Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра обчислювальної математики

Кваліфікаційна робота
на здобуття ступеня бакалавра

за спеціальністю 113 Прикладна математика
на тему:

Оптимізація гаджет бібліотек для рекурсивних zk-Snarks

Виконала студентка 4-го курсу
Ощипок Олена-Іванна Василівна

Науковий керівник:
асистент
Денисов Сергій Вікторович

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студентка

Роботу розглянуто й допущено до захисту на засіданні кафедри обчислювальної математики
«29» травня 2023 р.,
протокол № 8
Завідувач кафедри
С. І. Ляшко

Київ – 2023

РЕФЕРАТ

Обсяг роботи 43 сторінок, 8 ілюстрацій, 11 використаних джерел.

Об’єктом дослідження є множення точок еліптичної кривої на скаляр в системі гаджет бібліотеках рекурсивних zk-Snark’s. Множення повинно виконуватися до стандартів протоколу Zero-knowledge proof. Також розглянуто варіанти практичного застосування описаних методів.

Метою роботи є оптимізувати множення точок еліптичної кривої на скаляр за допомогою модернізованого метода “Подвійне скалярне множення з використанням трюку Штрауса-Шаміра з урахуванням Skew representation” та багато інших підходів. Ще одною метою є роботи: зробити використання множення точок еліптичної кривої на скаляр безпечним в межах протоколу Zero-knowledge proof. З'ясувати, який метод множення буде найдешевшим в контексті визначеної метрики. Зробити висновки згідно з отриманими результатами.

У роботі виконане теоретичне та практичне дослідження, огляд алгоритмів та методів розв’язання задачі оптимізації з використанням різноманітних хитростей та підходів. Кодова база була написана мовою програмування Rust в бібліотеці franklin-crypto. Арифметизація, яка застосовується в бібліотеці – Plonkish та lookup table. Крива, яка була використана для тестування множення – Bn256.

ЗМІСТ

ВСТУП

Сучасна криптографія – це метод захисту інформації й комунікації за допомогою шифрів, коду, і щоб тільки ті для кого назначена ця інформація, могли її читати та обробляти. Криптографія використовує такі інструменти абстрактної алгебри та теорії ймовірності. Сучасна криптографія має задовольняти наступні критерії:

1. Конфіденційність. Інформація не можу бути зрозуміла кимось, для кого вона не була призначена.
2. Цілісність. Інформація не може бути змінена при зберіганні або при посиланні між відправником й відповідним одержувачем без виявлення змін.
3. Не відмова (англ. Non-repudiation). Відправник інформації не може на більш пізньому етапі заперечувати свої наміри в створенні та передачі інформації.
4. Аутентифікація. Відправник та одержувач можуть підтвердити особистість один одного в походженні/призначення інформації.

Криптографія використовується в різних сферах, включаючи комунікації, банківські операції, електронну комерцію, комп'ютерну безпеку, розподілені технології збереження і передачі даних таких як блокчейн, та багато іншого.

Одним із важливих розділів криптографії є протоколи доведення знання (***zero knowledge protocols***). Протоколи доведення знання - це методи, які дозволяють довести факт про наявність певної інформації, не розголошуючи саму цю інформацію. Ці протоколи дозволяють двом або більше сторонам спілкуватися і взаємодіяти таким чином, щоб вони могли переконатися одна в одній про наявність певних фактів, не розголошуючи саму інформацію.

Протоколи доведення знання знаходять широке застосування в криптографії та комп'ютерній безпеці. Одне з найпопулярніших використань полягає в аутентифікації користувачів без розголошення їхніх особистих даних. Наприклад, у системах електронного голосування можна використовувати протоколи доведення знання (zero-knowledge proof) для переконання в тому, що голосувач має право голосу та вже не голосував, але не розкриваючи саму вибрану кандидатуру.
Протоколи доведення знання також знайшли застосування в області блокчейн-технологій та криптовалют. Вони можуть допомогти підтвердити правильність виконання певних умов або операцій без розголошення конфіденційної інформації, такої як баланс рахунку або вірність транзакцій.

Одним з найвідоміших прикладів протоколу доведення знання є протокол Фіата-Шаміра. В цьому протоколі доказувач збирається довести перевіряючій стороні, що він знає певне секретне значення x, яке задовольняє певне відношення R. Доказувач виконує певні обчислення та генерує доказ, який відправляє перевіряючій стороні. Перевіряюча сторона може перевірити цей доказ, не знаючи самого секретного значення x, шляхом перевірки відповідності обчислень та відношення R.

Протоколи доведення знання є складними і математично вимогливими, і їх впровадження вимагає ретельного аналізу безпеки та доведення коректності. Невірна реалізація або використання слабких параметрів може привести до компрометації безпеки та розкриття конфіденційності. Важливою частиною реалізації протоколів є використання *еліптичних кривих*. Їх використання є одним з найдорожчих в zk-SNARK та потребує ретельної перевірки безпеки.

Я представляю ефективну оптимізацію для алгоритмів на основі zk-SNARK і формул для еліптичних кривих. Множення та

мультиекспоненціювання точок на еліптичних кривих доказу з нульовим знанням є дорогим і тривалим процесом. Стандартне множення $[n] * P$, використовуючи ендоморфізм і skew представлення, щоб запобігти виняткам, які вимагали б обмежень. Розглядається множення точок на скаляр в еліптичних кривих в різних варіаціях точок: арифметичні дії в афінних і проєктивних точках. Ми також покажемо, як зробити множення та мультиекспоненціювання безпечнішими за допомогою розширення поля. Наші результати тестування та запропоновані методи сприяють покращенню найсучаснішої продуктивності обчислень еліптичних кривих.

У своїй сфері я зіткнулася з численними проблемами при ефективній реалізації множення точок еліптичної кривої за скаляром і знайшла ряд рішень, які допомогли мені їх вирішити. Слід зазначити, що для еліптичних кривих простого порядку і порядку більше двох, реалізація множення може відрізнятися.

Рядом досліджень ми прийшли до висновку, що найефективніше множення в zk-Snarks відбувається через ендоморфізм. На цей час ми працюємо з трьома кривими: простими кривими BN256, secp256k1 і багатофакторною кривою BLS12-381. Криві, які ми використовуємо, повинні мати точки порядку 3, інакше оптимізація та трюки, які ми використовуємо з генераторами, зазнають невдачі.

РОЗДІЛ 1. Zero-knowledge proof та властивості

1. Загальні поняття

1.1 Прувер та Верифікатор.

Важливо встановити поняття, що таке прувер та верифікатор, оскільки це буде надалі зустрічатися у визначенні zero-knowledge proof та Snark's.

Прувер — це (іноді прихована або трансцендентна) механізм надання доказу. В той час як **Верифікатор** виконує роль перевірки доказу згенерованим прувером. Як і в математиці, так і на практиці процедура перевірки пруфа верифікатором вважається простішою ніж процес створення пруфа прувером.

Асиметрія між складністю завдання верифікації та складністю завдання доведення теореми відображена в класі складності $\mathcal{NP}$, який можна розглядати як клас систем доказів. Кожна мова $L \in \mathcal{NP}$ має ефективну процедуру верифікації для доказів тверджень форми "$x \in L$ ". Слід зауважити, що кожна $L \in \mathcal{NP}$ характеризується поліноміально-розпізнаваним у часі відношенням R_L, такими що:

$$L = \{ x: \exists y, (x, y) \epsilon\, R_L \}, \tag{1.1}$$

і $(x, y) \epsilon\, R_L$ тільки тоді коли $|y| \leq poly(|x|)$ (доказ не повинен бути надто великим порівняно з твердженням до якого ми створюємо пруф). Отже, у цьому контексті, R_L це механізм за допомогою якого ми асоціюємо кожне твердження $x \in L$ з його пруфом y , що задовільняє $(x, y) \epsilon\, R_L$. Отже, вірні твердження (тобто, $x \in L$) та лише ці мають докази в цій системі доказів. Зверніть увагу, що процедура верифікації є "легкою" (тобто, поліноміального часу), тоді як винахід доказів може бути "складним" (якщо дійсно $\mathcal{NP}$ не міститься в $\mathcal{BPP}$).

Визначення. (грубе визначення zero-knowledge) Нехай $(\mathcal{NP}, \mathcal{V})$ буде деякою інтерактивною системою для деякої мови $\mathcal{L}$. Ми позначаємо $View_V^P(x)$ як випадкову змінну, яка описує вміст випадкової стрічки V^* та повідомлення, які V^* отримує від $\mathcal{P}$ під час спільного обчислення на загальному вводі x. *Система доведення* або ще як називають *система аргументів* з пре написаним прувером $\mathcal{P}$ та пре написаним $\mathcal{V}$ верифікатором для мови $\mathcal{V}$ буде вважатися нульовим знанням (zero-knowledge), якщо для кожної ймовірнісної інтерактивної машини з поліноміальним часом V^* існує ймовірнісний алгоритм поліноміального часу M^* (який цілком може залежити від V^*), називається симулятором, такий що $\forall x \in$ L розподіл виходу $M^*(x)$ симулятора є "нерозрізним" від $View_V^P(\mathcal{P}(x), V(x))$. (система доказу (P, V) є нульовим знанням, якщо для будь-якого "зовнішнього" перевірника V∗ можна знайти алгоритм M∗, який може відтворити вид V∗ на вводі x без необхідності взаємодії з P, і що вигляд V∗ від M∗ і P неможливо відрізнити з обчислювальної точки зору.)

Примітка. Важливо підмітити, що для кожного V^* взаємодіючого з $\mathcal{P}$, а не тільки для V, повинен існувати ("ідеальний") симулятор M^*. Цей симулятор, хоч і не має доступу до інтерактивної машини $\mathcal{P}$, проте він вміє імітувати роботу інтерактивної машини V^* з $\mathcal{P}$. Той факт, що такі симулятори існують, означає, що V^* не отримує ніяких знань від $\mathcal{P}$ (оскільки один і той же вихід може бути згенерований без доступу до $\mathcal{P}$).

Як було сказано вище ZKP – це протокол за допомогою якого одна сторона (доказуючий) може довести іншій стороні (перевіряючому), що дане твердження є істинним, тоді як доказуючий уникає передачі будь-якої інформації, крім факту, що твердження справді вірне.

ZKP має задовольняти трьом властивостям:

- Повнота: якщо твердження істинне, чесний верифікатор буде переконаний у цьому факті чесним прувером.

- Обґрунтованість. Якщо твердження хибне, жоден прувер не зможе переконати чесного верифікатора, що воно істинне, за винятком певної невеликої ймовірності.
- Нульове знання. Якщо твердження вірне, то жоден верифікатор не дізнається нічого, крім того факту, що твердження істинне.

  Перші два пункти є властивостями більш загальних інтерактивних систем. Третє – робить доведення нульовим знанням.

Zero-knowledge proof поділяються на інтерактивні доведення з нульовими знаннями (interactive zero-knowledge proofs) та неінтерактивне (non-interactive zero-knowledge proofs). Інтерактивні доведення з нульовими знаннями вимагає взаємодії між особою (або комп'ютерною системою), яка підтверджує доказ. Не інтерактивні доведення з нульовими знаннями не вимагають взаємодії між довідовачем і верифікатором. Ці криптографічні системи широко використовуються у блокчейні та в таких протоколах як zk-STARK, zk-SNARK.

1.2 Інтерактивні та неінтерактивні доведення з нульовим розголошенням

Криптографія з нульовим розголошенням є потужним інструментом, який дозволяє одній стороні (довірливій) довести іншій стороні (верифікатору), що вони знають деяку таємну інформацію, не розкриваючи саму цю інформацію. Існують два основних типи таких доведень: інтерактивні та неінтерактивні.

1.3 Інтерактивні доведення з нульовим розголошенням (Interactive Proof Systems)

Визначення інтерактивної системи доказів прямо стосується двох обчислювальних завдань, пов'язаних із системою доказів: «створення» доказу та перевірка достовірності доказу. Ці завдання виконуються двома різними сторонами, які називаються прувером і верифікатором, які взаємодіють одна з одною. У деяких випадках взаємодія може бути дуже простою і, зокрема, односпрямованою (тобто прувер надсилає текст, який називається доказом, верифікатору). Загалом, взаємодія може бути складнішою і мати форму опитування той що перевіряє того кого перевіряють.

