, є ідентичною. Відповідно, результати замірів всіх метрик, таких як час виконання, спожита пам'ять і об'єм виходу, не будуть

відрізнятися в обох випадках. Це дає змогу спростити процес тестування, зосереджуючись на замірах продуктивності, незалежно від валідності доказів.

Виходячи з цього, функції-генератори даних, які використовуються для створення синтетичних входів для програми, можуть генерувати будь-які дані. Ці дані можуть бути статичними чи динамічними, валідними чи невалідними. Головна вимога полягає в тому, щоб вони відповідали сигнатурі головної функції програми. Інші аспекти даних не впливають на результати замірів, що значно спрощує процес підготовки до експерименту.

Таким чином, використання generic-класу для виконання операцій, обчислення середніх значень та збереження результатів дозволяє отримати надійні та точні дані для подальшого аналізу. Незалежність замірів від валідності доказів та гнучкість у використанні різних типів синтетичних даних роблять цей підхід ефективним інструментом для оцінки продуктивності протоколу zkSNARK у різних умовах.

4 ОПИС ЕКСПЕРЕМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Проведення експериментальних досліджень

Після проведення теоретичного моделювання та замірів реальних значень по трьох метриках, можемо перейти до детального аналізу отриманих даних. Оскільки на практичні результати обчислення задач високої складності впливатиме не лише вибір схем, але й вибір цільових еліптичних кривих і бекендів, було вирішено розширити поле тестових наборів. Це розширення здійснюється за рахунок комбінацій схем із різними еліптичними кривими та поширеними бекендами, щоб забезпечити комплексний підхід до тестування.

З'ясувалося, що деякі комбінації не підтримують усі типи необхідних обчислень у складних задачах. Тому аналіз буде проводитися на трьох валідних комбінаціях: схема g16 з кривою bn128 і бекендом ark, схема g16 з кривою bn128 і бекендом bellman, а також схема gm17 з кривою bn128 і бекендом ark. Ці комбінації обрані через їхню здатність забезпечувати необхідний рівень підтримки для складних обчислювальних задач.

Після проведення вимірів на десяти ітераціях з усередненням результатів, було отримано 18 семплів даних, які представляють собою значну базу для аналізу.

Приклад такого семплу бенчмарка:


```
{
  "computation": {
    "timeMetric": 6332.663020002842,
    "memoryMetric": 9084.8,
    "volumeMetric": 27761404
  },
  "proofGeneration": {
    "timeMetric": 27719.686459994315,
    "memoryMetric": 10149.6,
    "volumeMetric": 836
  },
  "verification": {
    "timeMetric": 33.24549000263214,
    "memoryMetric": 15672,
    "volumeMetric": 4
  },
  "descriptor": "advanced:10@static",
  "type": "advanced",
  "scheme": "g16",
  "curve": "bn128",
  "backend": "ark"
}
```

Кожен із цих семплів надає детальну інформацію про продуктивність і ефективність кожної комбінації схем, кривих та бекендів у різних умовах.

Перед тим як приступити до аналізу цих даних, необхідно провести їх нормалізацію [16]. Нормалізація потрібна для приведення значень до проміжку [0, 1], що дозволить уніфікувати дані та полегшити їх порівняння. Для цього використовується модифікований minmax підхід [17], який враховує специфічні особливості домену, щоб забезпечити максимально точну та релевантну нормалізацію.

Цей процес включає в себе виявлення мінімальних і максимальних значень серед усіх зібраних даних для кожної з метрик. Потім кожне значення метрики трансформується відповідно до формули minmax, що дозволяє привести всі значення до єдиного масштабу. Такий підхід гарантує, що результати будуть коректно відображати реальні відмінності між різними комбінаціями параметрів, що тестуються.

Введемо такі параметри:

– C – комбінація опцій (схема + крива + бекенд);

– T – тип задачі (легка, середня, складна, надскладна);

– R – множина результатів експерименту (бенчмарки);

– M – метрика (час, пам’ять, об’єм);

– $V_{C,T}$ – значення деякої метрики M для комбінації C і задачі T;

– W_T – вага типу задачі T.

Визначимо набір комбінацій по складності задач:

$$\text{combinationMap}(C) = \{T_1, T_2, \dots, T_k\} \tag{4.1}$$

Визначимо усі комбінації, заміри яких наявні для всіх складностей:

$$\text{validCombinations} = \{C \mid \text{combinationMap}(C) = \{T_1, T_2, \dots, T_n\}\} \tag{4.2}$$

Тоді мінімальним і максимальним значенням метрики M будуть відповідно:

$$\min(M) = \min_{C \in \text{validCombinations},T} V_{C,T}(M) \tag{4.3}$$

$$\max(M) = \max_{C \in \text{validCombinations},T} V_{C,T}(M) \tag{4.4}$$

Нормалізуємо метрики M серед складності T по min-max:

$$\text{normalizedValue}\left(V_{C,T}(M)\right) = \frac{V_{C,T}(M) - \min(M)}{\max(M) - \min(M)} \quad (4.5)$$

