EOS

Efficient Outsourcing of SNARKs

Efficient Private Delegation of zkSNARK Provers

Tianyi Wang
Fujian Normal University
June 31 2025

Reference: [1] A. Chiesa et al., Eos: Efficient Private Delegation of zkSNARK Provers, *USENIX Security*, 2023.

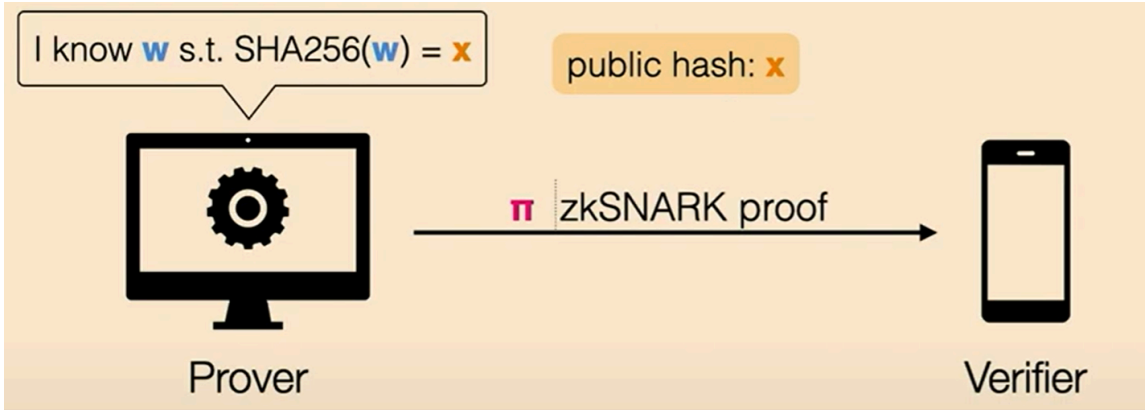
背景介绍

零知识证明（ZKP, zero-knowledge proof）：你有一个秘密，你想向别人证明你知道这个秘密，但你又不想把秘密本身透露给对方。

zkSNARKs：Z(zero)k(knowledge)S(succinct)N(non-interactive)AR(argument)K(knowledge)，零知识简洁非交互式知识论证。

应用：

- **保护隐私的加密货币**：可以隐藏交易的发送者、接收者和金额。
- **保护隐私的智能合约**：可以在不泄露合约具体内容的情况下，证明合约执行的正确性。



Reference: [1] https://www.youtube.com/watch?v=kIdMXwua4uU (USENIX Security '23 - Eos: Efficient Private Delegation of zkSNARK Provers)

为什么要有zkSNARK

传统交互式方案

如果没有 zkSNARK，使用传统 ZKP

证明者 $\mathcal{P}$ 知道 x 满足 $y = g^x \mod p$

$\mathcal{P}$ 选择随机 r，计算 $t = g^r \mod p$;

$\mathcal{P}$ 发送 t 给验证者。

验证者 $\mathcal{V}$ 发送 $\mathcal{P}$ 一个挑战 c。

$\mathcal{P}$ 计算 $s = r + c \cdot x \mod q$，发送 s。

$\mathcal{V}$ 检查：$g^s \stackrel{?}{=} t \cdot y^c \mod p$

上述内容即为 **Schnorr协议** 过程

我想要非交互式滴

将 $\mathcal{V}$ 的随机挑战 c 改为一个固定哈希

$\mathcal{P}$
选择随机 r，计算 $t = g^r \mod p$;
计算挑战：$c = H(g||y||t)$;
计算响应：$s = r + c \cdot x \mod q$;
输出证明：$\pi = (t, s)$。
$\mathcal{V}$
重新计算挑战：$c = H(g||y||t)$;
检查是否：$g^s \stackrel{?}{=} t \cdot y^c \mod p$。

新增的这一步被称为 **Fiat–Shamir 变换**

上述内容即为**非交互式 Schnorr 零知识证明（NIZK proof）** 过程

但是哈希函数怎么证明

Schnorr 协议仅用于离散对数形式，而对其他函数（多项式或哈希函数等）无能为力

这时就要搬出我们的 zkSNARK 了~

证明者 $\mathcal{P}$ 知道 x，使得 $H(x) = y$

分为四步：（具体见第二部分）

- 把 $H(x) = y$ 转换为一个 算术电路;
- 编译该电路为一组约束（如 R1CS）;
- 让证明者用 x 生成一个零知识证明 π，表明 “存在一个 x 使得约束成立 ”;
- 验证者使用公开的 y 和 π 来验证，而无需知道 x。

