Т3. №5(37)'2025

Інформаційні технології

УДК 004.42+004.056.55

КЛАСИФІКАЦІЯ СИСТЕМ ДОВЕДЕННЯ НЕІНТЕРАКТИВНИХ АРГУМЕНТІВ ЗНАННЯ

Ю.М. Паславський[1], І.М. Крошний[2]
[1] *Department of Computer Science, National Forestry University of Ukraine, Lviv, Ukraine*
[2] *Department of Computer Science, National Forestry University of Ukraine, Lviv, Ukraine*

E-mail: mykhailo.paslavskyi@nltu.edu.ua

АНОТАЦІЯ

Важливим криптографічним механізмом, який гарантує конфіденційність (властивість нульового розголошення) та забезпечує неможливість довести перевіряючому хибне твердження, є докази з нульовим розголошенням. Популярною реалізацією доказів із нульовим розголошенням є короткі, неінтерактивні докази, які можна швидко перевірити і які не потребують взаємодії між сторонами після початкового налаштування. Основним напрямом у розробці сучасних систем доведення є інтерактивне доведення, яке будується за два кроки. Перший – надсилання підтвердження поліному інтерактивного оракульного доказу та другий – створення правильних оракулів схеми поліноміальних зобов'язань за допомогою чітко визначених криптографічних методів оцінки поліномів. Перевірка використання однакових коефіцієнтів у кожній лінійній комбінації потребує перевірки як поліноміальної узгодженості, так і узгодженості змінних. Для побудови загальних схем стислого неінтерактивного аргументу знання з нульовим розголошенням було запропоновано поліном інтерактивного оракульного доказу, що моделює повідомлення як поліноміальні оракули. Усі тести доводяться за допомогою схем поліноміальних зобов'язань, а потім із нульовим знанням оцінюються в точці, заданій особою, що перевіряє інформацію. Надійність і конфіденційність усіх тестів базується на трьох основних категоріях поліномів інтерактивного оракульного доказу, а саме схеми поліноміальних зобов'язань зі сполученням, з аргументом внутрішнього добутку та з теорією коду. Протоколи стислих неінтерактивних аргументів знання з нульовим розголошенням реалізуються через програми високого рівня (компілятори), які перетворюються на проміжне представлення, тобто схему, що визначається системою обмежень. Використовувані компілятори поділяються на доменно-орієнтовані мови, вбудовані доменно-орієнтовані мови та віртуальні машини з нульовим розкриттям знань. Спеціалізовані доменно-орієнтовані мови опису апаратного забезпечення або мови програмування пропонують адаптований синтаксис для ефективного вираження обмежень в арифметичних схемах. Вбудовані доменно-орієнтовані мови реалізуються як функції в мовах програмування загального призначення й орієнтовані на провідні схеми, успадковані від вбудованої мови. Віртуальні машини з нульовим розкриттям знань обробляють операційний код циклу «вибірка – декодування – виконання», реплікуючи трасування обчислень для загальних програм і генеруючи відповідні підтвердження з нульовим розголошенням. Вони сумісні з існуючими мовами програмування високого рівня та можуть використовувати функції існуючих компіляторів. Компілятори оцінюються за перехресною або синтаксичною сумісністю. Загалом найбільшою перешкодою у використанні бібліотек неінтерактивних доказів є брак документації. Стандартизація може допомогти розробникам порівняти важливі функції різних бібліотек, а також встановити більш узгоджену базову лінію продуктивності. Документація бібліотеки щодо цих основних функцій є неявною, і розробникам потрібно розуміти основні криптографічні методи, щоб вибрати відповідну схему. Важливою є стандартизація параметрів компіляторів, що ускладнює повторне використання існуючих інструментів.

Ключові слова: *докази з нульовим розголошенням, інтерактивне доведення, поліном інтерактивного оракульного доказу, схеми поліноміальних зобов'язань, компілятори, доменно-орієнтовані мови, віртуальні машини, стандартизація.*

Вступ

Важливим криптографічним механізмом, який дає змогу одній особі довести іншій стороні істинність певного твердження, не розкриваючи при цьому жодної додаткової інформації про сам доказ або секрети, на яких він базується, є докази з нульовим розголошенням. Ця криптографічна конструкція поєднує гарантію конфіденційності (властивість нульового розголошення), тобто особа, яка перевіряє інформацію, не отримує нічого, що могло б розкрити секрети особи, що доводить інформацію. Другою критичною характеристикою є обґрунтованість, яка забезпечує, що жоден шахрай не зможе довести перевіряючому хибного твердження. Серед особливо популярних реалізацій доказів із нульовим розголошенням виділяються короткі, неінтерактивні докази, які можна швидко перевірити і які не потребують взаємодії між сторонами після початкового налаштування. Такий підхід дає змогу ефективно використовувати ці алгоритми в багатьох практичних застосуваннях – від конфіденційних обчислень до масштабованих блокчейн-систем. Ключовою властивістю безпеки неінтерактивних доказів є обґрунтованість знань, яка означає, що будь-хто, хто здатен згенерувати правильний доказ, насправді володіє певним «свідком», тобто секретною інформацією, що підтверджує істинність твердження. Свідок – це не просто довільний набір даних, а така інформація, яка дає змогу ефективно (наприклад, поліноміально) перевірити, що деяке твердження належить до певної мови класу недетермінованого поліноміального часу. Водночас ця модель традиційно розглядає лише одиничні дії доказувача, що залишає простір для розширення в напрямі багатосторонніх або паралельних сценаріїв.