Тоді загальна формула нормалізації значення метрики буде:

$$\text{normalizedValue}\left(V_{C,T}(M)\right) = \frac{V_{C,T}(M) - \min\limits_{C \in validCombinations,T} V_{C,T}(M)}{\max\limits_{C \in validCombinations,T} V_{C,T}(M) - \min\limits_{C \in validCombinations,T} V_{C,T}(M)} \quad (4.6)$$

Завдяки проведеній нормалізації та подальшому аналізу, можна буде отримати глибше розуміння того, як різні параметри впливають на продуктивність протоколу zkSNARK. Це дозволить зробити обґрунтовані висновки про найефективніші комбінації для використання в реальних умовах, забезпечуючи оптимальний баланс між продуктивністю та обчислювальними ресурсами.

5 АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Візуалізація результатів

Перед алгоритмічним ранжуванням візуалізуємо нормалізовані дані у вигляді лінійних і стовпчастих графіків. Ці візуалізації допомагають наочно продемонструвати тенденції та зміни в продуктивності різних комбінацій схем, кривих та бекендів у контексті зростаючої складності задач (див. рис. 5.1-5.6).

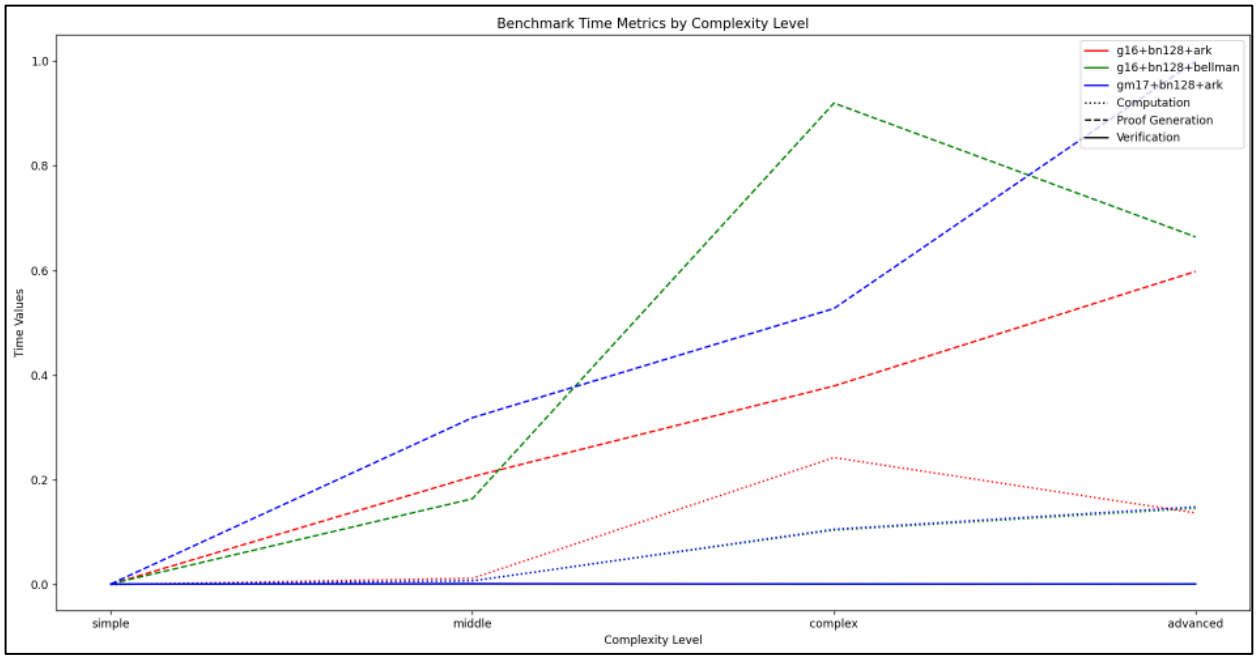

Рисунок 5.1 – Графік залежності затребуваного часу від складності задачі (рисунок виконано самостійно)