面临的挑战

虽然zkSNARKs 很强大，但是生成证明的过程却非常耗时和消耗计算资源。这就带来了一个难题：

- 如果用户自己生成证明，那么他们的设备可能需要**花费很长时间**，甚至无法完成。
- 如果把生成证明的任务交给云计算平台，虽然速度快了，但是用户的秘密（比如交易信息、合约内容）就会**暴露**给云计算平台。

EOS 的解决方案

EOS的核心思想：把生成证明的任务分配给多个“工人”来共同完成。

- **秘密分享**：用户把自己的秘密分成多份，发给不同的“工人”。 这样，只要有一个“工人”是诚实的，没有和其他“工人”串通，用户的秘密就不会泄露。
- **高效计算**：EOS 使用了一些特殊的技术，使得“工人”们可以高效地合作生成证明，而不需要消耗太多的计算资源。
- **安全验证**：EOS 设计了一种新的验证方法，可以确保“工人”们正确地执行了计算，防止他们作弊。

EOS 的优势

总的来说，EOS 具有以下优势：

- **保护隐私**：在生成证明的过程中，用户的秘密不会泄露给任何一个“工人”。
- **提高效率**：可以显著减少生成证明所需的时间和计算资源。
- **支持大规模计算**：使得生成复杂计算的证明成为可能，这在以前是很难实现的。

纠错

- 上次讲错的：zkSNARK证明的是一个私有的witness（etc.a pw or a sk）满足一个公开的circuit（etc.运行了一个智能合约的某段逻辑，且得到了正确结果），而非直接证明一个多项式

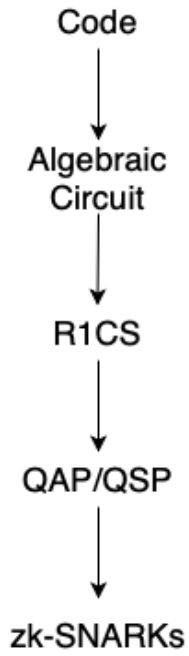
zkSNARK的关键组件

温馨提示：本章开始，难度飙升

大致流程：

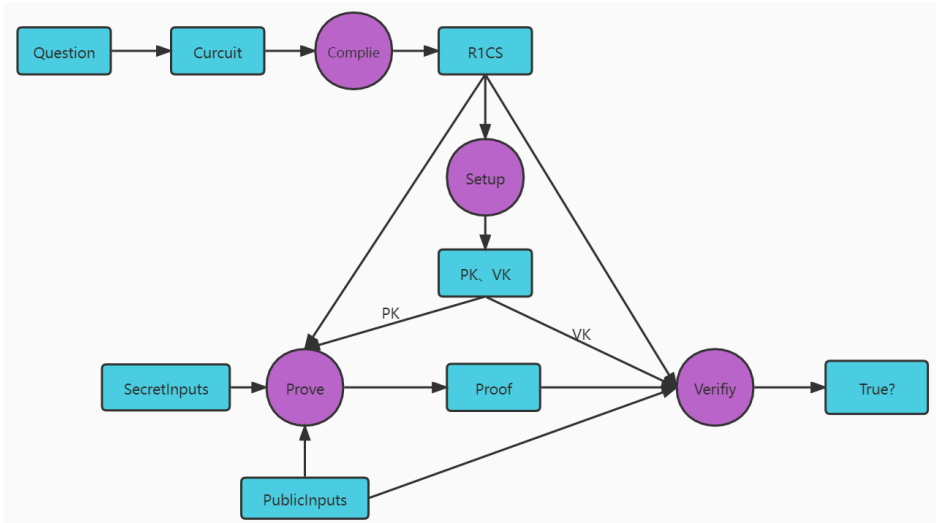
- 将多项式先抽象成算数门（加减乘除）
- 将算数门“拍平”为 **R1CS** 的电路表现形式
- 转化为 **QAP** 的形式，旨在**快速验证**整个电路
- 利用**双线性对 隐藏**解向量 $\vec{s}$
- 生成并输出证明