Інтерактивне доведення з нульовим розголошенням — це протокол, в якому прувер та верифікатор взаємодіють один з одним через серію повідомлень. Цей процес, зазвичай, включає кілька раундів взаємодії, після яких верифікатор або визнає доведення дійсним, або відхиляє його.

Визначення 2. (Інтерактивні доведення з нульовим розголошенням) – для мови $L \subseteq \{0,1\}^*$ та пари інтерактивних машин Тюрінга (P, V), де P має необмежену обчислювальну потужність, а V – ймовірнісний поліноміальний час, тоді пара (P, V) називається інтерактивною системою доказів мови L з нульовим доказом, якщо істині три наступні умови:

- Повнота: для будь-якого публічного $x \in L$ і полінома p(·),

$$\Pr[(P, V)(x) = 1] \geq 1 - \frac{1}{p(|x|)}, \tag{1.2}$$

- Надійність: для будь-якого публічного $x \notin L$ та будь-якої інтерактивної машини Тюрінга P' полінома $p(\cdot)$,

$$\Pr[(P', V)(x) = 1] < 1 - \frac{1}{p(|x|)}, \tag{1.3}$$

- Нульові знання: для кожної ймовірнісної поліноміальної машини Тюрінга V^* існує ймовірнісний поліноміальний алгоритм часу M^* такий, що для будь-якого $x \in L$ виконується:

$$(P, V^*)(x) \approx M^*(x), \tag{1.4}$$

1.4 Не інтерактивні доведення з нульовим розголошенням(Non-interactive Proof Systems).

Неінтерактивні доведення з нульовим розголошенням відрізняються від інтерактивних тим, що вони потребують лише одного повідомлення від прувера до верифікатора. Це досягається за допомогою так званого "*proof system*", який генерує глобальні параметри, які використовуються усіма учасниками.

Модель неінтерактивних доказів здається ближчою за духом до моделі $\mathcal{NP}$ -доказів, ніж до загальних інтерактивних доказів. У певному сенсі модель $\mathcal{NP}$-proof розширена, дозволяючи пруверу та верифікатору посилатися на звичайний випадковий рядок. В іншому випадку, як і у випадку $\mathcal{NP}$ -доказів, взаємодія є мінімальною (тобто односпрямованою: від прувера до верифікатора). Таким чином, у наведеному нижче визначенні і прувер, і верифікатор є звичайними ймовірнісними машинами, які, крім загального введення, також отримують рівномірно розподілений (загальний) еталонний рядок. Однак, для простоти, ми подаємо визначення для випадку, коли жодна з цих машин не отримує допоміжний вхід. Верифікатор також отримує як вхідні дані, вироблені пруверо.

Визначення (Не інтерактивні доведення з нульовим розголошенням) – для пари ймовірнісних машин Тюрінга (P, V), в яких P – ймовірнісний поліноміальний час, а V –детермінований поліноміальний час, тоді пара (P, V) називається не інтерактивною системою доказів мови L з нульовим доказом, якщо істині три наступні умови:

- Повнота: для будь-якого публічного $x \in L$ і полінома $p(\cdot)$,

$$\Pr\left[V\left(x, R, P(x, R)\right) = 1\right] \geq 1 - \frac{1}{p(|x|)}, \tag{1.5}$$

- Надійність: для будь-якого публічного $x \notin L$ та будь-якої інтерактивної машини Тюрінга P' полінома $p(\cdot)$,

$$\Pr\left[V\left(x, R, P'(x, R)\right) = 1\right] \geq 1 - \frac{1}{p(|x|)}, \tag{1.6}$$

- Нульові знання: для будь-якого $x \in L$ існує ймовірнісний поліноміальний алгоритм часу M такий, що виконується:

$$V(x) = \left(x, R \in \{0,1\}^{c(|x|)}, P(x, R)\right) \approx M^*(x)_{x \in L}, \tag{1.7}$$

Використання IPZK та NIZK відрізняється залежно від конкретних потреб і обмежень системи. Обидва типи протоколів мають свої переваги: IPZK зазвичай є більш легкими для розуміння і реалізації, тоді як NIZK забезпечують більшу ефективність та зручність для великих систем.

1.5 Які приклади та випадки використання доказів з нульовим знанням?

Докази з нульовим рівнем знань можна використовувати для захисту конфіденційності даних у різноманітних випадках використання, наприклад:

- Блокчейн: прозорість публічних блокчейнів, таких як біткойн та Ethereum, забезпечує публічну перевірку транзакцій. Однак це також передбачає незначну конфіденційність і може призвести до деанонімізації користувачів. Докази з нульовим знанням можуть забезпечити більшу конфіденційність публічних блокчейнів. Наприклад, криптовалюта Zcash заснована на короткому неінтерактивному аргументі знання з нульовим знанням (Zk-SNAKR), типу криптографічного методу з нульовим знанням. Ethereum також працює із доказами zk-SNARK з моменту оновлення візантії у 2017 році.
- Фінанси: ING використовує ZKP, які дозволяють клієнтам довести, що їх секретний номер знаходиться у відомому діапазоні. Наприклад, заявник на іпотеку може довести, що його дохід знаходиться в допустимому діапазоні, не повідомляючи точну його зарплату.
- Онлайн-голосування: ZKP можуть дозволити виборцям голосувати анонімно та підтвердити, що їхній голос був включений до остаточного підрахунку.
- Аутентифікація: ZKP можна використовувати для автентифікації користувачів без обміну секретною інформацією, як-от паролі.
- Машинне навчання: ZKP можуть дозволити власнику алгоритму машинного навчання переконувати інших у результатах моделі, не розкриваючи жодної інформації про саму модель ML.

РОЗДІЛ 2. Вступ до zk-SNARKs.

Zk-SNARKs, або "**Zero-Knowledge Succinct Non-Interactive Argument of Knowledge**", є формою невідкладного доказу з нульовим розголошенням, яка стала особливо популярною у сфері криптовалют через їх використання в Zcash та інших приватних валютах. Основна ідея полягає в тому, щоб дозволити одній стороні довести, що вона має відповідь на певне питання, не розкриваючи саму відповідь.

Zk-SNARK як зазначено вище складається з трьох алгоритмів (S, P, V), де S – сетап, P – прувер, V – верифікатор. Генерація пруфа прувером має бути швидкою, а його верифікація має бути швидкою та короткою. Обидва алгоритми повинні мати логарифмічну швидкість відносно розміру сьоркета.

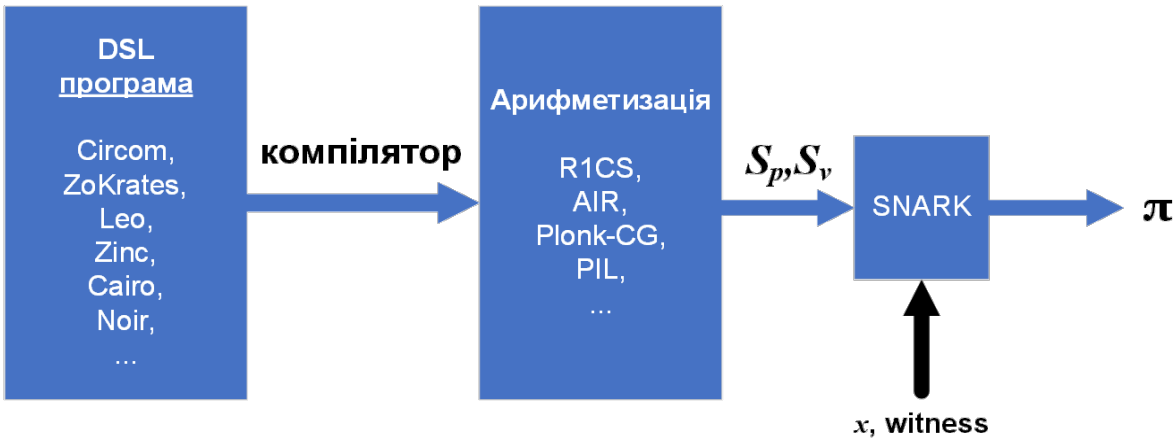


Рисунок 2.1. Загальна будова роботи zk-SNARK.

Як це працює?

Уявімо, що у нас є програма, яку позначимо як C, яка приймає два параметри: $C(x, w)$. Параметр x являє собою публічне значення, а w є секретним вхідним значенням. Мета отримати таке публічне значення x, щоб можна було переконатися в тому, що провіряючий знає секретне значення w, таке, що $C(x, w) == true$.

Припустимо, Бобу дається хеш H певного значення, і він хоче мати доказ того, що Аліса знає значення s, хеш якого дорівнює H. Зазвичай Аліса доводить це, передаючи s Бобу, після чого Боб обчислює хеш і перевіряє, що це дорівнює H. Проте припустимо, що Аліса не хоче розкривати цінне значення s Бобу, і замість цього вона просто хоче доказати, що знає значення s. Для цього вона може використати zk-SNARK. Описавши програму Аліси за допомогою функції $C(x, w)$ ми бачимо, що Алісі потрібно створити доказ того, що вона володіє таким s, що $C(H, w) == true$, не розкриваючи значення s. Це загальна проблема, яку вирішують zk-SNARK.

Zk-SNARK складається з трьох алгоритмів G, P, V, які визначаються наступним способом: генератор ключів G приймає секретний $lambda$ і програму C та генерує два публічних ключі: доказуючий ключ pk та ключ перевірки vk . Ці ключі являються загальнодоступними параметрами, з якими необхідно створити тільки один раз для даної програми C.

Доказуючий алгоритм P приймає в якості вхідних значень: ключ pk, публічне значення x і секретне значення w. Алгоритм генерує доведення $prf = P(pk, x, w)$ того, що доказувач знає секретне значення w і що секретне значення задовольняє програмі C.

Перевіряючий алгоритм V обчислює $V(vk, x, prf)$ результатом якого буде $true$, якщо доведення буде вірним, і $false$ в іншому випадку. Таким чином, ця функція повертає $true$, якщо перевіряючий знає, що секретне значення w задовільняє $C(H, w) == true$.

Зверніть увагу на секретний параметр $lambda$, який використовується в генераторі. Цей параметр іноді ускладнює використання zk-SNARK у реальних програмах. Причина цього в тому, що кожен, хто знає цей параметр, може створити підроблені докази. Точніше, з огляду на будь-яку програму C і публічного значення x, людина, яка знає $lambda$, але не знає

секретного w, може створити доведення $fake\ prf$, на який алгоритм $V(vk,\ x,\ fake\ prf)$ поверне $true$.

2.1 Арифметичні схеми (Arithmitic circuits).

Арифметичні схеми є ключовими компонентами в побудові zk-SNARKs (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge). Для більш детального вивчення цієї теми, необхідно розуміти, що таке арифметичні схеми, як вони використовуються в zk-SNARKs і чому вони є важливими. **Арифметичні схеми** є основою обчислювальних моделей. Вони являти собою собою направлені ациклічні графи, де вузли представляють арифметичні операції, а крайні вузли представляють вхідні та вихідні значення цих операцій. Арифметична схема кодує обчислення як послідовність операцій, що виконуються над вхідними даними. Вхідні значення схеми, відомі як вхідні змінні, проходять через серію арифметичних операцій, що визначаються структурою схеми.