Лінійні системи ймовірнісно перевіреного доведення й інтерактивне (оракульне) доведення є ядром стислих неінтерактивних аргументів знання з нульовим розголошенням на основі інформаційно-теоретичного доведення. Основні властивості цих аргументів – прозорість, постквантова безпека, універсальне налаштування й ефективність.

Аналіз літературних даних і постановка проблеми

Перший ефективний стислий неінтерактивний аргумент знання з нульовим розголошенням (zk-SNARK, Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) для загальних схем запропонували Дженнаро та ін. [13] у 2013 році. Вони використали техніку квадратичної програми проміжків (QSP, quadratic span program), яка є простішою формою квадратичної арифметичної програми QAP (Quadratic Arithmetic Program). Основна ідея цих алгоритмів полягає в побудові набору поліноміальних рівнянь і використанні пар для перевірки цих рівнянь.

У 2016 році в одне рівняння було інтегровано перевірку валідності, поліноміальну та змінну перевірки узгодженості з використанням для цього лише трьох пар. [7] Розмір доказу було додатково зменшено до оптимізованих трьох елементів. Після цього теоретичні досягнення та практична робота зосереджувалися на розробці компілятора для QAP [5]. Подальші роботи додатково аналізують властивості безпеки [6] та застосування його до конкретних програм разом з різними моделями, як-от багатостороння установка [15], універсальний довідковий рядок (URS) [7] та рекурсивне доведення [5].

Розмір доказу в цих системах залишається постійним, а час, витрачений на доказування, є лінійним. Ці атрибути є особливо вигідними та сприяють реальним впровадженням. Проте суттєвим обмеженням систем на основі QAP є значні накладні витрати часу роботи доказів і споживання пам'яті, що створює проблеми для масштабування до великих операторів. Крім того, кожен оператор вимагає окремої довіреної конфігурації [10].

Мета та задачі дослідження

Метою статті є дослідження теоретичних і прикладних аспектів реалізації протоколів стислих неінтерактивних аргументів знання з нульовим розголошенням на основі інтерактивних доведень, зокрема аналіз варіантів поліному інтерактивного оракульного доказу й оракулів схеми поліноміальних зобов'язань, їх обчислювальної складності, безпекових характеристик і програмної реалізації в різних компіляторах.

Для досягнення мети дослідження потрібно виконати такі **завдання**:

1. Провести класифікацію існуючих протоколів доведення знання без розкриття додаткової інформації з виокремленням їх теоретичних основ.

2. Проаналізувати криптографічні механізми схем поліноміальних зобов'язань, включно з варіантами на основі парних обчислень, аргументів внутрішнього добутку та лінійних кодів.

3. Оцінити ефективність одновимірних і багатоваріантних поліномів інтерактивного оракульного доказу з погляду їх продуктивності, безпеки та постквантової стійкості.

4. Визначити реалізаційні аспекти неінтерактивних протоколів у сучасних компіляторах, зокрема, з огляду на сумісність, ефективність та обмеження їх застосування.

5. Ідентифікувати основні проблеми реалізації та використання неінтерактивних протоколів, включно з браком документації, складністю компіляції та потребою в стандартизації.

Copyright © 2025, ОТА.

Матеріали та методи досліджень

Для досягнення мети дослідження було застосовано методи структурного та порівняльного аналізу, що дали змогу всебічно оцінити наявні технології та рішення для створення системи аналізу стислих неінтерактивних аргументів знання з нульовим розголошенням. Структурний аналіз сприяв визначенню основних компонентів системи, їх функціональних і нефункціональних вимог, а також взаємодії між ними. Визначено, що протоколи неінтерактивних аргументів знання реалізуються через програми високого рівня (компілятори), які перетворюються на проміжне представлення, тобто схему, що визначається системою обмежень. Порівняльний аналіз дав змогу оцінити переваги та недоліки наявних рішень, як-от синтаксис, інтерфейси, користувальницький досвід, простота доменно-орієнтованих мов і віртуальних машин із нульовим розкриттям знань. Одновимірні та багатоваріантні поліноми інтерактивного оракульного доказу варто оцінювати за розміром доказу, безпекою, постквантовою захищеністю та параметрами довіри.

Для збору та систематизації інформації було застосовано методи інформаційного й аналітичного підходу. Це дало змогу здійснити глибоке вивчення наукової літератури, сучасних систем доведення, визначення прийнятності налаштувань довіри, відповідну масштабованість і необхідність постквантової безпеки, а також доступних онлайн-ресурсів, що стосуються стандартизації інтелектуальних систем для встановлення узгодженої базової лінії продуктивності. Вивчення інтерактивного доведення, квадратичної арифметичної програми й поліному інтерактивного оракульного доказу забезпечило формулювання основних вимог до функціональності та зручності користування системами аналізу стислих неінтерактивних аргументів знання з нульовим розголошенням.