本章旨在介绍zkSNARK是如何工作的，以及Groth16算法的基础差异，而非直接讲解EOS
但对理解EOS和Siniel是如何工作的至关重要



布尔门	对应算数操作
A AND B	A × B
A OR B	A + B - A×B
NOT A	1 - A

其中 $A, B \in \{0, 1\}$。



Reference: [1] https://zhuanlan.zhihu.com/p/38205067
[2] https://learnblockchain.cn/article/3220
[3] https://github.com/slowmist/zkSnark-Groth16-Getting-Started

R1CS-电路“拍平”

同理可得

等式二 $sym_1 * x - y = 0$ 等价于

```
a=[0,0,0,1,0,0],b=[0,1,0,0,0,0],c=[0,0,0,0,1,0]
```

等式三 $(y + x) * 1 - sym_2 = 0$ 等价于

```
a=[0,1,0,0,1,0],b=[1,0,0,0,0,0],c=[0,0,0,0,0,1]
```

等式四 $(sym_2 + 5) * 1 - (\ out) = 0$ 等价于

```
a=[5,0,0,0,0,1],b=[1,0,0,0,0,0],c=[0,0,1,0,0,0]
```

证明者知道witness $\Leftrightarrow$ 每组拍平后的电路都成立

两个问题：1. 需验证每个拍平后的式子，时间复杂度过高；
2. witness包含于 $\vec{s}$，我们需要隐藏它（witness即为输入）。

对于一个函数

```
def qeval(x):
    y = x**3
    return x + y + 5
```

将它拍平成以下语句：
1. $sym_1 = x * x$
2. $y = sym_1 * x$
3. $sym_2 = y + x$
4. $\sim out = sym_2 + 5$

$\vec{s}$ 为解向量，其结构为 $(\ one, x,\ out, sym_1, y, sym_2)$

对于 $x * x - sym_1 = 0$（即第一步），等价于

```
a=[0,1,0,0,0,0],b=[0,1,0,0,0,0],c=[0,0,0,1,0,0]
```

此时满足 $(\vec{a} \cdot \vec{s}) * (\vec{b} \cdot \vec{s}) - \vec{c} \cdot \vec{s} = 0$

QAP-转为多项式，快速验证

前置知识-拉格朗日插值法：在笛卡尔直角坐标系上，输入n个点，输出一个n次多项式（形式为$f(x)=\sum_{i=0}^{n}a_ix^i$，其中$a_i$为每项系数）

~~把使用R1CS约束后的电路搬到这里来~~

~~把使用R1CS约束后的电路搬到这里来得到~~

```
a=[0,1,0,0,0,0],b=[0,1,0,0,0,0],c=[0,0,0,1,0,0]
a=[0,0,0,1,0,0],b=[0,1,0,0,0,0],c=[0,0,0,0,1,0]
a=[0,1,0,0,1,0],b=[1,0,0,0,0,0],c=[0,0,0,0,0,1]
a=[5,0,0,0,0,1],b=[1,0,0,0,0,0],c=[0,0,1,0,0,0]
```

把它看成一个**4行18列**的大矩阵

按列来取点，横坐标为 $i\in[1,\text{行数}]$

第一 **列** 可以取到点 `(1,0),(2,0),(3,0),(4,5)`

插值得到多项式 $f(x)=0.833x^3-5x^2+9.166x-5$

即数列 $[0.833,-5,9.166,-5]$

同理可得总计 **18** 个类似的四元数列

- A
  $[-5.0, 9.166, -5.0, 0.833]$
  $[8.0, -11.333, 5.0, -0.666]$
  $[0.0, 0.0, 0.0, 0.0]$
  $[-6.0, 9.5, -4.0, 0.5]$
  $[4.0, -7.0, 3.5, -0.5]$
  $[-1.0, 1.833, -1.0, 0.166]$
- B
  $[3.0, -5.166, 2.5, -0.333]$
  $[-2.0, 5.166, -2.5, 0.333]$
  $[0.0, 0.0, 0.0, 0.0]$
  $[0.0, 0.0, 0.0, 0.0]$
  $[0.0, 0.0, 0.0, 0.0]$
  $[0.0, 0.0, 0.0, 0.0]$
- C
  $[0.0, 0.0, 0.0, 0.0]$
  $[0.0, 0.0, 0.0, 0.0]$
  $[-1.0, 1.833, -1.0, 0.166]$
  $[4.0, -4.333, 1.5, -0.166]$
  $[-6.0, 9.5, -4.0, 0.5]$
  $[4.0, -7.0, 3.5, -0.5]$