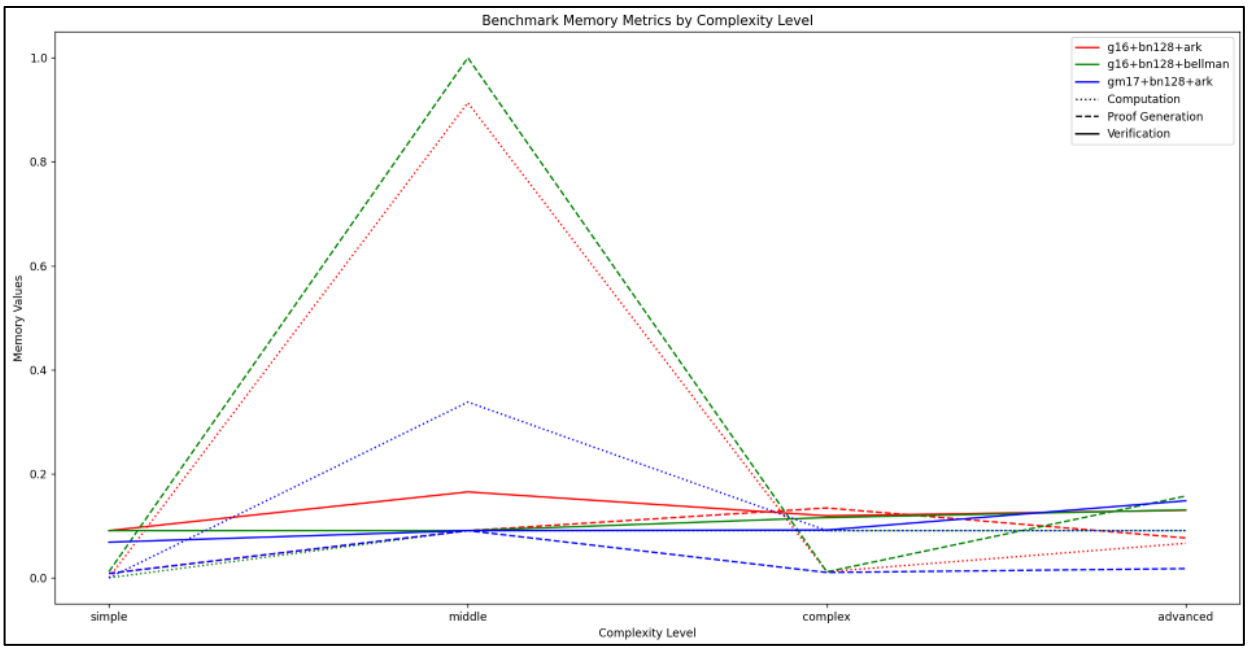

Рисунок 5.2 – Графік залежності затребуваної пам'яті від складності задачі (рисунок виконано самостійно)

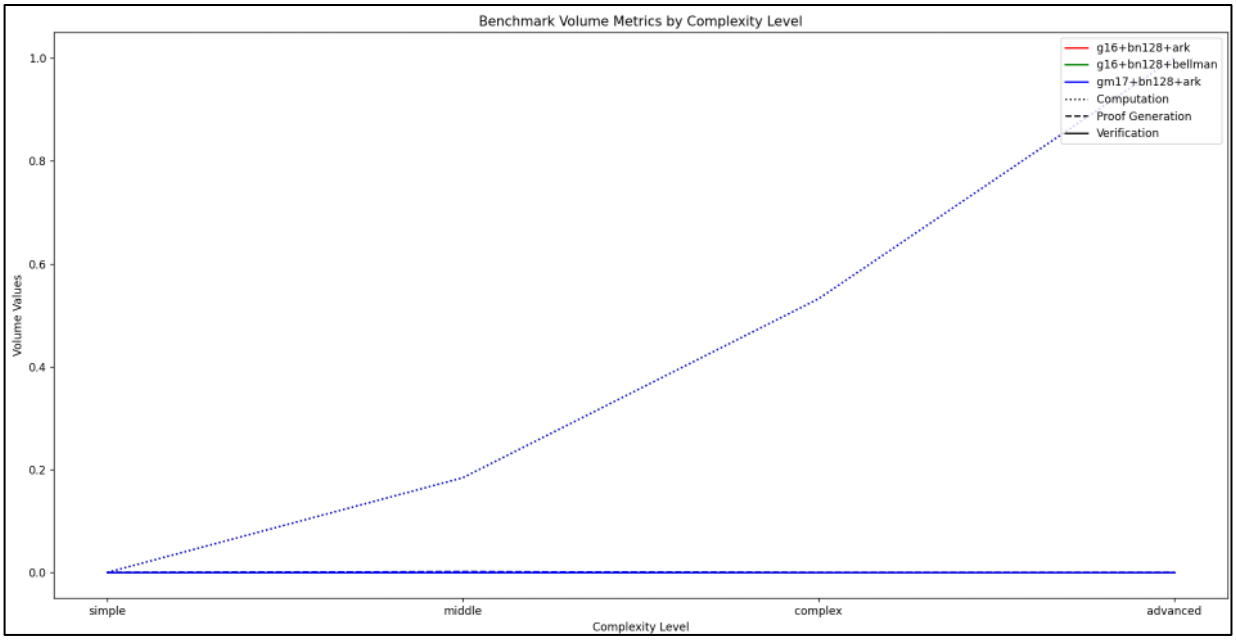

Рисунок 5.3 – Графік залежності затребуваного об'єму від складності задачі (рисунок виконано самостійно)

Далі наведено стовбчасті діаграми, які для кожної метрики репрезентують її значення на кожній з комбінацій по кожному з процесів (обчислення, генерація доказу, верифікація) в залежності від складності задачі.