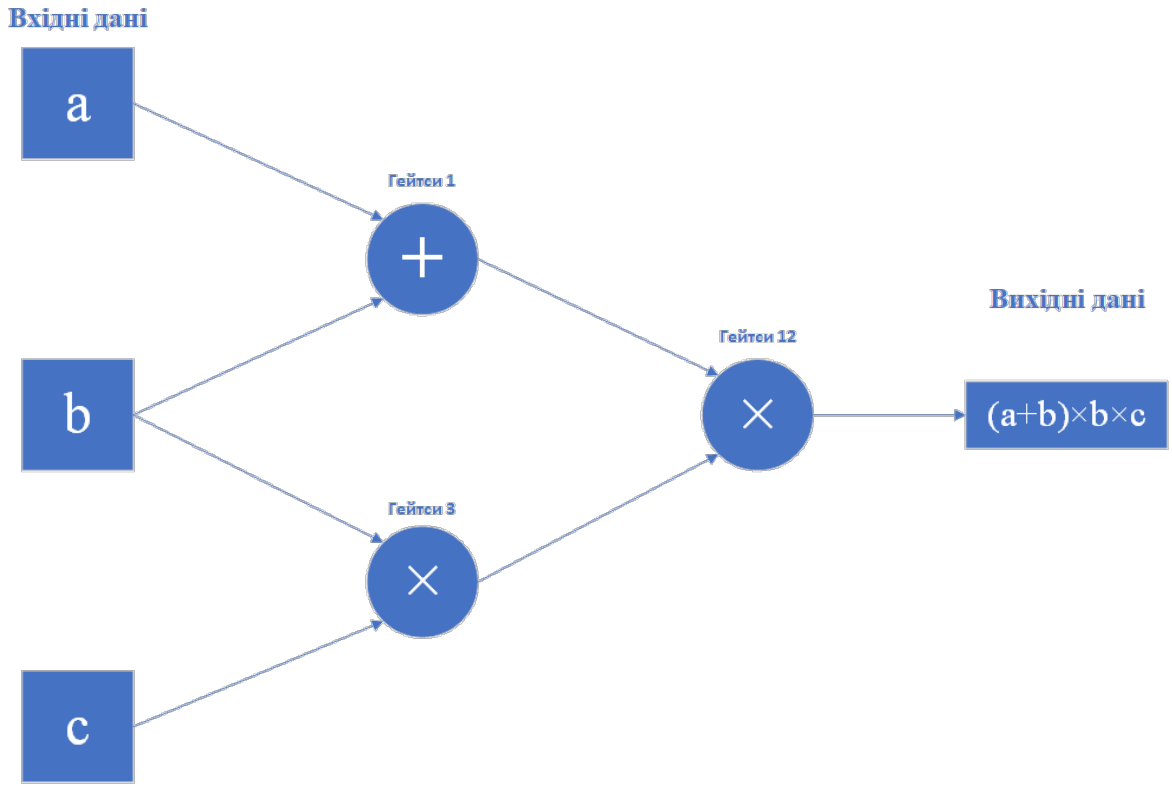


Рисунок 2.2. Структурна схема гейтів.

***Визначення*. Арифметичні схеми (Arithmitic circuits).** Візьмемо деяке скінченне поле $\mathbb{F} = \{0, \dots, p-1\}$ для деякого $p > 2$, тоді:

$$C: \mathbb{F}^n \rightarrow \mathbb{F}, \tag{2.1}$$

Буде називатися арифметичною схемою. Фактично це спрямований ациклічний граф, де внутрішні вузли називаються "*гейтси*" і вони позначаються арифметичними операціями $+, -$ чи $\times$, а вхідні дані позначаються в основному вхідними змінними $1, x_1, \dots, x_n$.

За прикладом, що на картинці вище можна прийти до висновку, що схема в основному визначає багатовимірний поліном. Уявіть, що арифметична схема це як рецепт для обчислення конкретного полінома.

Примітка. $|\boldsymbol{C}| =$ кількість гейтсів в C. У арифметичних схемах, використовуваних в zk-SNARKs, зазвичай використовуються два основних типи гейтсів: додавання (addition gates) та множення (multiplication gates). Вони відповідають операціям додавання та множення в арифметичному полі. Важливо зазначити, що в більшості випадків ці гейтси не працюють з простими числами, а з елементами скінченного поля. Це означає, що вони використовують спеціалізовані операції додавання та множення, які відповідають правилам скінченного поля, а не звичайних чисел.

2.2 Система аргументів(argument systems)

Системи аргументів бувають двох типів: інтерактивні та неінтерактивні.

Нехай дано арифметична схема $C(x, w) \rightarrow \mathbb{F}$, де x – публічне значення в $\mathbb{F}^n$, а w - секретне вхідне значення в $\mathbb{F}^m$.

Нижче наведений приклад роботи двох систем інтерактивної та неінтерактивної і принцип роботи ви можете зрозуміти, що він схожий.

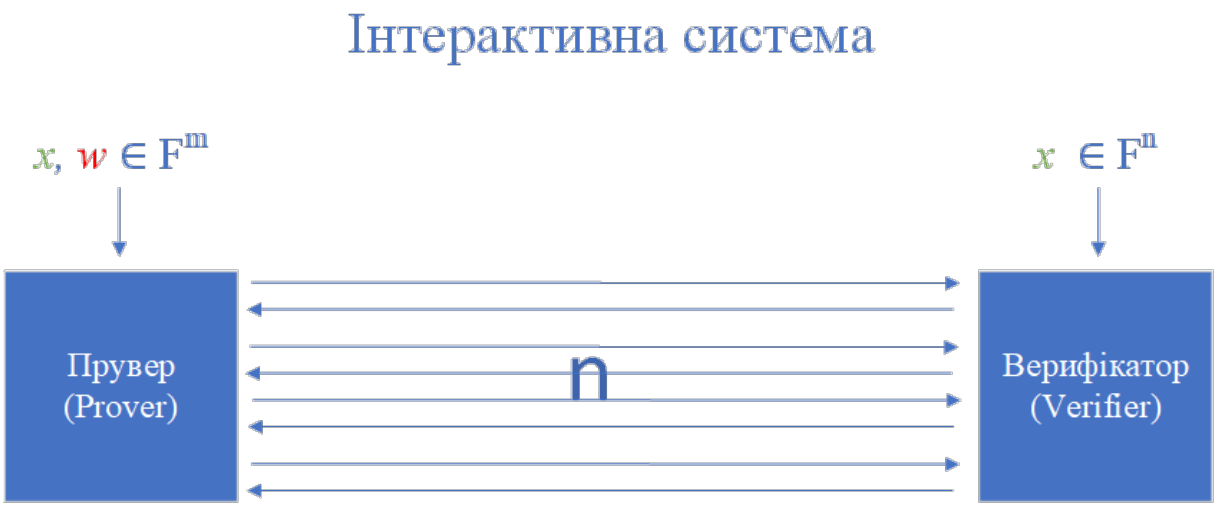


Рисунок 2.3. Робота інтерактивної системи

Що потрібно для неінтерактивної системи це пре процес який ми називаємо сетапом: $S(C) \rightarrow$ публічний параметер (S_p, S_v)

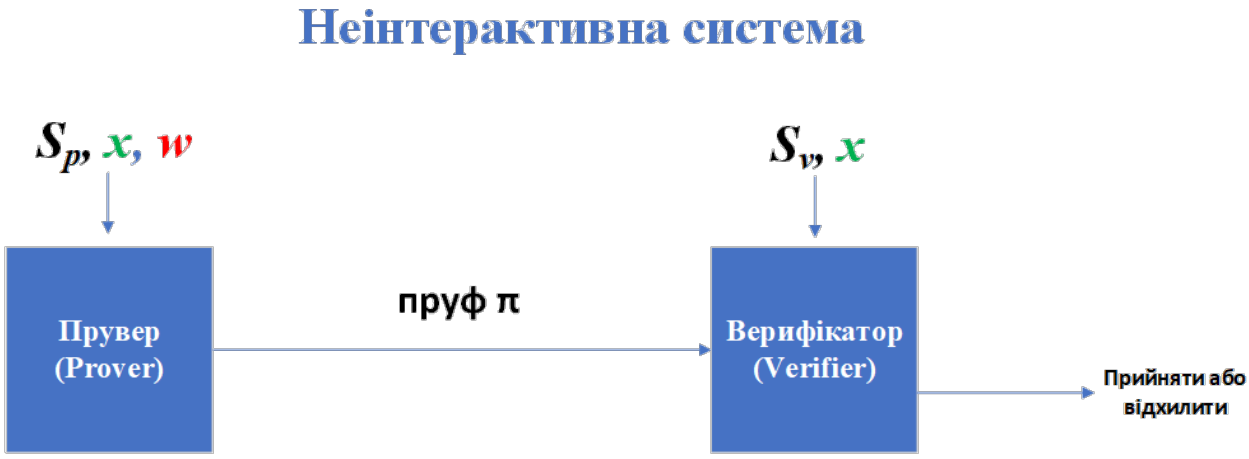


Рисунок 2.4. Робота неінтерактивної системи

2.3 Коміти(commits) або криптографічне зобов'язання.

Термін "*commit*" належать до схем криптографічного зобов'язання. Цей концепт важливий у доказах з нульовим знанням, включаючи zk-

SNARKs, оскільки вони часто покладаються на схеми зобов'язання для своєї роботи.

Криптографічне зобов’язання дозволяє одній стороні зобов’язатися на вибране значення. Коміти створені таким чином, що сторона яка їх зобов’язала, а потім відкрила для доказу того, що значення не змінилося з моменту зобов’язання. У zk-SNARKs, ці схеми зобов'язання часто використовуються як частина процесу створення доказу. Прувер може зобов'язатися на певні значення (зберігаючи їх в таємниці), виконувати обчислення на цих значеннях, а потім використовувати зобов'язання як частину доказу, що ці обчислення були виконані правильно. Це можна зробити без відкриття фактичних значень, зберігаючи властивість "нульового знання" доказу.

В zk-SNARKs коміти позначаються як $commit(m,r) \to com$, де r – випадкове значення.

Примітка. Для криптографічних зобов’язань зазвичай використовують фіксовані хеш функції $H: M \times R \to C$.

Ось наочний приклад використання криптографічних зобов’язань: Виберемо сімейство функцій $F = \{f: X \to Y\}$.

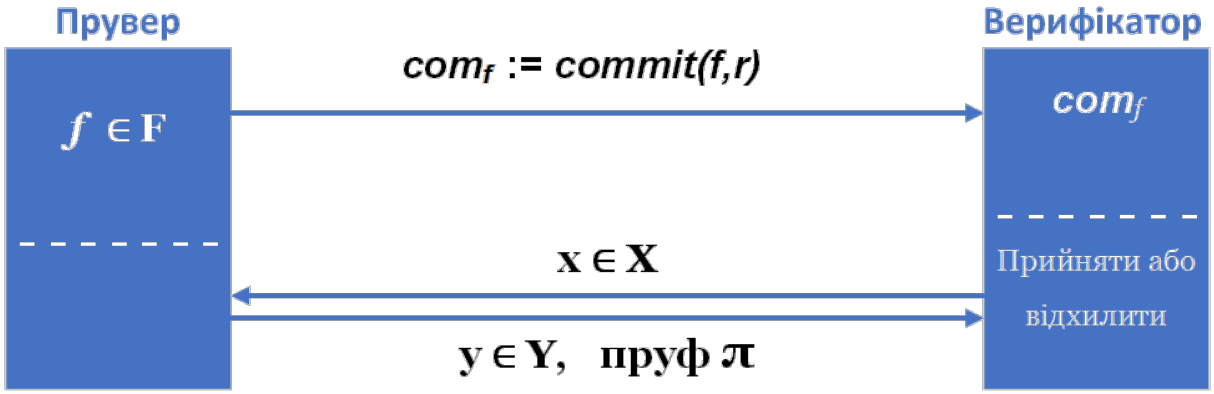


Рисунок 2.5 Взаємодія прувера і верифікатора.

2.4 Еліптичні криві

Криптографія з еліптичною кривою — це форма криптографії з відкритим ключем, яка спирається на математику еліптичних кривих. Причина, по якій цей тип криптографії настільки популярний і широко використовується, полягає в тому, що він забезпечує високий рівень безпеки з відносно невеликими ключами, які є швидшими та ефективнішими у використанні, ніж криптографії без ECC. Це також робить ECC більш масштабованим, що важливо в сучасному світі, де кількість даних, які ми передаємо та отримуємо, зростає з приголомшливою швидкістю.

Еліптичні криві є центральними для конструкції та роботи zk-SNARK, і вони застосовуються в кількох областях:

1. Еліптичні криві, зручні для створення пар: zk-SNARK значною мірою покладаються на використання еліптичних кривих із властивістю, відомою як «парінг». Парінг — це спеціальна операція на еліптичних кривих, яка відображає дві точки на кривій у третю, використовуючи білінійну, невироджену та обчислювану функцію. Еліптичні криві, зручні для створення пар, дозволяють ефективно обчислювати парінг, і саме ця ефективність робить zk-SNARK практичними. Ці пари використовуються в побудові як доказу, так і перевірки алгоритму zk-SNARK.