双线性对-隐藏解向量s

生成最终证明

Groth16：最常用的zkSNARK之一

https://yangzhe.me/2023/10/19/protocol-of-groth16

🔍 Comparison: EOS in zkSNARK Research vs EOS.IO Blockchain Platform

Category	EOS (zkSNARK paper)	EOS.IO (Blockchain Platform)
Full Name	Efficient Outsourcing of SNARKs	Enterprise Operation System
Field	Cryptography, zkSNARKs, Secure Computation	Blockchain Systems, Distributed Ledger
Core Goal	Privacy-preserving delegation of zkSNARK proving	High-performance, fee-free smart contract platform
Open Source	USENIX-2023	GitHub-EOS.IO
Main Technologies	zkSNARKs, PIOP, Polynomial Commitments, Secret Sharing	Blockchain VM, Token Economy, Resource Model

> ⚠️ Note: These two EOS projects are completely unrelated. The zkSNARK EOS is a cryptography research system; the blockchain EOS is an enterprise-grade DApp platform.

Reference: [1] https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa
[2] https://github.com/EOSIO/eos

What are zkSNARKs?

- Prove knowledge of a secret witness w for a statement $F(x, w) = 1$ without revealing w.

Property	Meaning
Zero-Knowledge	The proof **reveals nothing** beyond the fact that the **statement is true**.
Succinct	The proof is **very short** and can **be verified quickly**, regardless of the size of the computation.
Non-interactive	No need for multiple rounds of communication; **a single message** is sufficient.
Argument of Knowledge	Ensures that the prover actually "knows" a valid witness, rather than **faking the proof**.

喵喵喵

The Problem:

- Generating zkSNARKs is computationally expensive (time and memory).
- This limits their use in resource-constrained environments (e.g., mobile devices) and for complex computations.

Existing Approaches & Their Limitations:

- **Local Proving:** Too slow or resource-intensive for many users/applications.
- **Cloud Delegation (e.g., DIZK):** Powerful servers can generate proofs faster, BUT this sacrifices privacy by revealing the secret witness w to the cloud. Problematic for privacy-focused applications like private currencies or decentralized computation.

The Central Question: Can users privately and efficiently outsource zkSNARK proving to untrusted machines?

Eos: Contributions Overview

- **Eos (Efficient Outsourcing of SNARKs):**
  - Privacy-preserving delegation protocols for zkSNARKs with universal setup.
  - Enables a prover (delegator) to outsource proof generation to a set of workers.
- **Key Features & Guarantees:**
  - **Privacy:** No private information (witness `w`) revealed if at least one worker is honest and doesn't collude with others.
  - **Security:** Secure against malicious workers without relying on heavyweight cryptographic tools.
  - **Efficiency (compared to local proving on a recent smartphone):**
    - Reduces end-to-end latency by up to 26x.
    - Lowers delegator's active computation time by up to 1447x.
    - Enables proving up to 256x larger instances.
- **Implementation:** A Rust library, Eos, demonstrating concrete efficiency.

Related Work (Brief Highlights)

- **Trinocchio:** Outsourcing with MPC and zkSNARKs.
  - Differences: Targets a zkSNARK with circuit-specific setup, uses Shamir secret sharing (privacy against n/2 corruptions), semi-honest security.
- **Kanjalkar et al.:** Auditable MPC using Marlin zkSNARKs.
  - Differences: Also uses Shamir secret sharing (n/2 corruptions). Eos protocols are more general for other PIOPs and PC schemes.
- **Ozdemir and Boneh (OB22):** "Collaborative proving" for distributed secrets.
  - Similar insights (e.g., additive homomorphisms for polynomial commitments).
  - **Eos Differences:**
    - Leverages honest delegator for MPC preprocessing (vs. heavyweight crypto in OB22).
    - Uses novel "PIOP consistency checkers" for malicious security (vs. info-theoretic MACs in OB22, which adds overhead).
    - Result: Eos is 6-8x faster and requires 3-5x less communication than OB22 in delegation settings.
- **DIZK:** Distributes prover computation but doesn't hide the witness from workers. Complementary to Eos (workers could use DIZK internally).