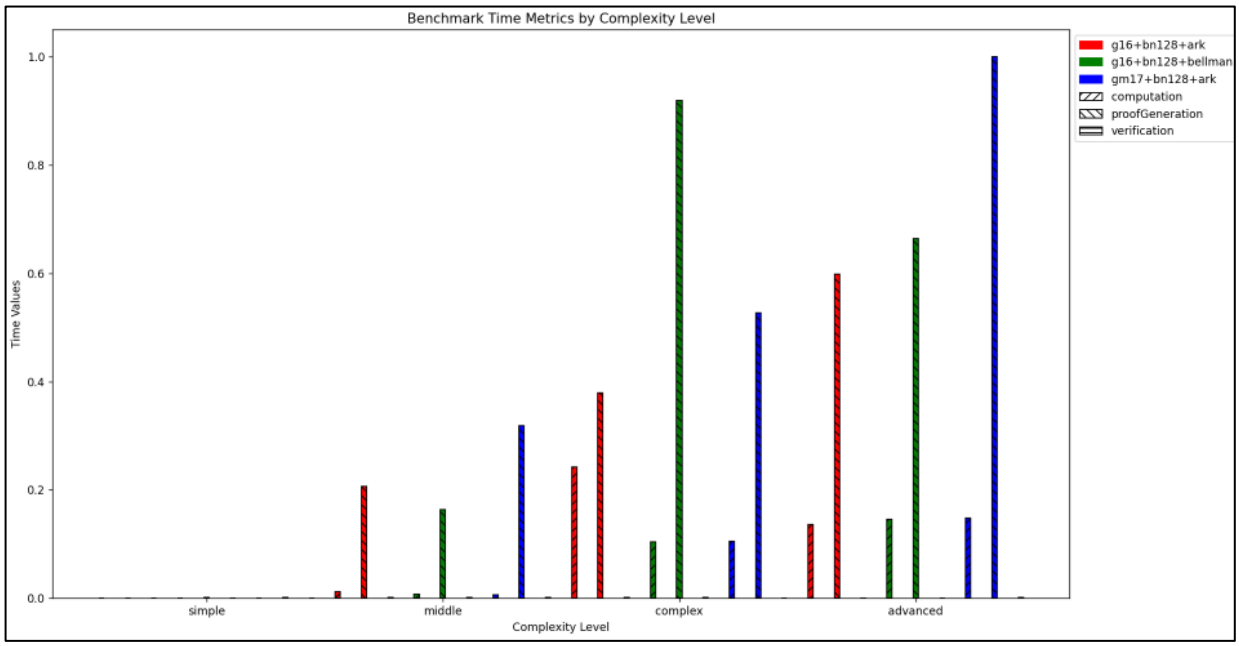

Рисунок 5.4 – Діаграма залежності затребуваного часу від складності задачі (рисунок виконано самостійно)