Результати досліджень

Основним напрямом у розробці сучасних систем доведення є інтерактивне доведення (IP, Interactive proof) – це узагальнення (лінійного) ймовірнісно перевіреного доведення (PCP, (Linear) Probabilistically Checkable Proof), у якому особа, що перевіряє інформацію, може надсилати випадкові повідомлення особі, що доводить інформацію протягом кількох раундів. Побудова IP поділяється на два кроки: (1) надсилання підтвердження поліному інтерактивного оракульного доказу PIOP (Polynomial Interactive Oracle Proof), який моделює повідомлення (оракул), надіслане особою, що доводить інформацію; та (2) створення правильних оракулів схеми поліноміальних зобов’язань PCS (Polynomial Commitment Scheme) за допомогою чітко визначених криптографічних методів оцінки поліномів, надісланих особою, що доводить інформацію.

zk-SNARK на основі PCP

Імовірнісно перевірений доказ PCP ((Linear) Probabilistically Checkable Proof) перевіряє лише випадково вибрану крихітну частину доказу з надзвичайно високою імовірністю. Спочатку PCP мали високу асимптотичну складність і не зосереджувалися на загальних обчислювальних моделях. У першому ефективному zk-SNARK (Zero-Knowledge Succinct Non-Interactive Argument of Knowledge) для загальних схем використовувалися техніки квадратичної програми проміжків [13]. Перевірка використання однакових коефіцієнтів у кожній лінійній комбінації потребує перевірки як поліноміальної узгодженості, так і узгодженості змінних:

− **Перевірка поліноміальної узгодженості:** особа, що доводить інформацію, обчислює $g^{L(X)}\, ma\, g^{\alpha L(x)}$, а особа, що перевіряє інформацію, верифікує, чи рівність $e\left(g^{\alpha L(x)}, g\right) = e\left(g^{L(x)}, g^{\alpha}\right)$ виконується. Для всіх поліномів особа, що доводить інформацію, також обчислює елементи групи для $R(x), O(x), t(x)$ і проводить аналогічну перевірку над ними.

− **Перевірка узгодженості змінних:** на основі згенерованих довірчою установкою заданих випадкових значення $\beta_l, \beta_r, \beta_o$ особа, що перевіряє інформацію, обчислює значення $\prod_i^m \left(g^{\beta_l l_i(x) + \beta_r r_i(x) + \beta_o o_i(x)}\right)^{c_i}$ як частину доказу, позначеного як $g^{Z(x)}$. Також ця особа перевіряє рівність $e\left(g^{Z(x)}, g^{\gamma}\right) = e\left(g^{L(x)}, g^{\beta_l \gamma}\right) \cdot e\left(g^{R(x)}, g^{\beta_r \gamma}\right) \cdot e\left(g^{O(x)}, g^{\beta_o \gamma}\right)$.

IP на основі GKR для багаторівневих схем

Ранні IP-протоколи були розроблені переважно для багаторівневих схем, де кожен вентиль підключався лише до вищого рівня. Для доведення задовільності такої схеми шляхом редукції від рівня до рівня розроблений протокол Голдвассерарage18 – Калаї – Ротблюма (GKR, Goldwasser–Kalai–Rothblum) [16], який перевіряє правильність обчислення з виходу останнього шару. Особа, що доводить інформацію, в протоколі GKR доводить для кожного попередження *c* на кожному рівні *i*, виконується таке рівняння:

$$V_{i+1}(c) = \sum_{a,\, b \in \{0,1\}^{s_i}} \Big(ADD_i(a,b,c) \cdot \big(V_i(a) + V_i(b)\big) + MUL_i(a,b,c) \cdot V_i(a) V_i(b) \Big),$$

де $ADD_i(a,b,c)$ та $MUL_i(a,b,c)$ – предикти доказів кубічної складності (повертають 1, коли *a,b,c* поєднують вентиль додавання або множення відповідно).

Подальші дослідження [8] розширили функції *V, ADD, MUL* до поліномів і використовують поліноміальні обчислення для оптимізації складності до

квазілінійної. Підходи на основі GKR є подвійно ефективними, що означає, що вони мають квазілінійне доведення разом з ефективною перевіркою, де час верифікації лінійний до входу багатошарової схеми. Незважаючи на вдосконалення протоколу GKR, суттєвим обмеженням є те, що він працює лише на багатошарових арифметичних схемах. Це створює значні накладні витрати під час доповнення загальних схем до багатошарових за допомогою фіктивних вентилів.

PIOP для загальних схем

Для побудови загальних схем zk-SNARK було запропоновано узагальнену форму IP, яка називається PIOP, що моделює повідомлення, надіслане особою, що доводить інформацію, як поліноміальні оракули, які повертають поліноміальні обчислення. Щоб отримати IP, оракули в PIOP повинні бути ініційовані за допомогою PCS, яка обчислює поліном у певній точці з урахуванням надійності та конфіденційності.

Одновимірний PIOP. Ідея одновимірного PIOP полягає в моделюванні обчислень у загальній схемі як полінома, а потім – у доведенні його властивостей. Особа, що доводить інформацію, використовує поліном *T* для кодування значень у всій обчислювальній області, як-от входи та значення проводів, а також поліном логічного елемента *S* для кодування всіх логічних елементів додавання та множення, наприклад, *S(a) = 0*, якщо *a* є логічним елементом додавання, а *S(a) = 1* представляє логічний елемент множення.