Core Cryptographic and Computational Primitives in Eos

The Eos system for delegating zkSNARK provers builds upon several key cryptographic and computational concepts. Understanding these primitives is essential to grasping how Eos achieves its efficiency and privacy goals.

Polynomial Interactive Oracle Proofs (PIOPs)

A **Polynomial Interactive Oracle Proof (PIOP)** is an interactive protocol between a prover and a verifier. It allows the prover to convince the verifier that it knows a valid witness `w` for a given instance `x` and index `i` concerning an indexed relation `R`.

- **Mechanism**: The prover achieves this by sending the verifier **polynomial oracles**. The verifier can then query these oracles at points of its choice and decide whether to accept or reject based on the answers.
- **Components**: A PIOP is typically specified by a tuple `PIOP = (F, k, s, I, P, V)`.
  - `F` is a finite field.
  - `k` is the number of rounds.
  - `s(j)` denotes the number of prover polynomials in the j-th round.
  - The **Indexer (I)** preprocesses the NP index `i` into a set of indexed polynomials during an offline phase. These are available to the prover in full and as oracles to the verifier.
  - The **Prover (P)**, given `(F, i, x, w)`, interacts with the verifier by sending oracle polynomials in each round.
  - The **Verifier (V)**, given `x`, sends messages (challenges) to the prover and queries the prover's polynomials and the indexed polynomials.
- **Properties**: The PIOPs considered in Eos are required to achieve perfect completeness, negligible knowledge soundness error, and zero knowledge.

Polynomial Commitment Schemes (PC Schemes) 🔒

A **Polynomial Commitment (PC) scheme** is a cryptographic primitive that enables a committer to commit to a polynomial and later prove to another party that the committed polynomial evaluates to a claimed value at a specific challenge point.

- **Functionality**: It allows a sender to cryptographically bind themselves to a polynomial `p`. Subsequently, they can provide a proof for any evaluation `p(z) = y`.
- **Algorithms**: A PC scheme typically consists of a tuple of algorithms `PC = (Setup, Trim, Commit, Open, Check)`.
  - `PC.Commit(ck, p; r̄) → C`: Takes a commitment key `ck` and a polynomial `p` (with optional randomness `r̄` if hiding) and outputs a commitment `C`.
  - `PC.Open(ck, C, p, z; r̄) → π`: Takes `ck`, commitment `C`, polynomial `p`, an evaluation point `z`, and randomness `r̄`, then outputs an evaluation proof `π`.
- **Properties**: PC schemes must satisfy completeness, extractability, and (often) hiding.
- **Role in zkSNARKs**: PC schemes are used to compile PIOPs into concretely efficient zkSNARKs. The PIOP prover uses the PC scheme to commit to the polynomial oracles it would normally send. Instead of sending the oracle, it sends the commitment. When the PIOP verifier wants to query an oracle at a point, the prover provides the evaluation along with a PC scheme evaluation proof.

Multi-Party Computation (MPC) and Eos's Approach 🤝

Multi-Party Computation (MPC) allows multiple parties to jointly compute a function over their private inputs without revealing those inputs to each other.

- **Strawman for Delegation**: A straightforward approach to delegating zkSNARK proving would be for the delegator to secret share its witness `w` with the workers, who then use an MPC protocol to securely evaluate the zkSNARK prover algorithm.
- **Shortcomings of Off-the-Shelf MPC**:
  - **Efficiency**: State-of-the-art MPC protocols that achieve malicious security against a dishonest majority often rely on relatively heavyweight public-key cryptography, which has significant computational overhead.
  - **Circuit Complexity**: These MPC protocols typically require expressing the computation as an arithmetic circuit. Directly translating complex zkSNARK prover algorithms (which include operations like elliptic curve multi-scalar multiplications and polynomial arithmetic) into circuits can be prohibitively expensive.
- **Eos's Specialized MPC Design**: Eos overcomes these issues with a specialized approach tailored for delegation.
  - **Leveraging the Honest Delegator**: Since the delegator is an honest participant (it's their computation), Eos uses this trust to simplify the MPC.
    - In collaborative mode, the delegator can help generate correlated randomness (like multiplication triples) needed for the MPC, eliminating