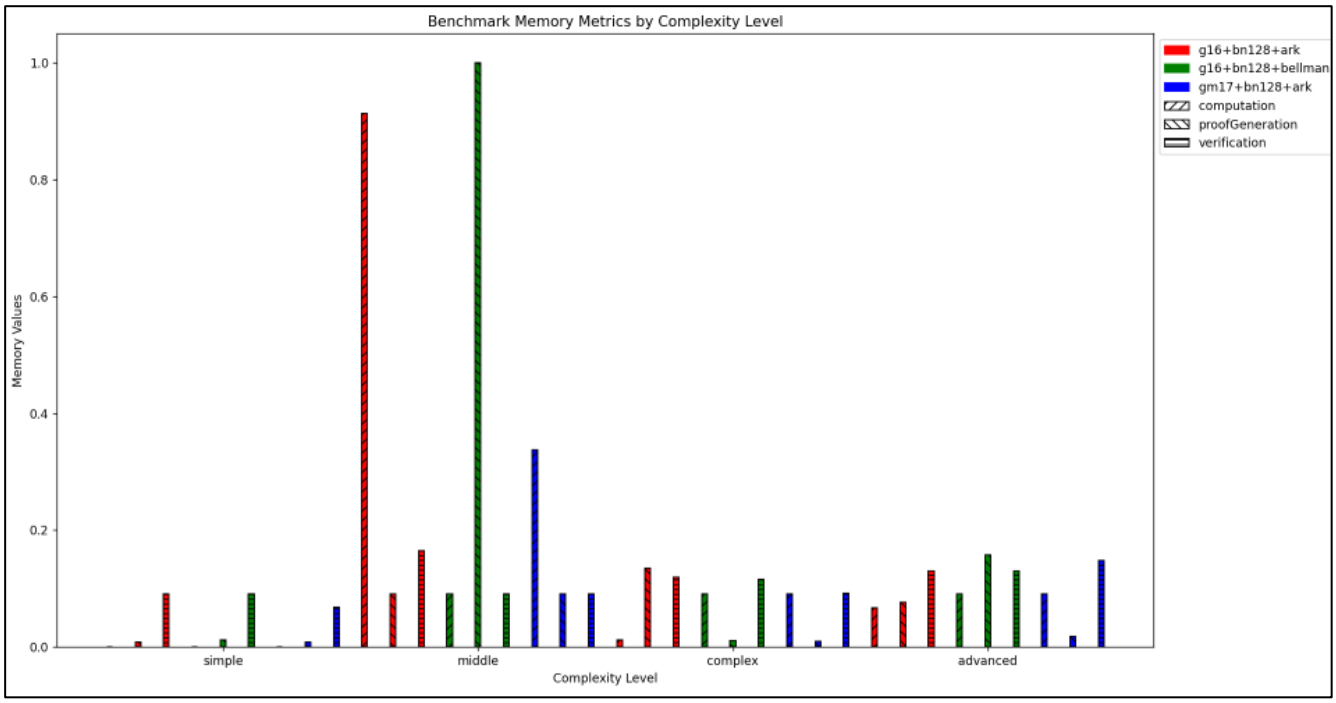

Рисунок 5.5 – Діаграма залежності затребуваної пам'яті від складності задачі (рисунок виконано самостійно)

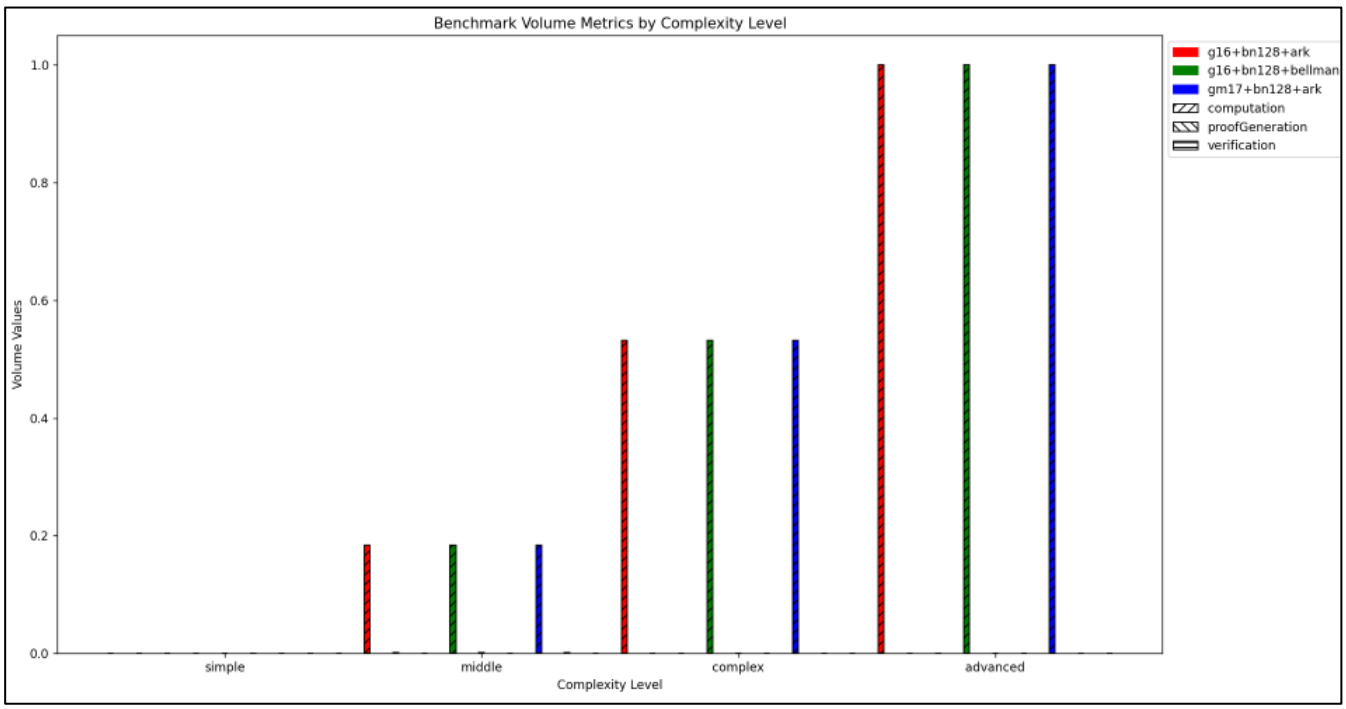

Рисунок 5.6 – Діаграма залежності затребуваного об'єму від складності задачі (рисунок виконано самостійно)

З візуалізацій бачимо, що комбінація {g16 схема + bn128 крива + bellman бекенд} виявляється ефективнішою на складних задачах, на які і орієнтуєнтоване

дослідження. Це можна помітити за повільнішим зростанням значень метрик при збільшенні складності задач.