2. Формулювання проблеми: в контексті zk-SNARK однією з ключових проблем є проблема дискретного логарифма еліптичної кривої (ECDLP). ECDLP — це обчислювальна задача знаходження такого цілого числа d, що $Q = dP$ для заданих точок P і Q на еліптичній кривій. Уявна складність цієї проблеми лежить в основі безпеки багатьох криптографічних систем на основі еліптичних кривих, включаючи zk-SNARK.

3. Етап налаштування: у zk-SNARK етап налаштування включає генерацію криптографічних параметрів, які використовуються на наступних етапах створення доказів і перевірки. Це включає математичні операції над еліптичними кривими, такі як піднесення до степеня та об'єднання в пари.

4. Генерація доказів: на етапі генерації доказів засіб перевірки виконує обчислення на еліптичних кривих для створення zk-SNARK. Це включає такі операції, як додавання точок і множення на кривій.

5. Перевірка доказів: аналогічно, під час етапу перевірки перевіряючий виконує обчислення на еліптичних кривих, щоб підтвердити правильність zk-SNARK. Це передбачає такі операції, як поєднання точок на кривій.

У двох словах, еліптичні криві використовуються в zk-SNARK для створення складної математичної структури, яка дозволяє верифікатору переконати верифікатор у тому, що вони знають значення, яке задовольняє певне поліноміальне рівняння, не розкриваючи нічого про саме значення. Ці криві використовуються в базових математичних конструкціях, які роблять zk-SNARK неінтерактивними, нульовими знаннями та лаконічними.

Визначення. Еліптична крива – це множина точок який задовольняє невироджений поліном двох змінних. Якщо K поле тоді еліптична крива має задовольняти таку властивість:

$$E: \{(x, y) \in K \times K | f(x, y) \in F[x, y], f(x, y) = 0\}, \qquad (2.2)$$

де $f(x, y)$ конкретний невироджений поліном від x, y порядку 3. Поліном є невиродженим, якщо він має різні корені. Якщо поле K має характеристику, відмінну від 2, 3, то за допомогою перетворення змінних можна показати, що E має таку ж поведінку, як еліптична крива вигляду:

$$E: y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad (2.3)$$

Рівняння вище називається рівнянням Веєрштраса для еліптичної кривої.

Примітка. Рівняння Веєрштраса може бути спрощене до виду:

$$E: y^2 = x^3 + Ax + B, \quad (2.4)$$

Покладаючи $a_1 = a_2 = a_3 = 0,\ a_4 = A$, і $a_6 = B$.

2.5 Проєктивне представлення

Визначення. Проєктивні однорідні координати ми покладемо $x = X/Z$ і $y = Y/Z$, і якщо K поле тоді рівняння Веєрштраса еліптичної кривої стане:

$$E: Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 ,$$

(1)

Точка нескінченності буде представлятися як $(0, \theta, 0)$ для деякого $\theta \epsilon \mathbb{K}\ \{0\}$. Тепер афінна точка, яка записувалася як (x_1, y_1) буде записуватися як $(\theta x_1, \theta y_1, \theta)$ для деякого $\theta \epsilon \mathbb{K}\ \{0\}$. Проєктивна точка $(X_1, Y_1, Z_1) \neq \mathcal{O}$ буде відповідати афінній точці $(X_1/Z_1, Y_1/Z_1)$.

2.6 Ендоморфізм

Нехай E – еліптична крива визначена над скінченним полем $\mathbb{F}_q$. Точку нескінченності будемо позначати $\mathcal{O}$. Для будь-якого $n \geq 1$, група раціональних точок $\mathbb{F}_{q^n}$ на E будемо позначати $E(\mathbb{F}_{q^n})$.

Визначення. Ендоморфізмом еліптичної кривої E є відображення

$$\phi : E \to E, \quad (2.5)$$

Що задовольняє $\phi(\mathcal{O}) = \mathcal{O}$ [7]. Якщо відображення визначено над полем $\mathbb{F}_q$, то й ендоморфізм ϕ також визначений над полем $\mathbb{F}_q$.

Примітка. ϕ є груповим гомоморфізмом $E(\mathbb{F}_q)$ і також для $E(\mathbb{F}_{q^n})$ для будь-якого $n \geq 1$.

2.7 Множення точок еліптичної кривої на скаляр з використанням ендоморфізму

Нехай E – еліптична крива визначена над скінченним полем $\mathbb{F}_q$, $P \in E(\mathbb{F}_q)$ буде точкою простого порядку n. Нехай ϕ ендоморфізм визначений над полем $\mathbb{F}_q$. Припустимо, що характеристичний многочлен ϕ має корінь λ по модулю n — оскільки характеристичний поліном ендоморфізму має другий степінь, ми очікуємо, що приблизно половина всіх кривих матиме корінь за модулем n. Відображення ϕ діє на $\langle P \rangle$ як відображення множення $[\lambda]$.

Описаний метод буде корисними, якщо обчислення ϕ коштує менше, ніж обчислення приблизно $(\log 2\, n)/3$ подвоєння точки. На практиці ми очікуємо, що алгоритм буде застосовано, коли вартість φ менше (скажімо) 5 дублювання.

Основною ідеєю [8] здійснити множення за ендоморфізмом, де скаляр k розбиваємо на $k = k_1 + k_2\lambda \, mod \, n$, де $k_1 k_2 \in [0, \lceil\sqrt{n}\rceil]$, тоді ми отримаємо

$$kP = (k_1 + k_2\lambda)P, \tag{2.6}$$

$$= k_1 P + k_2(\lambda P), \tag{2.7}$$

$$= k_1 P + k_2\phi(P), \tag{2.8}$$

Тепер за допомогою алгоритму [8], який називається *одночасне багатократне множення*:

Алгоритм 1. Одночасне багатократне множення(англ. Simultaneous multiple point multiplication)

Вхід: $w, u = (u_{t-1}, \dots, u_1, u_0)_2, v = (v_{t-1}, \dots, v_1, v_0)_2, P, Q$.

Вихід: $uP + vQ$.

1. Обчислюємо $iP + jQ$, для кожного $i, j \in [0, 2^w - 1]$.
2. Розбиваємо на вікна вхідні скаляри $u = (u^{d-1}, \dots, u^1, u^0)$ та $v = (v^{d-1}, \dots, v^1, v^0)$, $d = \lceil t/w \rceil$.
3. $R \leftarrow \mathcal{O}$.
4. For i from $d - 1$ down to 0 do

$$R \leftarrow 2^w R$$

$$R \leftarrow R + \left(u^i P + v^i Q\right).$$

5. Повертаємо (R).

Оскільки довжини бітів k_1 і k_2 у формулі (2.7) становлять половину довжини бітів k, ми могли б очікувати значного прискорення, оскільки ми усунули значну кількість подвоєння точок за рахунок кількох додаваннь точок. Точний аналіз буде продемонстровано в практичній частині.

2.8 Практична частина

В цьому розділі буде виконана практична частина по оптимізації. Будуть використані такі методи як skew представлення чисел, скалярне множення точок еліптичної кривої з використанням ендоморфізму, прекомпутація: Дерево селектів, хеш сет та оперативна пам'ять з використання Waksman, проєктивне представлення та точки над розширеними полями з використанням генераторів для безпеки системи zero-knowledge proof. Код написаний мовою програмування Rust відповідно до бібліотеки franklin-crypto, гілка: pre-release.

2.9 Skew representation

Ми використовуємо деякі хитрощі для скалярної оптимізації. *Skew представлення* або ще як можна перефразувати зображення чисел десяткової системи в систему двох чисел $\{-1, 1\}$. Навіщо нам потрібен скаляр у такому форматі? Класичний метод double-and-add на практиці використовує такі операції на ЕС як дублювання і додавання.

```
let bits = bit_representation(s) # вектор бінарних бітів (from LSB to MSB) big-endian.
   let res = 𝒪 # точка на нескінченності
   let temp = P #
   for bit in bits:
       if bit == 1:
           res = res + temp # point add
       temp = temp + temp # double
   return res
```

як показано вище, дублювання відбувається завжди, але от алгоритм додавання тільки тоді, коли біт дорівнює 1. Для своєї оптимізації я

використовую метод double-and-add написаний в моїй вітці pre-release в бібліотеці franklin-crypto за алгоритмом з [9].

Алгоритм 2. Speeding up Some Elliptic Curve Operations in Affine Coordinates [9] ($2P + Q$ над $\mathbb{F}_p$)
Вхід: $P = (x_1, y_1), Q = (x_2, y_2) \in E_p$ **Вихід**: $(x_3, y_3) = 2P + Q$ $d = (x_2 - x_1)^2 \times (2x_1 + x_2) - (y_2 - y_1)^2,$ $D = d \times (x_2 - x_1), I = D^{-1},$ $\lambda = (y_2 - y_1) \times dI, \lambda_1 = -\lambda + 2y_1 \times (x_2 - x_1)^3 I,$ $x_3 = (\lambda_1 - \lambda) \times (\lambda_1 + \lambda) + x_2,\ y_3 = (x_1 - x_3) \times \lambda_1 - y_1.$

Лема. Існує алгоритм, який обчислює $2P + Q = (x_3, y_3)$ який коштує $1I + 7M + 2S$, точки задано $P = (x_1, y_1)$ і $Q = (x_2, y_2)$ на еліптичній кривій $y^2 = x^3 + ax + b$ над великим характеристичним полем F_p.

В контексті оцінки процедур в криптографії на еліптичних кривих, I, M та S часто означають наступне:

- **I**: Inversion (Інверсія) - операція обернення елементу в полі.
- **M**: Multiplication (Множення) - операція множення двох чисел.
- **S**: Squaring (Піднесення до квадрата**)** - операція піднесення числа до квадрата.

$1I + 7M + 2S$ тоді означає, що для виконання обчислення $2P + Q = (x_3, y_3)$ потрібно виконати одну операцію інверсії, сім операцій множення та дві операції піднесення до квадрату.

Оптимізація полягає ось в чому:

В контексті zero-knowledge потрібно було б кожного разу за допомогою методу select() в залежності від біта $\{0, 1\}$ вибирати між

звичайним множенням та оптимізованим, а це коштувало б нам одного констреїнту (гейтсу). Тому представлення скаляра через skew дозволяє на кожній ітерації додавати точку P чи віднімати її не створюючи констреїнт перевірки.

Визначення. Skew representation. Нехай задано $k_0, k_1, \dots, k_{n-1}$, де всі цифри $k_i \in \{1, -1\}$ для $0 \leq i < 1$. Довжина цифрового представлення кожного під скаляра k_j фіксована і $l = \lceil \log_2 r/m \rceil + 1$, де r - порядок простої підгрупи.

Алгоритм 3. Skew representation.

Вхід: $x = [x_0,\ x_1, \dots, x_n] = \sum_{i=0}^{l} x_i 2^i, x_i \in \{0, 1\}$
Вихід: $y = [y_{-1}, y_0, y_1, \dots, y_n] = -y_{-1} + \sum_{i=1}^{l}(1 - 2y_i)2^i,$
For $i =$ n downto -1 do
If $i =$ n then
$y_n = 0,$
end if
$y_i = 1 - x_{i+1},$
end for

Ідея з Skew взяти з книги [10], тільки там автор пропонує представлення десяткового числа як ternary representation, а саме {-1, 0, 1}.

2.10 Алгоритм множення точок еліптичної кривої на скаляр з використанням ендоморфізму

Нехай точка $P \in E(\mathbb{F}_q)$, що містить ендоморфізм степеня 2 ϕ. Скалярне множення $k * P$ має вигляд $k_1P + k_2\phi(P)$. Якщо k_1, k_2 мають приблизно половину бітову довжину вихідного скаляра k, то слід очікувати усунення половини кількості подвоєнь за допомогою техніки одночасного мультискалярного множення Штрауса-Шаміра. Таким чином, метод є особливо корисний для прискорення у випадку, коли базова точка P є змінною, відомому як скалярне множення зі змінною основою. Алгоритм 4 реалізує таку хитрість, яка вимагає в середньому лише ℓ подвоєнь і 0.75ℓ додавань.

Звичайно, це має сенс, коли k_1 та k_2 мають розмір $l/2$ і коли k_1, k_2 та $Q = [\lambda]P$ є доступні за розумною ціною. Я поєднала множення Штрауса-Шаміра (Алгоритм 1) та хитрість з skew представленням та отримала:

Алгоритм 4. Подвійне скалярне множення з використанням трюку Штрауса-Шаміра з урахуванням Skew representation.