Особа доводить задовільність схеми *y* за допомогою такого рівняння для будь-якого *y*:

$$S(y)\left[T(y)+T(\omega y)\right]+\left(1-S(y)\right)T(y)T(\omega y)=T\left(\omega^2 y\right),$$

де *ω* – зміщення логічного елемента, $T(y), T(\omega y), T\left(\omega^2 y\right)$ – лівий і правий входи та вихід логічного елемента *y* відповідно.

Існують різні інші поліноміальні співвідношення, пов’язані з *T* і *S*, щоб забезпечити коректність схем, зокрема нульовий тест, перевірка на добуток і на перестановку. Усі тести доводяться за допомогою PCS, коли особа, що доводить інформацію, спочатку надсилає підтвердження цих поліномів, а потім із нульовим знанням оцінює їх у точці, заданій особою, що перевіряє інформацію. Надійність і конфіденційність усіх тестів базується на трьох основних категоріях PCS.

PIOP зі сполученням. Поліноміальне зобов’язання Кейт, Заверухи та Голдберга (KZG) [2] має докази обчислення, які складаються лише з одного білінійного групового елемента, а перевірка обчислення потребує лише одного парного обчислення. Для обчислення $f(u)=\nu$ в точці *u*, особа, що доводить інформацію, будує функцію $f(x)-\nu=(x-u)t(x)$ для деякого полінома $t(x)$ і обчислює доказ як $\pi=g^{t(s)}$, де *s* – це секретне значення, обчислене під час довірчого налаштування. Перевірка виконується за допомогою операції сполучення $e\left(com/g^{\nu},g\right)=e\left(g^{s}/g^{u},\pi\right)$ (*com* – це зобов’язання для полінома, згенерованого під час налаштування). Однак ця асимптотично оптимальна продуктивність досягається ціною довірчого налаштування, яка виводить g^s, і *s* має бути видалений після генерації.

PIOP з аргументом внутрішнього добутку (IPA, inner-product argument). Щоб усунути налаштування довіри в PCS на основі пар, створюється куленепробивний (BulletProof) [3] екземпляр PIOP через нову PCS, використовуючи алгебраїчні трюки на основі IPA. Доводячи поліном *f* зі ступенем *m, що* дорівнює *v* в точці *u* (тобто

$$f(u)=\sum_{i=0}^{m}c_i u^i=\nu,$$

де c_i – коефіцієнт), особа, що доводить інформацію, згортає поліном у дві частини як $f(u)=f_L(u)+u^{m/2}f_R(u)$. Спочатку довівши правильність згортання, а потім рекурсивно викликаючи процедуру, особа, що доводить інформацію, може отримати логарифмічний доказ і лінійний час доведення та перевірки, пов’язаний зі степенем полінома.

PIOP з теорією коду. Щоб досягти як прозорого налаштування, так і постквантової безпеки, варто використовувати лінійний код для побудови PCS [14]. Код $[n,k,\Delta]$ має три властивості: (1) може закодувати довільне повідомлення в кодове слово; (2) мінімальна відстань (Хеммінга) між будь-якими двома кодовими словами дорівнює Δ; та (3) будь-яка лінійна комбінація кодових слів також є кодовим словом. Використання коду Ріда – Соломона [17] розглядає повідомлення як поліном ступеня *k–1* та кодове слово як його обчислення в *n* фіксованих точках. У PCS коефіцієнти полінома *m+1* спочатку кодуються в $O\left(\sqrt{m}\right)$ кодових слів. Потім особа, що доводить інформацію, фіксує кодові слова, використовуючи дерево Меркла, щоб забезпечити перевірку існування певних кодових слів. Щоб перевірити обчислення $f(u)=\nu$, особа, що перевіряє інформацію, надсилає повідомлення $\left(1,u,\ldots,u^{O(\sqrt{m})}\right)$ запитуючи в особи, що доводить інформацію, виконання лінійних комбінацій кодових слів, використовуючи повідомлення як коефіцієнти. Особа, що доводить інформацію, перевіряє (1) згенерований результат із використанням зафіксованого раніше кодового слова (використовуючи дерево Меркла), а також (2) чи результат є кодовим словом того ж класу, що й закодовані кодові слова. Оскільки повідомлення має довжину $O\left(\sqrt{m}\right)$, розмір доведення та час верифікації мають складність $O\left(\sqrt{m}\right)$. Вузьким місцем для особи, що доводить інформацію, є кодування полінома, яке потребує швидкого перетворення Фур’є (FFT, Fast Fourier Transform), складність якого становить $O\left(\sqrt{m}\right)$.

Copyright © 2025, ОТА.