5.2 Алгоритмічне ранжування

Після візуалізації, переходимо до процесу ранжування. Для цього було обрано модифікований метод лінійної адитивної згортки із застосуванням вагових коефіцієнтів [18]. Цей метод дозволяє об'єднати різні метрики в один загальний показник, враховуючи відносну важливість кожної метрики для кінцевого результату. Вагові коефіцієнти визначаються на основі попереднього аналізу і оцінки, що дозволяє забезпечити справедливе і обґрунтоване ранжування різних комбінацій:

```
taskWeights = {
  simple: 1,
  middle: 2,
  complex: 3,
  advanced: 4,
};
metricWeights = {
  computationTime: 2.0,
  computationMemory: 0.5,
  computationVolume: 0.5,
  proofGenerationTime: 2.0,
  proofGenerationMemory: 1.0,
  proofGenerationVolume: 0.5,
  verificationTime: 3.0,
  verificationMemory: 2.5,
  verificationVolume: 1.0,
};
```

Завдяки такому підходу, можемо ефективно порівнювати продуктивність різних варіантів zkSNARK, беручи до уваги всі важливі аспекти їхньої роботи. Це дозволяє не лише визначити найкращі комбінації для використання в реальних умовах, але й отримати глибше розуміння того, як різні фактори впливають на загальну ефективність протоколу.

Коефіцієнти значущості типів задач були визначені у вигляді арифметичної прогресії з кроком +1, що дозволяє підвищувати значущість кожного наступного типу задачі від простої до надважкої. Таким чином, алгоритм буде віддавати перевагу тим наборам, які показують кращі результати на складніших задачах, а

успіхи лише в легких обчисленнях не матимуть великого впливу на кінцевий результат.

Цей підхід забезпечує об'єктивну оцінку продуктивності комбінацій, акцентуючи увагу на їх здатності ефективно працювати з важкими обчислювальними задачами. Для кожного типу задачі (легка, середня, важка, надважка) присвоюються відповідні коефіцієнти значущості, які відображають їх відносну важливість у загальному аналізі.

Обчислимо бали кожної метрики по комбінаціям за типом задачі:

$$\text{Score}(C, MT) = W_m(\text{time}) \cdot V_{C,MT,\text{timeMetric}} + W_m(\text{memory}) \cdot V_{C,MT,\text{memoryMetric}} + W_m(\text{volume}) \cdot V_{C,MT,\text{volumeMetric}} \quad (5.1)$$

де C – комбінація опцій (схема + крива + бекенд),

MT – тип метрики (обчислення, генерація доказу, верифікація),

M – метрика (час, пам’ять, об’єм),

$V_{C,MT,M}$ – значення деякої метрики M для комбінації C і типу метрики MT,

W_m – вага метрики M.

Тепер можемо обчислити бали кожної з комбінацій за отриманими балами метрики:

$$\text{TotalScore}(C) = \sum_{MT \in \{\text{computation,proofGeneration,verification}\}} \text{Score}(C, MT) \quad (5.2)$$

Сортування результатів згідно балів за спаданням дозволяє визначити топ комбінацій, які проявили себе краще з огляду на складність задачі.

Провівши попередню агрегацію нормалізованих даних, ранжуємо комбінації за допомогою методів-імлементацій вище описаного методу:

```
function calculateScore(metric: Metric, metricType: string): number {
  const weights = {
    timeMetric: metricWeights[`${metricType}Time`],
    memoryMetric: metricWeights[`${metricType}Memory`],
    volumeMetric: metricWeights[`${metricType}Volume`],
  };

  return (
    metric.timeMetric * weights.timeMetric +
    metric.memoryMetric * weights.memoryMetric +
    metric.volumeMetric * weights.volumeMetric
  );
}

export function rankSetCombinations(
```

```
  aggregatedMetrics: AggregatedMetrics[],
): AggregatedMetrics[] {
  const rankedCombinations = aggregatedMetrics.sort((a, b) => {
    const aScore =
      calculateScore(a.weightedComputation, 'computation') +
      calculateScore(a.weightedProofGeneration, 'proofGeneration') +
      calculateScore(a.weightedVerification, 'verification');
```

Продовження коду:

```
    const bScore =
      calculateScore(b.weightedComputation, 'computation') +
      calculateScore(b.weightedProofGeneration, 'proofGeneration') +
      calculateScore(b.weightedVerification, 'verification');

    return aScore - bScore; // Lower score is better
  });

  return rankedCombinations;
}
```

Після ранжування було встановлено, що перше місце серед тестованих комбінацій займає {scheme: 'g16', curve: 'bn128', backend: 'ark'}. Ця комбінація показала найкращі результати при вирішенні прикладних задач з великою складністю і щільністю обчислень. Друге місце посіла комбінація {scheme: 'gm17', curve: 'bn128', backend: 'ark'}, а третє місце – {scheme: 'g16', curve: 'bn128', backend: 'bellman'}.