Вхід: $u = \sum_{i=0}^{l-1} u_i 2^i, v = \sum_{i=0}^{l-1} v_i 2^i, (u_i, v_i) \in \mathcal{S}^2,\ (P, Q) \in \mathcal{E}^2,\ P \neq \pm Q$
Вихід: $R = \pm[u]P \pm [v]Q$
Перед обчислити (Precomputation)$W_{i,j} = \pm[i]P \pm [j]Q, \forall (i, j) \in \mathcal{S}^2$,
Покладемо $R = W_{u_{l-1}, v_{l-1}}$
For $i = \mathrm{l} - 2$ downto 0 do
$R \leftarrow [2]R$
$R \leftarrow R + W_{u_i, v_i}$
End for

Варто зауважити, що на кожному етапі ітерації ми не маємо умови if, тобто точка додається в будь-якому випадку. Важливу нішу в оптимізації займає Precomputation, тому що і тут є низка варіантів ефективних так і менш ефективних.

Попередні обчислення (Precomputation)

Ми використовуємо однаковий інтерфейс для всіх трьох методів. Процес обчислення лінійної комбінації всіх можливих точок, однак, коли ми записуємо результат у пам'ять.

Ось код програми, яка відповідає за перебір усіх можливих лінійних комбінацій (Код написано згідно з бібліотекою franklin-crypto, гілка: pre-release):

```
        // using Selector tree makes sense only if there are more than 1 element
        let mut entries = entries.to_vec();
        let mut workpad : Vec<(u64, PointWrapper<'a, E, G, T>)> = vec![(1, initial)];
        let mut pos = 0;

        for (_is_first, _is_last, elem) in entries[1..].iter_mut().identify_first_last() {
            let mut new_working_pad = Vec::with_capacity(workpad.len() << 1);
            for (addr, acc) in workpad.iter_mut() {
                let msb = get_bit_at_pos(*addr, pos);
                if msb {
                    new_working_pad.push(
                        (extend_addr(*addr, pos, !msb), acc.sub_mixed(cs, elem)?));
                    new_working_pad.push(
                        (extend_addr(*addr, pos, msb), acc.add_mixed(cs, elem)?));
                }
                else {
                    new_working_pad.push(
                        (extend_addr(*addr, pos, !msb), acc.add_mixed(cs, elem)?));
                    new_working_pad.push(
                        (extend_addr(*addr, pos, msb), acc.sub_mixed(cs, elem)?));
                }
            };
            pos += 1;
            workpad = new_working_pad       }
```

Рисунок 2.6. Код програми (перебір усіх можливих лінійних комбінацій)

Відбувається наступне: ми отримуємо на вході набір точок $P_1, P_2, \dots, P_n$. З цих точок ми генеруємо лінійні комбінації типу $P_1 - P_2 +$

$... + P_n$, і присвоюємо адреси цій лінійній комбінації $0100 ... 00$, потім $P_1 + P_2 - P_3 + ... + P_n$ - $001 ... 00$ і так далі, поки не переберемо всі можливі комбінації. Як показано вище в коді, ми записуємо кортеж $(адреса, точка)$ у вектор. Наступним кроком нам потрібно довести, що точка, вибрана за адресою, є правильною. У нас є три підходи для цього.

2. 11 Пам’ять з вільним доступом з використанням Ваксман алгоритму (Ram with Waksman).

Для таких попередніх обчислень ми використовуємо щось на кшталт пам'яті з довільним доступом. Робота пам'яті з випадковим доступом полягає у тому, що спочатку ми попередньо обчислюємо всі можливі точки $2^w - 1$, де w - ширина вікна. Потім записуємо ключ, який $k \in [0, 2^{2w}]$. У загальному випадку ми доводимо коректність пам'яті, з якої ми витягли точки. Тут ми виконуємо деяку попередню обробку. У цьому фрагменті коду ми групуємо елементи. Ми створюємо 4 масиви, два з яких складаються з невідсортованих елементів, а другий - з відсортованих. Будь ласка, зверніть увагу, що ми маємо 4 масиви, 2 - відсортовані і 2 - невідсортовані, через обмеження місткості типу, в якому ми записуємо точки. Для того, щоб компактно перевести точку до нашого типу, ми використовуємо стиснення точки стиснення. Далі нам потрібно довести, що перестановка виконана правильно. В оперативній пам'яті ми сортуємо точки в порядку зростання кількості адрес. Після цього доводимо, що сортування виконано правильно, використовуючи такий підхід: https://hal.inria.fr/inria-00072871/document. Мережі Waksman - це тип сортувальних мереж, які спеціально використовуються в маршрутизації та телекомунікаціях. Їх назвали на честь Авіезрі Фраенкел Ваксмана, який запропонував цю модель. Основний принцип роботи:

Мережі Waksman - це ідеальний приклад " oblivious" мережі. Термін " oblivious" тут означає, що мережа направляє пакети від вхідних до бажаних вихідних, не маючи жодних знань про дані, які вона передає, тобто маршрут, яким проходить пакет, не залежить від вмісту пакета.

Останній етап перевірки потрібно обмежити, щоб елементів масиву було не більше, ніж розмір пам'яті. Ми порівнюємо елементи в відсортованій таблиці і якщо попередній елемент і поточний відрізняються, то додаємо одиницю до лічильника. В кінці ми генеруємо обмеження, яке полягає в тому, що константа (розмір пам'яті) і лічильник дорівнюють.

2.12 HashSet

У цій версії ми також використовуємо оперативну пам'ять, але обмежуємо її за допомогою рівності поліномів виду:

$$\prod(x + a_i) = \prod(b_i * y), \qquad (2.9)$$

Де x та y - коміти функції. У бібліотеці franklin-crypto ми використовуємо rescue хеш-функцію. a_i та b_i - відсортований та невідсортований вектор відповідно. Для останніх елементів масиву не більше розмір пам'яті - див. попередній алгоритм. Перевірка правильності сортування масиву однакова для обох методів. В основному, ми вводимо акумулятор, який додає по одному балу кожного разу, коли попередній і наступний елементи відрізняються. Але ми доводимо, що накопичувач дорівнює постійній величині $2^w - 1$.

```
    fn enforce_correctness_of_sorted_packed_queries<CS>(&mut self, cs: &mut CS)-> Result<(),
SynthesisError>
    where CS: ConstraintSystem<E>
    {
```

```
        let limit = Num::Constant(u64_to_ff((1u64 << self.address_width) - 1));
        let mut counter = Num::zero();
        let iter =
self.sorted_packed_elems_0.iter().zip(self.sorted_packed_elems_1.iter()).identify_first_last(
);
        let mut el_0_prev = Num::zero();
        let mut el_1_prev = Num::zero();

        for (is_first, _is_last, (el_0_cur, el_1_cur)) in iter {
            if !is_first {
                let is_equal_0 = Num::equals(cs, &el_0_prev, &el_0_cur)?;
                let is_equal_1 = Num::equals(cs, &el_1_prev, &el_1_cur)?;
                let both_are_equal = Boolean::and(cs, &is_equal_0, &is_equal_1)?;
                counter = counter.conditionally_increment(cs, &both_are_equal.not())?;
            }

            el_0_prev = *el_0_cur;
            el_1_prev = *el_1_cur;
        }
        counter.enforce_equal(cs, &limit)
    }
```

Рисунок 2.7. Код програми

2.14 Дерево селектів

Призначення деревоподібного селектора полягає у наступному: дано n точок $P_1, P_2, \dots, P_n$ (в афінних або проєктивних координатах), ми хочемо зберегти і згодом вибрати лінійні комбінації виду $\pm P_1, \pm P_2, \dots, \pm P_n$. Як випливає з формули вище, ключем будуть скалярні біти. У прикладі нижче показано дерево, яке буде сформовано з розміром ключа 2. Ми використовуємо select для вибору правильної гілки та її обмеження. Всього буде зроблено 6 select: 3 select для координати X і 3 select для координати Y. Однак, ми можемо використати формулу симетрії заперечення для точок еліптичної кривої: $-P(x, y) = P(x, -y)$

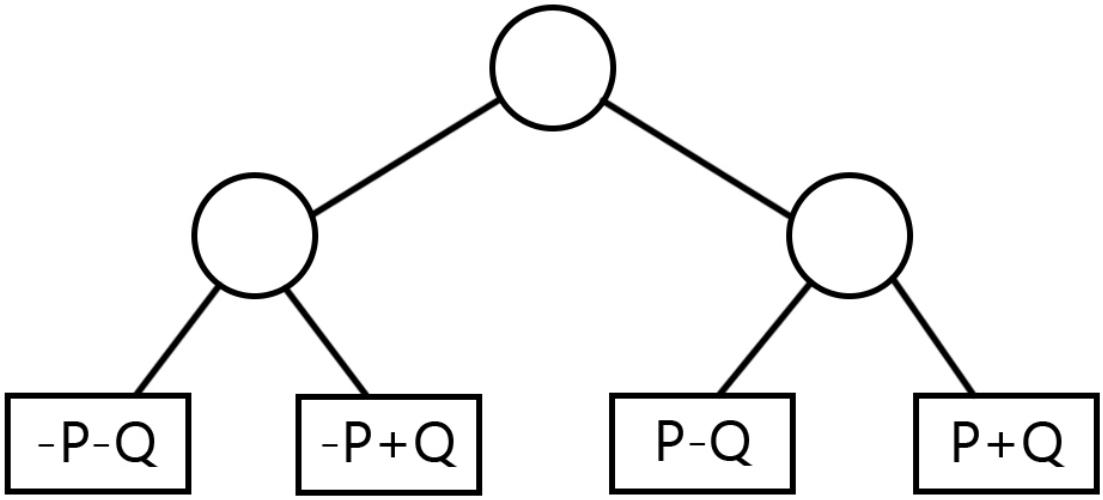


Рисунок 2.8. Дерево селектів.

Таким чином, ми доводимо, що точка знаходиться у правильному ключі. У бібліотеці franklin-crypto, завдяки черзі виникають особливості: якщо останні біти істинні, то ми переходимо до іншого дерева.

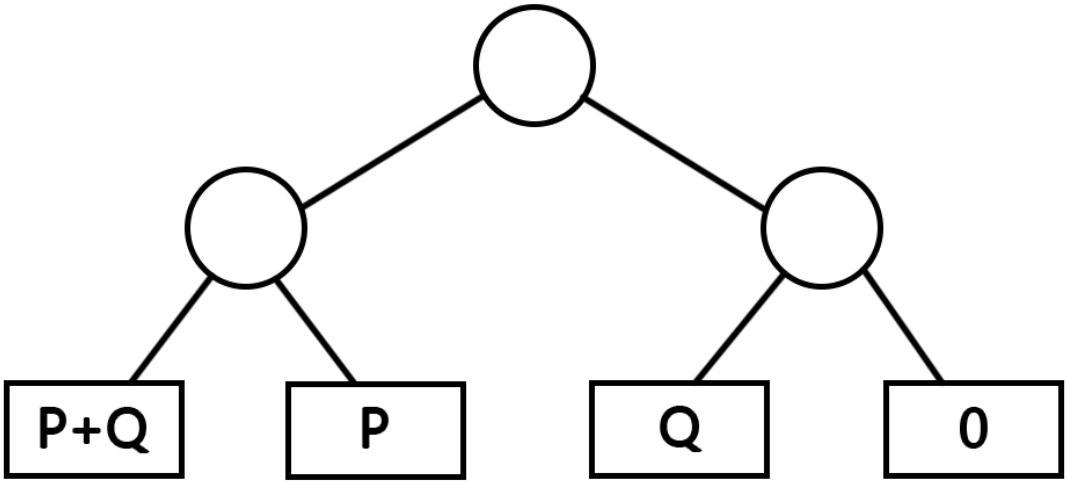


Рисунок 2.9. Дерево селектів.

Враховуючи особливість skew representation. Останній біт, який відповідає додавати точку чи ні, тому маємо таке дерево.