У пізніших роботах узагальнюється ідея поліноміального кодування шляхом поділу коефіцієнтів у поліномі на багатовимірності та кодування їх у більше кодових слів [9] для досягнення компромісу між часом і простором. Fractal та інші наступні роботи [1] використовують новий варіант під назвою швидкий інтерактивний оракульний доказ близькості за методом Ріда – Соломона (FRI, Fast Reed–Solomon IOP of Proximity) [5]. FRI розглядає поліноміальні коефіцієнти як вектор розміру $O\left(\sqrt{m}\right)$ та рекурсивно кодує його, складаючи навпіл щоразу для досягнення логарифмічного розміру доказу. Застосовуючи всі вищезгадані передові методи теорії коду, існуючі PCS-коди можуть досягти логарифмічного розміру верифікатора та доказу, а також лінійного доказу та постквантової безпеки.

Багатоваріантний PIOP. Хоча ефективні PCS можуть зменшити розмір доказу й навантаження на особу, що перевіряє інформацію, використання FFT для побудови ключового полінома в одновимірному PIOP було вузьким місцем на стороні особи, що доводить інформацію, оскільки воно вводить квазілінійну складність. Щоб вирішити цю проблему ефективності, було проведено дослідження, спрямовані на багатоваріантне поліноміальне обчислення для усунення швидкого перетворення Фур'є [4]. Ці роботи потребують модифікації протоколу PIOP та PCS до багатоваріантного типу, а потім використання протоколу *Sumcheck* для доведення, що для деякої функції $f:\{0,1\}^n \to F$ виконується:

$$\sum_{x\in\{0,1\}^n} f(x) = S,$$

тобто сума значень *f(x)* на всіх 2^n булевих входах дорівнює заданому *S*.

Далі ми фіксуємо першу змінну x_1 та визначаємо унарну функцію:

$$g_1(z) = \sum_{x_2,\ldots,x_n\in\{0,1\}} f(z,x_2,\ldots,x_n).$$

На кожному рекурсивному раунді особа, що доводить інформацію (Prover), надсилає одновимірний поліном $g_i(z)$, який є сумою залишених змінних.

Особа, що перевіряє інформацію (Verifier), верифікує функцію

$$g_i(0) + g_i(1) = g_{i-1}(r_{i-1})$$

і вибирає випадкову точку $r_i \in F$ і продовжує.

На фінальному кроці особа, що перевіряє інформацію, запитує значення функції $f(r_1,r_2,\ldots,r_n)$ і перевіряє, що воно узгоджене з останнім поліном.

Реалізації протоколів zk-SNARK

Протоколи zk-SNARK реалізуються через програми високого рівня (компілятори), які перетворюються на проміжне представлення, тобто схему, що визначається системою обмежень. Потім схема передається до системи доведення, яка реалізує специфічні методи zk-SNARK для виведення доказу.

Зазвичай використовувані компілятори для zk поділяються на доменно-орієнтовані мови (DSL, Domain-Specific Languages), вбудовані доменно-орієнтовані мови (eDSL, Embedded Domain-Specific Languages) та віртуальні машини з нульовим розкриттям знань (zk-VM, Zero-Knowledge Virtual Machines). Вхід DSL – це незалежний файл із синтаксисом, пов'язаним з обмеженнями схеми, окремий від бібліотечних функцій, а його вихідні дані – це окремий файл, що містить інформацію про схему. Вхід eDSL поєднує бібліотечні функції, пов'язані із системою обмежень, часто з використанням гаджетів (вбудовані функції для складних обмежень, як-от внутрішні добутки або специфікації циклів), які допомагають створювати вхідні дані компілятора, а виходом eDSL є структура даних для системи доказу. Вхід zk-VM – операційні коди, скомпільовані компіляторами загального призначення, а його виходом є інформація про схему.

Доменно-орієнтовані мови (DSL). DSL – це спеціалізовані мови програмування, розроблені для конкретних проблемних областей, що пропонують адаптований синтаксис для ефективного вираження обмежень в арифметичних схемах для zk-SNARK. Сучасні DSL класифікуються як мови опису апаратного забезпечення (HDL, hardware description languages) [11] або мови програмування (PL, programming languages) [12].

Мови високого рівня (HDL) описують синтез схем безпосередньо у провідній формі, забезпечуючи елегантний синтаксис, але створюючи труднощі для програмістів через їхню незалежну провідну структуру й обмежену абстракцію типів даних, оскільки вхідні дані представлені як структури сигнальних даних. На противагу цьому мови програмування (PL) визначають обмеження мовами програмування високого рівня, підтримуючи різні типи даних і нагадуючи такі мови, як Rust або Python. Це робить PL більш доступними для програмістів без знань провідних схем, пропонуючи найпростіший спосіб визначення обмежень. Однак гнучкий синтаксис PL збільшує ризики вразливості та створює проблеми з ефективністю.

Вбудовані доменно-орієнтовані мови (eDSL). eDSL для zk-SNARK набули популярності в останні роки та реалізуються як функції в мовах програмування загального призначення, що відрізняє їх від традиційних компіляторів у контексті мов програмування. eDSL призначені для опису синтезу схем, подібно до HDL, але вони орієнтовані на провідні схеми (структуроване представлення арифметичних схем, де схема описується як множина проводів (wires) і гейтів (gates), що з'єднують ці проводи),

пропонуючи більшу виразність та простоту використання, успадковуючи структури даних і функції програмування від вбудованої мови. Ці eDSL оптимізують розробку ZK-доказів, інтегруючи визначення схеми та генерацію доказів в один файл, спрощуючи код і даючи можливість програмістам використовувати існуючі функції бібліотеки. Однак написання коду в eDSL вимагає від розробників чіткого розрізнення внутрішньосхемних і зовнішніх операцій, що потребує експертних знань конкретної мови та дизайну бібліотеки.