Комбінація схеми g16 з кривою bn128 та r1cs бекендом ark виявилася оптимальною для вирішення задач з високою складністю. Це можна пояснити кількома факторами:

– схема g16 відома своєю високою ефективністю при генерації та верифікації доказів; вона використовує оптимізовані поліноміальні структури, які дозволяють швидко виконувати обчислення навіть при великих обсягах даних;

– еліптична крива bn128 забезпечує високу криптографічну безпеку та оптимізовану продуктивність, що є критично важливим для забезпечення надійності zkSNARK протоколу;

– використання бекенду ark дозволяє ефективно обробляти складні обчислення завдяки його спеціалізованим алгоритмам, що забезпечують високу продуктивність і стабільність системи.

Комбінація схеми gm17 з тією ж кривою bn128 і бекендом ark також показала високу продуктивність, хоча й поступається першому місцю. Схема gm17

забезпечує додаткову гнучкість та безпеку, використовуючи складніші математичні конструкції, що робить її стійкішою до потенційних вразливостей. Проте, її продуктивність виявилася дещо нижчою у порівнянні з g16, що може бути пов’язано з більшими витратами ресурсів на обробку додаткових обчислень.

Комбінація схеми g16 з кривою bn128 та бекендом bellman також показала хороші результати, але поступилася іншим комбінаціям. Бекенд bellman забезпечує надійність та стабільність, проте може бути менш ефективним у порівнянні з ark при обробці дуже складних обчислень.

Варто зазначити, що в ході теоретичного експерименту було визначено оптимальною схему Marlin, а наступною за нею – GM17. Однак, графіки, які не враховували вагових коефіцієнтів параметрів і, відповідно, не відображали важливості впливу деяких метрик і етапів у реальних задачах, підтверджували першість набору {scheme: 'g16', curve: 'bn128', backend: 'bellman'}.

Проте, при врахуванні особливостей прикладних задач, таких як складність і щільність обчислень, оптимальним набором було обрано {scheme: 'g16', curve: 'bn128', backend: 'ark'}. Цей підхід забезпечує більшу ефективність і продуктивність у вирішенні складних задач, що підтверджується результатами проведеного експерименту.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено ґрунтовний аналіз предметної області доказів із нульовим розголошенням, зокрема протоколу zkSNARK. Було досліджено різні схеми генерації доказів, еліптичні криві та бекенди, які використовуються в рамках цього протоколу.

Після теоретичного аналізу та моделювання була визначена оптимальна схема Marlin для стандартних задач згідно обраних критеріїв. Однак, подальші практичні дослідження показали, що для складних прикладних задач, які вимагають високої щільності обчислень, більш оптимальною виявилася комбінація схеми g16, еліптичної кривої bn128 та бекенду ark.

Були розроблені програмні рішення для проведення експериментів з вимірювання метрик роботи протоколу для різних комбінацій параметрів на задачах різної складності. Отримані результати були проаналізовані з урахуванням важливості метрик та складності задач за допомогою модифікованого методу лінійної адитивної згортки.

Таким чином, основним результатом роботи є визначення оптимального набору параметрів протоколу zkSNARK для ефективного застосування в системах із складними обчислювальними задачами, що вимагають високого рівня конфіденційності та безпеки даних. Це відкриває нові можливості для впровадження технологій доказів із нульовим розголошенням у різноманітних сферах, таких як державний сектор, оборонна промисловість та соціально-орієнтовані системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Chen, T., Lu, H., Khunphithya, T., & Luo, A. 2022. A Review of zk-SNARKs. arXiv preprint arXiv:2202.06877 / URL: https://arxiv.org/abs/2202.06877 (дата звернення 12.02.2024).

2. Lai, Y.T. 2023. Modern Zero Knowledge Cryptography, Lecture 7: Arithmetisations- MIT IAP, 2023.1. Lecture 7: Arithmetisations, 23 January 2023 / URL: https://assets.super.so/9c1ce0ba-bad4-4680-8c65-3a46532bf44a/files/e11309fb-7356-42ad-9c78-565341abd80d.pdf (дата звернення: 12.02.2024).

3. Park, C., Chung, M., & Ryu, D. 2023. A Blockchain-based Protocol of Trusted Setup Ceremony for Zero-Knowledge Proof. BIOTC '23: Proceedings of the 2023 5th Blockchain and Internet of Things Conference. July 2023. Pages 35–40 / URL: https://doi.org/10.1145/3625078.3625083 (дата звернення: 12.02.2024).