2.15 Проєктне представлення точки

В афінній формі кожна точка еліптичної кривої має 2 координати, наприклад (x, y), у новій проективній формі кожна матиме 3 координати, як (X, Y, Z) з обмеженням, що Z ніколи не дорівнює нулю. Пряме відображення $(x, y) \rightarrow (xz, yz, z)$, для будь-якого ненульового z (зазвичай для зручності вибирається рівним 1). Зворотне відображення задається $(X, Y, Z) \rightarrow (X/Z, Y/Z)$, якщо Z відмінне від нуля. Навіщо потрібні проєктивні точки у zk-Snark? Коли ми написали вільні від винятків формули, нам потрібно передати правильно, що точка може знаходитись на нескінченності. Арифметика, яку ми використовуємо для афінної точки, не передбачає що ми можемо мати точку на нескінченності як вхідні дані, тому виникне помилка і ми не зможемо згенерувати доведення. Використання арифметики на проєктивній кривій буде безпечним тому, що ми можемо коректно порахувати точку, а потім загорнути в точку $(0,0)$ з відміткою $point\ is\ infinity == true$ чи $false$. Як показано у наведених нижче тестах, ці операції будуть дорогими. Варто додати, що проєктивні точки ми використовуємо тільки в попередніх обчисленнях з використанням функцій sub_mixed чи add_mixed. Ці функції особливі, бо вони дозволяють додавати чи віднімати проєктивні точки з афінними. на виході ми отримуємо точки в проєктивних координатах. Попередньо розклавши скаляр**,** обчислюється лінійна комбінація з використанням ендоморфізму.

Примітка. Exception free formulas або як було написано вище це формули без винятків, тобто коли немає умов, які передбачають “if”.

Примітка. Важливо сказати, що ідея з проєктивними точками та з точками над розширенням будуть працювати тільки для кривих простого порядку.

Примітка. Коли ми допускається винятки, накопичувач (acc- акомулятор до якого додається точка) може бути в основній групі - це буде просто довільна точка з невідомим дискретним логарифмом. Якщо ми прагнемо до вільності від виключень, то беремо offset_generator як точку, визначену на $E(\mathbb{F}_{q^2})$. Коли алгоритм у вигляді формули не містить виключень, ми говоримо, що він є повним.

2.16 Точка над розширеним полем

Нам також потрібно розширене поле над еліптичною кривою для правильної обробки точки нескінченності. Основною мотивацією використання точок над розширенням – арифметика буде дешевою. Використовуємо разом із генератором точок порядка 3. Проте перехід з точки над розширенням до афінних точок буде коштувати кожного разу 4 select(), що доволі дорого. Це і є основна причина чому в перед обчисленні використовуються проєктивні точки.

2.17 Трюк з генераторами

При множенні точки на скаляр і піднесенні точки до степеня ми використовуємо трюки з генераторами для найвищої безпеки. Припустимо, що ми виконуємо множення без генератора, тоді злочинець може вибрати таку точку, яка при множенні на скаляр стане нулем. Наша бібліотека не розглядає випадок, коли ситуація є вільною від винятків, тому ми просто не зможемо згенерувати доведення. В якості розв'язку використовується генератор зсувів *offset_generator* порядка 2. Ми добавляємо генератор на початку нашого множення і прив’язуємо до нього лічильник який буде рахувати кількість дублювань у множенні і в кінці, враховуючи кількість дублювань, віднімаємо генератор, який продублювали за лічильником. Якщо ми все ще маємо процес без виключень, то використовуємо точки над

розширеним полем і генератор третього порядку. Він використовується для скидання точок над розширенням. Щоб отримати кінцеву точку нам потрібно вирахувати генератор adj_offset – Генератор точок порядку 3, отже, оскільки ми використовуємо цей генератор безпосередньо для додавання точок, то в нас відсутні будь-які дублювання, отже ми маємо 3 можливих варіанти acc, acc + adj_offset, acc – adj, що належить E(F_p). Що легко встановлюється за допомогою селектів, яку все-таки точку відняти.

Всі подальші результати отримані на еліптичній кривій BN256, що є кривою простого порядку. Пораховані константні генератори для неї:

```
    let fp2_offset_generator_x_c0 = Fq::from_str(

"16300907393763553952797146753989960732368004674693012223284497678682995867793"
    ).expect("should parse");
    let fp2_offset_generator_x_c1 = Fq::zero();
    let fp2_offset_generator_y_c0 = Fq::zero();
    let fp2_offset_generator_y_c1 = Fq::from_str(

"18847519929951292720903138501492801907098041246451137057597277895538377195039"
    ).expect("should parse");

    let fp2_pt_ord3_x_c0 = Fq::zero();
    let fp2_pt_ord3_x_c1 = Fq::zero();
    let fp2_pt_ord3_y_c0 = Fq::zero();
    let fp2_pt_ord3_y_c1 = Fq::from_str(

"21888242871839275217838484774961031245859103671646299620740479376884814904650"
    ).expect("should parse");
```

Нижче зображений увесь процес повного множення точок еліптичної кривої на скаляр:

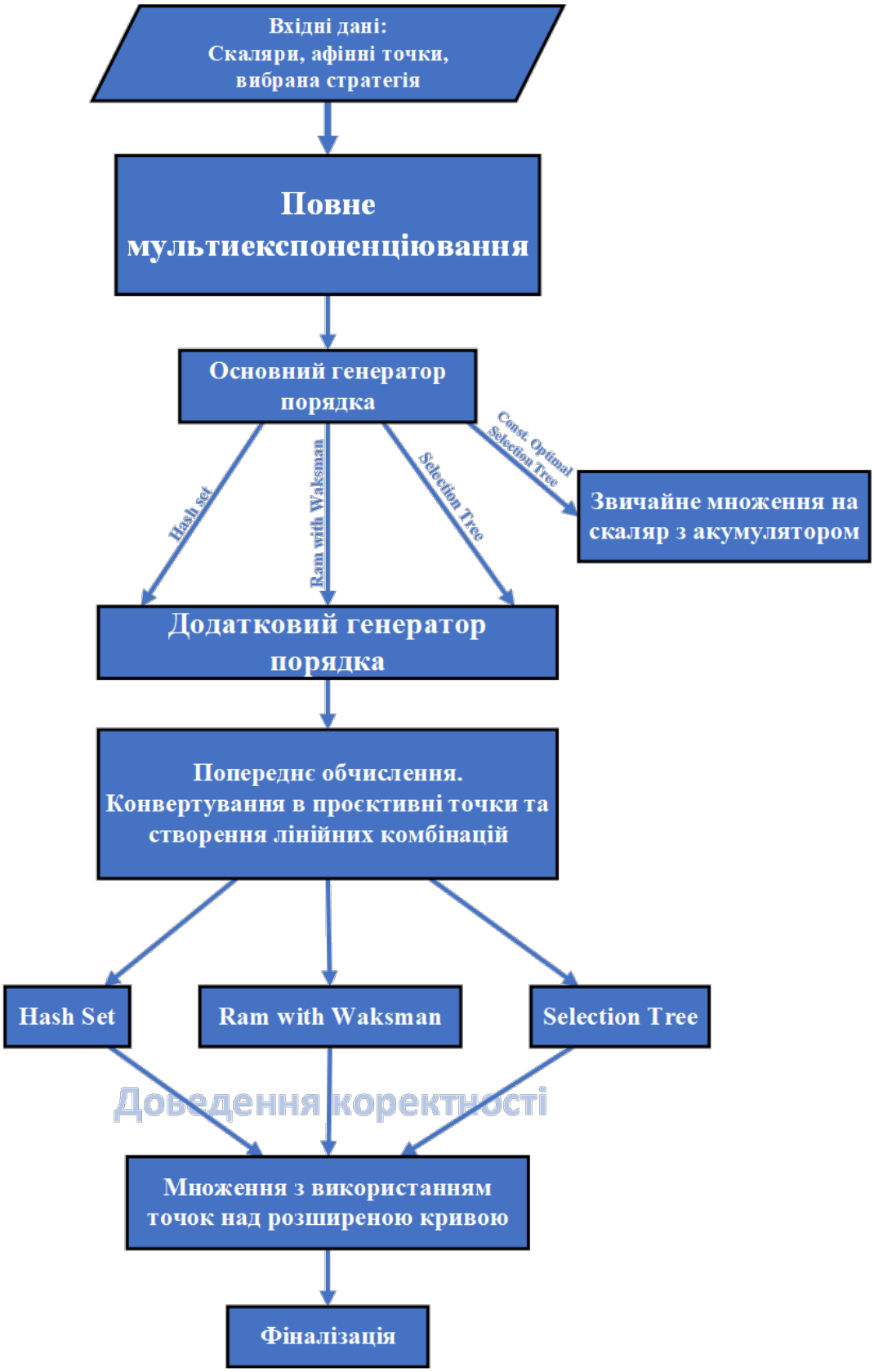


Рисунок 2.10. Схема мультиекспоненціювання

Результат досліджень

Таблиця 2.1

Метод/Стратегія	*Ширина вікна*	*Кількість констреїнтів*
Мультиекспоненціювання повне з Waksman стратегією.	*2*	*196325*
Мультиекспоненціювання повне з HashSets стратегією.	*2*	*220478*
Мультиекспоненціювання повне з SelectionTree стратегією.	*2*	*157595*
Мультиекспоненціювання не повне з Waksman стратегією.	*2*	*105731*
Мультиекспоненціювання не повне з HashSets стратегією.	*2*	*129884*
Мультиекспоненціювання не повне з SelectionTree стратегією.	*2*	*72207*
Мультиекспоненціювання повне з константною оптимальною стратегією.	*4*	*136695*
Мультиекспоненціювання не повне з константною оптимальною стратегією.	*4*	*53645*
Множення на скаляр повне.	*-*	*135973*
Множення на скаляр не повне.	*-*	*53645*
Звичайне множення на скаляр	*-*	*195344*

Висновок

Zero-knowledge protocol є загально вживаним та доволі новим криптографічним протоколом. Кожного року з’являються численні статі, які допомагають вдосконалювати та розвивати цю технологію. Сама технологія Zero-knowledge proof, я вважаю, ще дуже мало розвинена і її можна багато де ще застосувати окрім так популярного блокчейну. Сам протокол, як було сказано, насправді одна велика машина, де еліптичні криві виконують роль мотора. Ми використовуємо такі галузі математики як теорія чисел, теорія ймовірності, проєктивну геометрію, алгебру та навіть теорію ігор (наприклад консенсус алгоритми) тощо. В роботі показано алгоритми на основі пруф системи Plonk, проте на практиці використовуються й інші пруф системи, де, наприклад деякі з низ працюють на основі криптографії на решітках. Plonk, в основному, використовує арифметику в полі та еліптичні криві. Важливість кривих третього порядку в блокчейні вже давно доказана. (Наприклад: електронні підписи також працюють на еліптичних кривих).

В даній роботі було оптимізовано множення та мультиексподенціювання точок еліптичних кривих на скаляр в середовищі протоколу Zero-knowledge proof, які потребують особливої реалізації. Було використано різноманітні алгоритми та підходи для порівняння ефективного множення, а також трюки для покращення безпеки. Було встановлено, що множення точок еліптичних кривих на скаляр буде найефективнішим саме за допомогою ендоморфізму. До цього ж множення за ендоморфізму ми застосували різноманітні трюки, які дозволили максимально ефективно застосувати алгоритм.

Результати дослідження винесені в таблиці показують, який з підходів дає найменшу кількість гейтсів. Висновком експерименту є, що найефективнішим підходом є мультиексподенціювання з SelectionTree

стратегією. Також варто зауважити, що повний алгоритм, тобто алгоритми з exception free формулами, коштують набагато більше ніж звичайні. На це впливають підходи з проєктивними точками, точками над розширенням та генератори. Такою буде ціною безпеки множення точок еліптичної кривої на скаляр з exception free formulas.

Хочу додати, що існує багато “сирих” ідей та алгоритмів, які висунули талановиті математики, проте надзвичайно важливо вміти цю ідею застосувати в тій чи іншій галузі і отримати позитивний результат, що я і успішно зробила. Саму цього мене і вчили на Прикладній математиці.

“Довіряй математиці, а не валідаторам” Matter Labs.