Віртуальні машини з нульовим розкриттям знань (zk-VMs) обробляють операційний код циклу «вибірка – декодування – виконання», реплікуючи трасування обчислень для загальних програм (зазвичай смарт-контрактів) та генеруючи відповідні ZK-підтвердження. Вони підтримують різні архітектури наборів інструкцій (ISA, instruction set architectures). zk-VM сумісні з існуючими мовами програмування високого рівня та можуть використовувати функції існуючих компіляторів. Однак, незважаючи на орієнтацію на низькорівневі операційні коди, zk-VM не повністю сумісні з програмами верхнього рівня та часто потребують незначних або значних модифікацій програми, що може ускладнити керування програмістами через схильність до помилок. Крім того, zk-VM використовують обчислювальну модель машини Тюрінга замість схем, що значно збільшує накладні витрати. Хоча zk-VM зменшують навантаження на програмістів, пов’язане з обмеженнями написання, вони можуть мати проблеми з ефективністю, особливо для великомасштабних програм.

Оцінку сумісності компіляторів варто проводити за двома властивостями:

– **перехресна сумісність** – вказує на те, чи може результат компіляції одного компілятора бути використаний іншим. Компілятори DSL пропонують помірну перехресну сумісність, оскільки вони розділяють систему обмежень і систему доведення. Зі стандартизацією результатів компіляції в майбутньому бібліотеки зможуть зосередитися лише на наданні систем доведення, приймаючи результати DSL як вхідні дані. Компілятори eDSL мають низьку перехресну сумісність, оскільки вони визначають схему в мові програмування, що ускладнює використання їхньої визначеної схеми на платформах з іншими мовами. Навіть однією мовою результат компіляції може бути несумісним, оскільки функції гаджетів відрізняються. Віртуальні машини zk мають низьку перехресну сумісність, оскільки вони розроблені лише для деяких специфічних програм високого рівня;

– **синтаксична сумісність** – вказує на те, чи має мова введення компілятора подібний синтаксис до іншої мови. Сумісність синтаксису важлива, оскільки дає змогу програмісту, знайомому з мовою, перейти до іншої без ретельного вивчення. На жаль, ми виявляємо, що навіть в одній категорії мови компілятора мають зовсім інший синтаксис, тож буде важко вивчити їх усі. У DSL HDL є мовою апаратних схем, тоді як PL більше схожа на загальну мову програмування. У eDSL синтаксис залежить від базової мови бібліотеки, починаючи від C, C++, Rust, Go та JavaScript. У zk-VM добре підтримуються лише операції з мов смарт-контрактів, а операції з інших загальних мов не пройдуть компіляцію.

Обговорення результатів

Основною проблемою zk-SNARK є компроміс між ефективністю та безпекою – лінійний PCP досягає постійного розміру доказу, але завдяки налаштуванню довіри.

Керівними принципами вибору відповідної системи доведення є (1) визначення прийнятності налаштувань довіри; (2) відповідна масштабованість; (3) необхідність постквантової безпеки.

Загалом найбільшою перешкодою у використанні бібліотек zk-SNARK є брак документації. Тому варто створювати й уніфікувати динамічні документи користувача, деталі API гаджетів в eDSL та синтаксис мови в DSL.

Стандартизація може допомогти розробникам порівняти важливі функції в полі zk-SNARK різних бібліотек, а також встановити більш узгоджену базову лінію продуктивності. Документація бібліотеки щодо цих основних функцій є неявною, і розробникам потрібно розуміти основні криптографічні методи, щоб вибрати відповідну схему. Важливою є стандартизація параметрів компіляторів, що ускладнює повторне використання існуючих інструментів.

Залишається відкритим питання, чи системи доказування для конкретних сценаріїв працюватимуть краще, ніж загальні системи доказування. Розробка таких специфічних систем доказування потребує співпраці між теоретичними розробками та інженерією.

Реалізації схем zk-SNARK обмежені в різних мовах програмування та не містять жодних інтерфейсів для сумісності. Хоча такі компоненти, як системи обмежень і системи доведення, можна розділити в коді, їхні функції та інструменти обмежені відповідними бібліотеками. Однак відповідний вибір потребує експертних знань систем обмежень, що непрактично для програмістів.

Багато завдань вимагають глибокого знайомства як з мовою програмування, так і з системою обмежень. У багатьох бібліотеках відсутній компілятор для перетворення високорівневого коду на схемні представлення, що змушує програмістів вручну додавати обмеження. На схемному рівні програмісти повинні

Copyright © 2025, ОТА.

обробляти складні деталі, як-от операції з кривими, цикли та перестановки. Вибір невідповідної кривої також може знизити ефективність чи створити вразливості.