4. Kappos, G., Yousaf, H., Ateniese, G., & Meiklejohn, S. 2020. A Refined Analysis of Zcash Anonymity / URL: https://www.researchgate.net/publication/339194579_A_Refined_Analysis_of_Zcash_Anonymity (дата звернення: 12.02.2024).

5. Качко О.Г., Мельникова О.А. Some methods for increasing the speed of operations on elliptic curves in the normal basis // 10-а науково-технічна конференція "Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні". Науково-технічний збірник. – Вип. 11. – Київ, 2005.

6. Качко О.Г., Аулов І.Ф. Mechanisms for improving the performance of cryptographic libraries for end users in cloud technologies // Міжнародна наукова школа – семінар Питання оптимізації обчислень (ПОО-XLII) 21-25 вересня 2015. Україна, Закарпатська область, Мукачівський район, смт Чинадієво. – 21-25 вересня 2015 року.

7. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., & Virza, M. 2019. Aurora: Transparent Succinct Arguments for R1CS / URL: https://www.semanticscholar.org/paper/Aurora%3A-Transparent-Succinct-Arguments-for-R1CS-Ben-Sasson-Chiesa/e3add65a298819a5dc34d0e3e3480bb1432bce5b (дата звернення:12.02.2024).

8. Baghery, K., Kohlweiss, M., Siim, J., & Volkhov, M. 2021. Another Look at Extraction and Randomization of Groth's zk-SNARK. Financial Cryptography 2021 / URL: https://iohk.io/en/research/library/papers/another-look-at-extraction-and-randomization-of-groths-zk-snark/ (дата звернення: 12.02.2024).

9. Hoseini, S.V. 2018. Mathematics and Data Structures in Blockchain and Ethereum. Aalto University / URL: https://www.researchgate.net/publication/340418164_Mathematics_and_Data_Structures_in_Blockchain_and_Ethereum (дата звернення: 12.02.2024). DOI:10.13140/RG.2.2.35846.52805/1. Advisor: Valtteri Niemi.

10. Kanjalkar, S., Zhang, Y., Gandlur, S., & Miller, A. 2021. Publicly Auditable MPC-as-a-Service with succinct verification and universal setup. Universidad Pontificia Comillas / URL: https://www.researchgate.net/publication/353170920_Publicly_Auditable_MPC-as-a-Service_with_succinct_verification_and_universal_setup (дата звернення 12.02.2024).

11. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, P., & Ward, N. 2020. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS / URL: https://eprint.iacr.org/2019/1047.pdf (дата звернення: 12.02.2024).

12. Eberhardt, J., & Tai, S. 2018. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). DOI:10.1109/Cybermatics_2018.2018.00199. Technische Universität Berlin / URL: https://www.researchgate.net/publication/333585572_ZoKrates_-_Scalable_Privacy-Preserving_Off-Chain_Computations (дата звернення: 12.02.2024).

13. Ma, B., Pathak, V.N., Liu, L., & Ruj, S. 2024. One-Phase Batch Update on Sparse Merkle Trees for Rollups. In Distributed Ledger Technology (pp.1-21). Indian Institute of Technology Indore. DOI:10.1007/978-981-97-0006-6_1 / URL: https://www.researchgate.net/publication/378084159_One-Phase_Batch_Update_on_Sparse_Merkle_Trees_for_Rollups (дата звернення: 12.02.2024).

14. Marsaglia, G. 2003. Xorshift RNGs. Journal of Statistical Software, 8(14), 1–6 / URL: https://www.jstatsoft.org/article/view/v008i14 (дата звернення: 12.02.2024).

15. Качко О.Г., Телевний Д.К. Study of applicability of SMT/SAT proofs in cryptanalysis of Keccak family hash functions // Журнал Радіотехніка. – 2017. – Вип. 189. – С. 75-80.

16. Patro, S.G.K., & Sahu, K.K. 2015. Normalization: A Preprocessing Stage. arXiv preprint arXiv:1503.06462. Research Scholar, Department of CSE & IT, VSSUT, Burla, Odisha, India / URL: https://arxiv.org/pdf/1503.06462 (дата звернення: 12.02.2024).

17. Jain, Y., & Bhandare, S.K. 2011. Min Max Normalization Based Data Perturbation Method for Privacy Protection. International Journal of Computer and Communication Technology, 2(8). DOI:10.47893/IJCCT.2013.1201. Samrat Ashok Technological Institute / URL: https://www.researchgate.net/publication/228518958_Min_Max_Normalization_Based_Data_Perturbation_Method_for_Privacy_Protection (дата звернення: 12.02.2024).

18. Yılmaz, E., Aydın, D., & Ahmed, S.E. 2023. Modified Local Linear Estimators in Partially Linear Additive Models with Right-Censored Data Based on Different Censorship Solution Techniques. Entropy, 25(9), 1307. DOI: https://doi.org/10.3390/e25091307 / URL: https://www.mdpi.com/1099-4300/25/9/1307 (дата звернення: 12.02.2024).