Використана література

[1] Blum, Manuel; Feldman, Paul; Micali, Silvio *(1988).* "Non-Interactive Zero-Knowledge and Its Applications*". Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC 1988). c. 103–112.*

[2] Justin Thaler (2021). "Proofs, Arguments, and Zero-Knowledge". *Foundations and Trends in Theoretical Computer Science. c. 33-71, 170-174, 280-288.*

[3] Nigel Smart (2004). "Cryptography: An introduction*". Mcgraw-Hill College*. ISBN: 0077099877; 9780077099879. c. 371-380

[4] Wu, Huixin; Wang, Feng (2014). "A Survey of Noninteractive Zero Knowledge Proof System and Its Applications". *The Scientific World Journal*. 2014: 560484. doi:10.1155/2014/560484.

[5] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer and Madars Virza(2013). "SNARKs for C :Verifying Program Executions Succinctly and in Zero Knowledge". *Part of the Lecture Notes in Computer Science book series* (LNSC,volume 8043). c.35-37.

[6] Oded Goldreich (2001). "Foundations of cryptography. Basic tools". *Publisher: Cambridge University Press*. ISBN: 9780521791724,0521791723. c. 184-246.

[7] J.H. Silverman (1985). "The Arithmetic of Elliptic Curves". *Springer-Verlag*.

[8] R.P. Gallant, J.L. Lambert, and S.A. Vanstone (2001). "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms". In J. Kilian, editor, Advances in Cryptology - CRYPTO, volume 2139 of LNCS, Springer.

[9] Wei Yu, Kwang Ho Kim , Myong Song Jo (2015). "New Fast Algorithms for Elliptic Curve Arithmetic in Affine Coordinates". Part of the Lecture Notes in Computer Science book series (LNSC,volume 9241)

[10] J.H. Silverman (2000). " An Introduction to Mathematical Cryptography". *Springer-Verlag*. Mathematics Subject Classification (2000): 94A60, 11T71, 14G50, 68P25

[11] Bruno Beauquier, Eric Darrot(1999). “On Arbitrary Waksman Networks and their Vulnerability”. *Bruno Beauquier, Eric Darrot. On Arbitrary Waksman Networks and their Vulnerability. RR-3788, INRIA. 1999. inria-00072871.*

Додаток А

Бібліотека franklin-crypto, branch: pre-release. Файл: sw_affine.rs, multiexp.rs, tests.rs.

Link: https://github.com/matter-labs/franklin-crypto/tree/pre-release

Функція для мультиекспоненціювання.

```
    #[track_caller]
    fn multiexp_impl<CS: ConstraintSystem<E>>(
        cs: &mut CS, scalars: &mut [FieldElement<'a, E, G::Scalar>], points: &mut
[Self],
        geometry: MultiExpGeometry, exception_free_version: bool
    ) -> Result<(AffinePoint<'a, E, G, T>, Boolean), SynthesisError>
    {
        assert_eq!(scalars.len(), points.len());
        let params = points[0].circuit_params;
        let offset_generator = Self::get_offset_generator(exception_free_version,
params);

        let mut acc = offset_generator.clone();
        let mut idx = 0;
        let mut num_of_doubles = 0;

        while idx < points.len() {
            let chunk_size = std::cmp::min(points.len() - idx, geometry.width);
            let chunk_geometry = MultiExpGeometry {
                strategy: geometry.strategy,
                width: chunk_size,
            };
            let (new_acc, num_of_doubles_in_chunk) = if chunk_size > 1 {
                match geometry.strategy {
                    MultiexpStrategy::HashSetsBasedRam |
MultiexpStrategy::WaksmanBasedRam => {
                        Self::multiexp_impl_chunk_processing::<CS, Memory<E, G,
T>>(
                            cs, &mut scalars[idx..idx+chunk_size],
&points[idx..idx+chunk_size], acc,
                            exception_free_version, chunk_geometry
                        )?
                    },
                    MultiexpStrategy::SelectionTree => {
                        Self::multiexp_impl_chunk_processing::<CS, TreeSelector<E,
G, T>>(
```

```
                            cs, &mut scalars[idx..idx+chunk_size],
&points[idx..idx+chunk_size], acc,
                            exception_free_version, chunk_geometry
                        )?
                    }
                }
            } else {
                points[idx].mul_by_scalar_impl_wth_accumulator(cs, &mut
scalars[idx], acc)?
            };
            acc = new_acc;
            num_of_doubles += num_of_doubles_in_chunk;
            idx += chunk_size;
        }

        Self::final_normalization(
            cs, acc, offset_generator, num_of_doubles, exception_free_version,
exception_free_version
        )
    }
```

Функція мультиекспоненціювання:

```
    #[track_caller]
    fn mul_by_scalar_impl_wth_accumulator<CS: ConstraintSystem<E>>(
        &mut self, cs: &mut CS, scalar: &mut FieldElement<'a, E, G::Scalar>, mut
acc: PointWrapper<'a, E, G, T>
    ) -> Result<(PointWrapper<'a, E, G, T>, usize), SynthesisError> {
        if scalar.is_constant() {
            unimplemented!();
        }
        let params = self.circuit_params;
        let aux_data = Self::compute_endo_aux_data(cs, self, scalar)?;
        let point_minus_point_endo = aux_data.point.sub_unequal_unchecked(cs,
&aux_data.point_endo)?;
        let mut point_plus_point_endo = aux_data.point.add_unequal_unchecked(cs,
&aux_data.point_endo)?;

        let mut acc = acc.add_mixed(cs, &mut point_plus_point_endo)?;
        let num_of_doubles = aux_data.point_scalar_decomposition[1..].len();
        let iter = aux_data.point_scalar_decomposition[1..].iter().zip(
            aux_data.point_endo_scalar_decomposition[1..].iter()
        ).rev();

        // if x = [x_0, x_1, ..., x_n] = /sum x_i 2^i - binary representation of x:
x_i /in {0, 1}
```

```
        // then x = [y_-1, y_0, y_1, ..., y_n] - skewed naf representation: where
y_i /in {0, 1}
        // x = -y_-1 + /sum_{i >= 1} (1 - 2* y_i) 2^i
        // algorithm for construction of skewed representation:
        // for -1 <= y < n: y_i = ~x_{i+1} = 1 - x_{i+1} and y_n = 0 (always)
        // indeed:
        // y = -y_-1 + /sum (1 - 2* y_i) 2^i = x_0 - 1 + /sum (2* x_{i+1} - 1) 2^i
+2^n =
        // = x - 1 - \sum_{i=0}^{n-1} 2^i + 2^n = x - 1 - (2^n - 1) + 2^n = x

        for (k1_bit, k2_bit) in iter {
            // selection tree looks like following:
            //
            //                          |true --- P + Q
            //         |true---k2_bit--|
            //         |               |false --- P - Q
            // k1_bit--|
            //         |
            //         |                |true --- -P + Q
            //         |false---k2_bit--|
            //                          |false --- -P - Q
            //
            // hence:
            // res.X = select(k1_bit ^ k2_bit, P-Q.X, P+Q.X)
            // tmp.Y = select(k1_bit ^ k2_bit, P-Q.Y, P+Q.Y)
            // res.Y = conditionally_negate(!k1, tmp.Y)
            let xor_flag = Boolean::xor(cs, &k1_bit, &k2_bit)?;
            let selected_x = FieldElement:: conditionally_select(
                cs, &xor_flag, &point_minus_point_endo.get_x(),
&point_plus_point_endo.get_x()
            )?;
            let tmp_y = FieldElement::conditionally_select(
                cs, &xor_flag, &point_minus_point_endo.get_y(),
&point_plus_point_endo.get_y()
            )?;
            let selected_y = tmp_y.conditionally_negate(cs, &k1_bit.not())?;
            let mut tmp = unsafe { AffinePoint::from_xy_unchecked(selected_x,
selected_y, params) };
            acc = acc.double_and_add_mixed(cs, &mut tmp)?;
        }

        // we subtract either O, or P, or Q or P + Q
        // selection tree in this case looks like following:
        //
```

```
        //                          |true --- O
        //          |true---k2_bit--|
        //          |               |false --- Q
        // k1_bit--|
        //          |
        //          |                |true --- P
        //          |false---k2_bit--|
        //                          |false --- P+Q
        //
        let k1_bit = aux_data.point_scalar_decomposition.first().unwrap();
        let k2_bit = aux_data.point_endo_scalar_decomposition.first().unwrap();
        let acc_is_unchanged = Boolean::and(cs, &k1_bit, &k2_bit)?;
        let mut tmp = AffinePoint::conditionally_select(cs, &k2_bit,
&aux_data.point, &point_plus_point_endo)?;
        tmp = AffinePoint::conditionally_select(cs, &k1_bit, &aux_data.point_endo,
&tmp)?;
        let skew_acc = acc.sub_mixed(cs, &mut tmp)?;
        acc = PointWrapper::conditionally_select(cs, &acc_is_unchanged, &acc,
&skew_acc)?;

        Ok((acc, num_of_doubles))
    }
    #[track_caller]
    fn multiexp_impl_chunk_processing<CS: ConstraintSystem<E>, S: Selector<'a, E,
G, T>>(
        cs: &mut CS, scalars: &mut [FieldElement<'a, E, G::Scalar>], points:
&[Self],
        mut acc: PointWrapper<'a, E, G, T>, exception_free_version: bool, geometry:
MultiExpGeometry
    ) -> Result<(PointWrapper<'a, E, G, T>, usize), SynthesisError>
    {
        assert_eq!(scalars.len(), points.len());
        let params = points[0].circuit_params;

        struct Multizip<T>(Vec<T>);

        impl<T> Iterator for Multizip<T>
        where T: Iterator,
        {
            type Item = Vec<T::Item>;

            fn next(&mut self) -> Option<Self::Item> {
                self.0.iter_mut().map(Iterator::next).collect()
            }
```

```
        }

        let should_add_adj_generator = !params.is_prime_order_curve &&
exception_free_version;
        let mut points_unrolled = Vec::with_capacity(points.len() << 1);
        let mut scalars_unrolled = Vec::with_capacity(points.len() << 1);
        let iter = scalars.iter_mut().zip(points.iter()).identify_first_last();
        for (is_first, _is_last, (mut scalar, point)) in iter {
            let aux_data = Self::compute_endo_aux_data(cs, point, &mut scalar)?;

scalars_unrolled.push(aux_data.point_scalar_decomposition.into_iter().rev());

scalars_unrolled.push(aux_data.point_endo_scalar_decomposition.into_iter().rev());

            let point_reg = if is_first && should_add_adj_generator {
                let x = G::from_xy_checked(params.fp2_pt_ord3_x_c0,
params.fp2_pt_ord3_y_c0).expect("valid point");
                let adj_generator = AffinePoint::constant(x, params);
                aux_data.point.add_unequal_unchecked(cs, &adj_generator)?
            } else {
                aux_data.point.clone()
            };
            points_unrolled.push(point_reg);
            points_unrolled.push(aux_data.point_endo);
        }

        let exception_free_combiner = exception_free_version &&
params.is_prime_order_curve;
        let mut combiner = Combiner::<E, G, T, S>::new(cs, &points_unrolled,
geometry, exception_free_combiner)?;
        let (mut initial, is_initial_point_at_infty, _) = combiner.get_initial();
        // I do not know how to get rid of this boilerplate code
        let mut acc = if exception_free_combiner {
            let to_add = Self::safe_conversion_to_ext(cs, &initial,
&is_initial_point_at_infty)?;
            let acc_as_ext = acc.get_as_ext_point();
            let res = acc_as_ext.add_unequal_unchecked(cs, &to_add)?;
            PointWrapper::AffineOverExtField(res)
        } else {
            acc.add_mixed(cs, &mut initial)?
        };

        for (_, is_last, bits) in Multizip(scalars_unrolled).identify_first_last()
{
```

```
            if !is_last {
                let (mut to_add, is_point_at_infty) = combiner.select(cs, &bits)?;
                acc = if exception_free_combiner {
                    let to_add = Self::safe_conversion_to_ext(cs, &to_add,
&is_point_at_infty)?;
                    let acc_as_ext = acc.get_as_ext_point();
                    let res = acc_as_ext.double_and_add_unequal_unchecked(cs,
&to_add)?;
                    PointWrapper::AffineOverExtField(res)
                } else {
                    acc.double_and_add_mixed(cs, &mut to_add)?
                };
            }
            else {
                let (mut to_add, is_point_at_infty) = combiner.select_last(cs,
&bits)?;
                acc = if exception_free_combiner {
                    let to_add = Self::safe_conversion_to_ext(cs, &to_add,
&is_point_at_infty)?;
                    let acc_as_ext = acc.get_as_ext_point();
                    let res = acc_as_ext.add_unequal_unchecked(cs, &to_add)?;
                    PointWrapper::AffineOverExtField(res)
                } else {
                    let acc_modified = acc.add_mixed(cs, &mut to_add)?;
                    PointWrapper::conditionally_select(cs, &is_point_at_infty,
&acc, &acc_modified)?
                };
            }
        }

        combiner.postprocess(cs)?;
        let limit = params.get_endomorphism_bitlen_limit();
        Ok((acc, limit - 1))
    }
    pub fn new<CS: ConstraintSystem<E>>(
        cs: &mut CS, entries: &[AffinePoint<'a, E, G, T>],
        geometry: MultiExpGeometry, exception_free_version: bool
    ) -> Result<Self, SynthesisError> {
        let first_elem = entries.get(0).expect("Entries must be nonempty");
        let params = first_elem.circuit_params;
        // for now we do not work over BLS12-381 but all it trvially extendable
        assert_eq!(params.is_prime_order_curve, true);

        let get_bit_at_pos = |num: u64, pos: usize| -> bool {
```