Висновки

Отже, відсутність універсальної стандартизації є проблемою, пов'язаною із сумісністю. В одній категорії існують значні відмінності в синтаксисі, що ускладнює міграцію проєктів і заплутує програмістів, а також навіть для тієї самої схеми результат компіляції не може бути використаний системою перевірки в іншій бібліотеці. Основним напрямом у розробці сучасних систем доведення є IP, який може усунути необхідність налаштування довіри, довгого спільного рядка посилання (CRS) та повільного процесу доведення в zk-SNARK на основі QAP.

Варто розробити систему заходів для вирішення або пом'якшення проблем із мовою та сумісністю через створення образів. Проблему неправильного використання схем і кривих можна вирішити розробкою вичерпних рекомендацій. Проблеми, пов'язані з компіляторами, вирішуються класифікацією існуючих компіляторів у кожній бібліотеці та ґрунтовним аналізом їхніх сильних і слабких сторін. Доречно задокументувати розроблені матеріали з відкритим кодом (API, Application Programming Interface), що забезпечують прозорість, взаємодію та інновації, а саме документацію користувача (встановлення, використання та тестування) і зразки коду для програм.

Конфлікт інтересів

Автори декларують, що не мають конфлікту інтересів стосовно цього дослідження, у тому числі фінансового, особистісного характеру, авторства чи іншого характеру, що міг би вплинути на дослідження та його результати, представлені в цій статті.

Фінансування

Дослідження проводилося без фінансової підтримки.

Доступність даних

Рукопис не має пов'язаних даних.

ЛІТЕРАТУРА

[1] Alan Szepieniec and Yuncong Zhang. Polynomial iops for linear algebra relations. In International Conference on Public-Key Cryptography (PKC), pages 523–552. Springer, 2022. DOI: 10.1007/978-3-030-97121-2_19.

[2] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Advances in Cryptology – ASIACRYPT 2010, pages 177–194, Berlin, Heidelberg, 2010. DOI: 10.1007/978-3-642-17373-8_11.

[3] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidentia ltransaction sand more. In 2018 IEEE symposium on security and privacy (SP), pages 315–334, 2018. DOI: 10.1109/SP.2018.00020.

[4] Benoit Libert. Simulation-extractable kzg polynomial commitments and applications to hyperplonk. In International Conference on Public-Key Cryptography (PKC), pages 68–98. Springer, 2024. DOI: 10.1007/978-3-031-57722-2_3.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In 45th international colloquium on automata, languages, and programming (icalp 2018), 2018. DOI: 10.4230/LIPIcs.ICALP.2018.14.

[6] Helger Lipmaa. A unifie dframewor kfo rnon-universal snarks. In International Conference on Public-Key Cryptography (PKC), pages 553–583. Springer, 2022. DOI: 10.1007/978-3-030-97121-2_20.

[7] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-snarks. In Annual International Cryptology Conference, pages 698–728, 2018. DOI: 10.1007/978-3-319-96878-0_24.

[8] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficien tinteractiv eproof sfo rgenera larithmetic circuits with linear prover time. In ACM SIGSAC Conference on Computer and Communications Security, pages 159–177, 2021. DOI: 10.1145/3460120.3484767.

[9] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Theory of Cryptography (TCC), pages 19–46. Springer, 2020. DOI: 10.1007/978-3-030-64378-2_2.

[10] Liang J., Hu D., Wu P., Yang Y., Shen Q., & Wu Z. (2025). SoK: Understanding zk-SNARKs: The gap between research and practice. *arXiv*. DOI: 10.48550/arXiv.2502.02387.

[11] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. IEEE Transactions on Dependable and Secure Computing, vol. 20 (6), pp. 4733–4751, 2022.

[12] Nada Amin, John Burnham, François Garillot, Rosario Gennaro, Daniel Rogozin, Cameron Wong, et al. Lurk: Lambda, the ultimate recursive knowledge. Cryptology ePrint Archive, 2023.

[13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 626–645, 2013. DOI: 10.1007/978-3-642-38348-9_29.

[14] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero:

Lightweight sublinear arguments without a trusted setup. In ACM SIGSAC Conference on Computer and Communications Security, pages 2087–2104, 2017. DOI: 10.1145/3133956.3134104.

[15] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive, 2017.

[16] Shafi Goldwasser, Yael Tauman Kalai, and Guy Rothblum. Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM), vol. 62 (4), pp. 1–64, 2008. DOI: 10.1145/1391289.1391296.

[17] Stephen Wicker and Vijay Bhargava. Reed-Solomon codes and their applications. John Wiley & Sons, 1999.

CLASSIFICATION OF NON-INTERACTIVE KNOWLEDGE ARGUMENT PROOF SYSTEMS

Yurii Paslavskyi, Ihor Kroshnyi

An important cryptographic mechanism that guarantees confidentiality (the zero-disclosure property) and ensures that it is impossible to prove a false statement to the verifier is zero-disclosure proofs. A popular implementation of zero-disclosure proofs is short, non-interactive proofs that can be quickly verified and that do not require interaction between the parties after the initial setup. The main direction in the development of modern proof systems is interactive proof, which is built in two steps. The first is sending a confirmation of the polynomial of an interactive oracle proof and the second is creating correct oracles of the polynomial commitment scheme using well-defined cryptographic methods for evaluating polynomials. Verifying the use of the same coefficients in each linear combination requires checking both polynomial consistency and variable consistency. To construct general schemes of concise non-interactive zero-disclosure knowledge argument, an interactive oracle proof polynomial was proposed that models messages as polynomial oracles. All tests are proved using polynomial commitment schemes and then evaluated with zero knowledge at a point specified by the person verifying the information. The reliability and confidentiality of all tests are based on three main categories of interactive oracle proof polynomials, namely polynomial commitment schemes with conjunction, with inner product argument and with code theory. The protocols of concise non-interactive zero-disclosure knowledge arguments are implemented through high-level programs (compilers), which are converted into an intermediate representation, i.e. a scheme defined by a system of constraints. The compilers used are divided into domain-oriented languages, embedded domain-oriented languages, and zero-knowledge virtual machines. Specialized domain-oriented hardware description languages or programming languages offer an adapted syntax for efficiently expressing constraints in arithmetic schemes. Embedded domain-oriented languages are implemented as functions in general-purpose programming languages and are oriented to the overhead schemes inherited from the embedded language. Zero-knowledge virtual machines process the opcode of the fetch-decode-execute cycle, replicating the computation trace for general programs and generating corresponding zero-knowledge proofs. They are compatible with existing high-level programming languages and can use the features of existing compilers. Compilers are evaluated for cross- or syntactic compatibility. In general, the biggest obstacle to using non-interactive proof libraries is the lack of documentation. Standardization can help developers compare important features across libraries and establish a more consistent performance baseline. Library documentation for these core features is implicit, and developers need to understand the underlying cryptographic techniques to choose an appropriate scheme. Standardization of compiler options is important, making it difficult to reuse existing tools.

Keywords: *zero-knowledge proof, interactive proof, polynomial oracle proof, polynomial commitment schemes, compilers, domain-specific language, virtual machine, standardization.*

REFERENCES

[1] Alan Szepieniec and Yuncong Zhang. Polynomial iops for linear algebra relationsIn International Conference on Public-Key Cryptography (PKC), pages 523–552. Springer. 2022. DOI: 10.1007/978-3-030-97121-2_19.

[2] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Advances in Cryptology – ASIACRYPT 2010, pages 177–194, Berlin, Heidelberg. 2010. DOI: 10.1007/978-3-642-17373-8_11.

[3] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidentia ltransaction sand more. – In 2018 IEEE symposium on security and privacy (SP), pages 315–334, 2018. DOI: 10.1109/SP.2018.00020.

[4] Benoit Libert. Simulation-extractable kzg polynomial commitments and applications to hyperplonk. In International Conference on Public-Key Cryptography (PKC), pages 68–98. Springer, 2024. DOI: 10.1007/978-3-031-57722-2_3.

[5] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. (2018). Fast reed-solomon interactive oracle proofs of proximity. In 45th international colloquium on automata, languages, and programming (icalp 2018). DOI: 10.4230/LIPIcs.ICALP.2018.14.

[6] Helger Lipmaa. A unifie dframewor kfo rnon-universal snarks. In International Conference on Public-Key Cryptography (PKC), pages 553–583. Springer, 2022. DOI: 10.1007/978-3-030-97121-2_20.

[7] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and

Copyright © 2025, OTA.

universal common reference strings with applications to zk-snarks. In Annual International Cryptology Conference, pages 698–728, 2018. DOI: 10.1007/978-3-319-96878-0_24.

[8] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficien tinteractiv eproof sfo rgenera larithmetic circuits with linear prover time. In ACM SIGSAC Conference on Computer and Communications Security, pages 159–177, 2021. DOI: 10.1145/3460120.3484767.

[9] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. In Theory of Cryptography (TCC), pages 19–46. Springer, 2020. DOI: 10.1007/978-3-030-64378-2_2.

[10] Liang J., Hu D., Wu P., Yang Y., Shen Q., & Wu Z. SoK: Understanding zk-SNARKs: The gap between research and practice. *arXiv*. 2025. DOI: 10.48550/arXiv.2502.02387.

[11] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Albert Rubio, and Jordi Baylina. Circom: A circuit description language for building zero-knowledge applications. IEEE Transactions on Dependable and Secure Computing, vol. 20 (6), pp. 4733–4751, 2022.

[12] Nada Amin, John Burnham, François Garillot, Rosario Gennaro, Daniel Rogozin, Cameron Wong, et al. Lurk: Lambda, the ultimate recursive knowledge. Cryptology ePrint Archive. 2023.

[13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), pages 626–645. 2013. DOI: 10.1007/978-3-642-38348-9_29.

[14] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In ACM SIGSAC Conference on Computer and Communications Security, pages 2087–2104. 2017. DOI: 10.1145/3133956.3134104.

[15] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive. 2017.

[16] Shafi Goldwasser, Yael Tauman Kalai, and Guy Rothblum. Delegating computation: interactive proofs for muggles. Journal of the ACM (JACM), vol. 62 (4), pp. 1–64, 2008. DOI: 10.1145/1391289.1391296.

[17] Stephen Wicker and Vijay BhargavaReed-Solomon codes and their applications. John Wiley & Sons. 1999.

Стаття надійшла до редакції 16.09.2025
Стаття прийнята 04.10.2025
Статтю опубліковано