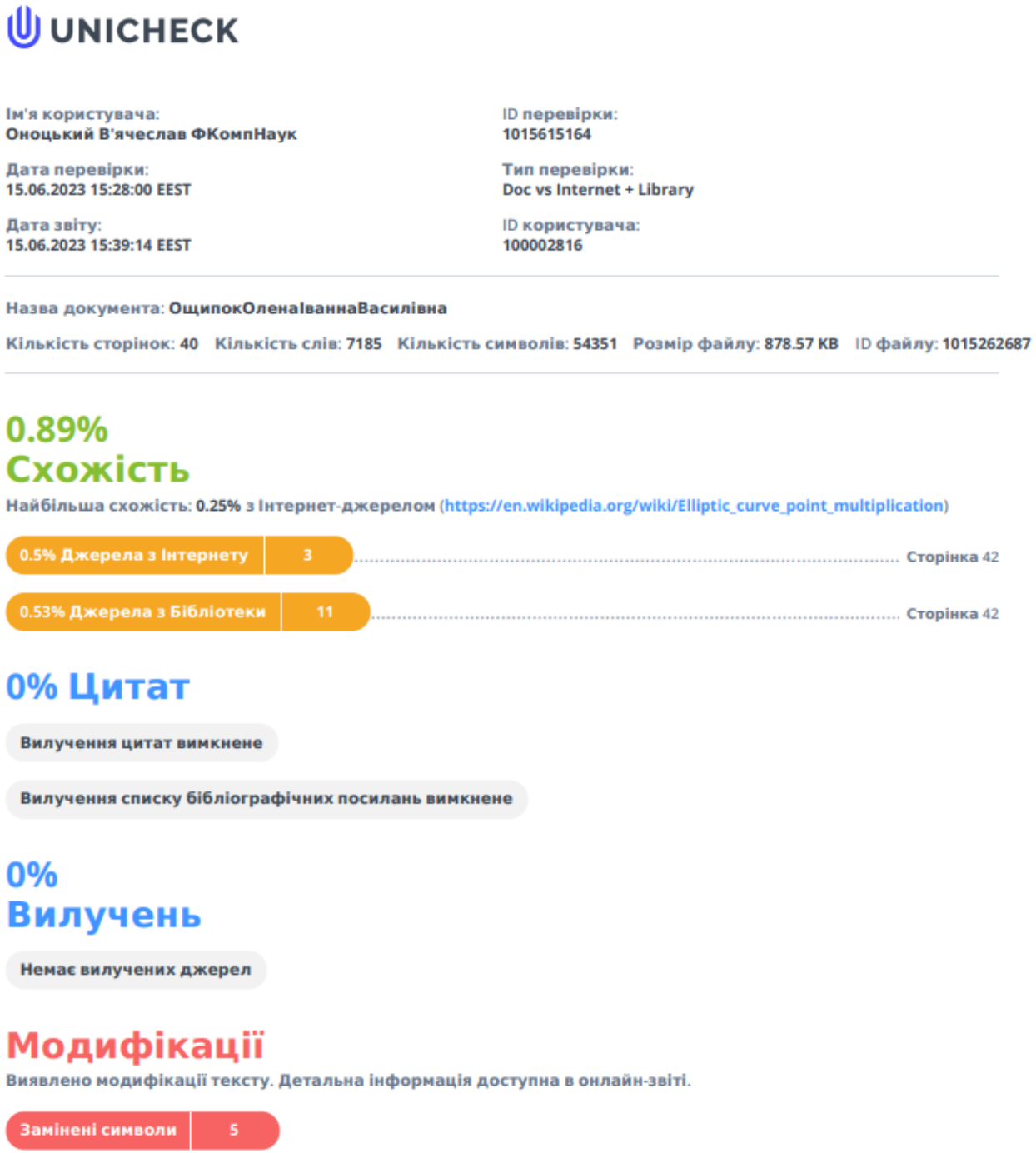
```
            (num >> pos) & 1 != 0
        };
        let extend_addr = |num: u64, pos: usize, msb: bool| -> u64 {
            if msb { (1u64 << (pos+1)) + num } else { num }
        };

        let initial = if exception_free_version {

PointWrapper::HomogeniousForm(ProjectivePoint::from(first_elem.clone()))
        } else {
            PointWrapper::AffineOverBaseField(first_elem.clone())
        };

        // using Selector tree makes sense only if there are more than 1 element
        let mut entries = entries.to_vec();
        let mut workpad : Vec<(u64, PointWrapper<'a, E, G, T>)> = vec![(1,
initial)];
        let mut pos = 0;

        for (_is_first, _is_last, elem) in
entries[1..].iter_mut().identify_first_last() {
            let mut new_working_pad = Vec::with_capacity(workpad.len() << 1);
            for (addr, acc) in workpad.iter_mut() {
                let msb = get_bit_at_pos(*addr, pos);
                if msb {
                    new_working_pad.push((extend_addr(*addr, pos, !msb),
acc.sub_mixed(cs, elem)?));
                    new_working_pad.push((extend_addr(*addr, pos, msb),
acc.add_mixed(cs, elem)?));
                }
                else {
                    new_working_pad.push((extend_addr(*addr, pos, !msb),
acc.add_mixed(cs, elem)?));
                    new_working_pad.push((extend_addr(*addr, pos, msb),
acc.sub_mixed(cs, elem)?));
                }
            };
            pos += 1;
            workpad = new_working_pad
        }
        assert_eq!(workpad.len(), 1 << (entries.len() - 1));

        let mut precompute = Vec::with_capacity(workpad.len());
        let mut initial_point_affine = AffinePoint::uninitialized(params);
```

```
        let mut initial_point_is_infty = Boolean::constant(false);
        let mut initial_point_proj = ProjectivePoint::zero(params);

        for (_is_first, is_last, elem) in workpad.into_iter().identify_first_last()
{
            let (addr, wrapped_point) = elem;
            let (point_affine, is_infty) =
wrapped_point.clone().convert_to_affine_or_uninitialized(cs, true)?;

            if is_last {
                initial_point_affine = point_affine.clone();
                initial_point_is_infty = is_infty;
                initial_point_proj = wrapped_point.convert_to_projective();
            }
            precompute.push((addr, point_affine));
        }

        let mut selector = S::new(geometry);
        selector.absorb_precompute(cs, precompute)?;

        Ok(Combiner {
            entries,
            selector_impl: selector,
            exception_free_version,
            initial_point_affine,
            initial_point_is_infty,
            initial_point_proj,
            params
        })
    }
```

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

СИСТЕМА ЗАПОБІГАННЯ ТА ВИЯВЛЕННЯ АКАДЕМІЧНОГО ПЛАГІАТУ

Довідка про оригінальність кваліфікаційної роботи за освітнім рівнем бакалавр



Експертна оцінка роботи науковим керівником:

Робота виконана самостійно та не містить відомостей без посилань на джерела.

Найбільш схоже джерело – Вікіпедія. «Множення точок еліптичної кривої»

Науковий керівник:	(підпис)	Денисов С.В. (ПІБ)
Оператор	(підпис)	Оноцький В.В. (ПІБ)

Відгук на кваліфікаційну роботу бакалавра на тему:
Оптимізація гаджет бібліотек для рекурсивних zk-Snarks

студентки 4-го курсу факультету комп’ютерних наук та кібернетики
Київського національного університету імені Тараса Шевченка
Ощипок Олени-Іванни Василівни

Технологія Zero-knowledge proof, якій присвячена дипломна робота, з'явилася у 1980-х роках як результат досліджень в галузі теорії чисел та комп'ютерної науки. Вона була винайдена Шафі Голдвассер, Сілвіо Мікалі та Чарлзом Ракоффом в MIT. Спочатку ZKP використовувалась у протоколах автентифікації, але з часом її потенціал у інших областях став очевидним.

Студентка докладно описала ідеї та методи, пов’язані з цією технологією, навела приклади реалізації та сценарії використання, зробивши акцент на поняттях zero-knowledge proof (доказу з нульовою інформацією), zk-Snark та оптимізації множення точок еліптичних кривих на скаляр в цих системах. Також в роботі розглянуто використання еліптичних кривих у цьому контексті.

В роботі Олена-Іванна показала на практичних та теоретичних прикладах, як гаджет бібліотеки можуть бути оптимізовані для рекурсивних zk-SNARKs, що має реальне практичне застосування. Студентка також виконала великий об’єм роботи з кодом, щоб продемонструвати, що припущення наведені в дипломній роботі, справді виконуються, і множення точок еліптичної кривої на скаляр оптимізоване.

Дипломна робота демонструє високий рівень володіння предметом та здатність авторки висвітлити складні теми та принципи в зрозумілий та доступний спосіб. Вважаю, що дипломна робота виконана на високому рівні, відповідає вимогам до дипломних робіт і заслуговує на оцінку «відмінно», а її авторка заслуговує на присвоєння кваліфікації бакалавра.

Асистент кафедри обчислювальної математики
Факультету комп’ютерних наук та кібернетики
Київського національного університету
імені Тараса Шевченка  Сергій ДЕНИСОВ

Рецензія

на кваліфікаційну роботу бакалавра на тему:
«Оптимізація гаджет бібліотек для рекурсивних zk-Snarks»
студентки 4-го курсу факультету комп'ютерних наук та кібернетики Київського національного університету імені Тараса Шевченка
Ощипок Олени-Іванни Василівни

Студентка Ощипок Олена-Іванна у своїй роботі «Оптимізація гаджет бібліотек для рекурсивних zk-Snarks», проводить глибокий та детальний аналіз використання zero-knowleadge proof і zk-Snarks в контексті еліптичних кривих, зокрема їх оптимізацію для множення точок еліптичних кривих на скаляр.

Студентка демонструє глибоке розуміння теми, надаючи чітке та структуроване викладання важливих понять. Зокрема, стаття досліджує алгоритми множення точок еліптичної кривої на скаляр з використанням ендоморфізму, подвійне скалярне множення з використанням трюку Штрауса-Шаміра з урахуванням skew representation, і зображення точок еліптичної кривої в проєктивному зображені, щоб можна було обробляти точки нескінченності та зробити множення більш безпечним в контексті zero-knowledge proofs. Автор також використовує генератори точок порядку два та три в роботі – вдалий підхід до уникнення виключень в роботі системи.

Однак, є деякі аспекти, які могли бути більш детально висвітлені. Було б варто розглянути додаткові аспекти оптимізації в контексті більш складних сценаріїв використання, а також висвітлити результати використання наведеної арифметики на афінних точках. При цьому, в цілому дипломна робота написана на високому рівні і відповідає сучасному дискурсу з обраної теми.

Вважаю, що кваліфікаційна робота Ощипок Олени-Іванни Василівни відповідає всім вимогам, які висуваються до бакалаврських робіт, і заслуговує на оцінку «відмінно», а її автор заслуговує на присвоєння кваліфікації бакалавра.

Рецензент:
кандидат технічних наук,
доцентка — Катерина ГОЛУБЄВА