Eos System Overview

- **Target zkSNARKs:** Those constructed from Polynomial Interactive Oracle Proofs (PIOPs) and Polynomial Commitment (PC) schemes (e.g., Marlin).
- **Participants:**
  - **Delegator (D):** Computationally weak party wishing to prove a statement.
  - **Workers ($W_1, \ldots, W_n$):** Powerful (cloud) servers performing the bulk of computation.
- **Protocol Phases:**
  - **Preprocessing Phase:** Witness-independent. Material can be pre-computed.
  - **Online Phase:** Witness-dependent. D sends shares of its private witness `w` and public input `x` to workers.
- **Communication Modes (Fig. 1):**
  - **Isolated Mode:** Each honest worker communicates only with D. (Stronger security guarantees, higher communication/latency costs).
  - **Collaborative Mode:** Workers communicate directly with each other and with D.
- **Threat Model:**
  - Privacy: Witness `w` is hidden if at least one worker is honest and non-colluding.
  - Integrity: Dishonest workers can deviate arbitrarily (malicious security).
- **Witness Reduction:** Delegator performs witness reduction and secret shares the resulting low-level witness.

Core Idea 1: Specialized MPC for Delegation

- **Strawman (Off-the-shelf MPC):** Secret share `w`, then use general MPC (e.g., SPDZ) for prover computation.
  - Inefficient: Heavyweight public-key crypto for correlated randomness for multiplication gates, overhead of authenticated shares for malicious security. Complex prover algorithms are expensive as MPC circuits.
- **Eos Approach: Leverage the Honest Delegator (D)**
  - The delegator is trusted to be honest (it's their own witness).
  - **Collaborative Mode:** D can generate correlated randomness (e.g., multiplication triples) for workers.
  - **Isolated Mode:** D can directly implement the multiplication functionality.
  - This eliminates the need for workers to perform expensive public-key operations for these MPC components.

Core Idea 2: Enforcing Malicious Security Efficiently

- **Challenge:** How to ensure malicious workers don't cheat or learn `w` without costly authenticated shares?
- **Insight:** The computation *is* generating a zkSNARK, which is "error-detecting".
- **Problem with Naive Check:** Delegator simply verifying the final proof isn't enough. Adversary might malleate shares of `w` to produce an invalid proof for a *related* statement, potentially leaking information about `w` by observing if the verification passes/fails.
- **Eos Solution: PIOP Consistency Checkers (Section 5)**
  - Delegator efficiently checks that intermediate polynomials computed (via MPC) by workers are "consistent" with those an honest prover would have computed using *D's actual witness*.
  - This is done by D querying specific evaluations of the secret-shared polynomials and checking them against locally computed values (formalized in Section 5.1, example for MARLIN PIOP in Section 5.2 and Appendix B).
  - Avoids the overhead of general MPC malicious security techniques like authenticated triples.

Core Idea 3: Efficient Circuits for zkSNARK Provers (Section 4)

- **Goal:** Minimize overhead when translating zkSNARK prover algorithms into MPC-friendly circuits.
- **Leveraging Algebraic Structure:** PIOPs and PC schemes involve many polynomial operations.
- **Optimized Circuits for Polynomial Arithmetic (on secret-shared polynomials):**
  - **Linear Operations (Depth 0 for MPC):**
    - Addition, Subtraction (`PolyAdd` - Claim 4.1).
    - FFT / IFFT (Multi-point evaluation/interpolation over smooth subgroups - Claims 4.2, 4.3).
    - Division by a *public* polynomial (`PolyDiv` - Claim 4.6).
  - **Polynomial Multiplication (`PolyMul`) (Depth 1 for MPC - Claim 4.5):**
    - Use FFT: Evaluate p_1, p_2 on a domain, pointwise multiply evaluations, IFFT back. Only pointwise products are actual multiplications in MPC.
- **Optimized Circuits for PC Schemes (e.g., KZG):**
  - Elliptic Curve Operations: Usually expensive in circuits.
  - Eos Insight: Many operations (like Multi-Scalar Multiplication - MSM) are linear if scalars are private but bases are public (Claim 4.7 for MSM). $C_{KZG}.Commit$ has depth 0 (Claim 4.8). $C_{KZG}.Open$ also has depth 0 (Claim 4.9).

Implementation & Key Optimizations (Section 7)

- **Eos Library:**
  - Implemented in Rust.
  - Builds upon the `arkworks` ecosystem.
  - Generic design: Supports various PIOP-based zkSNARKs by using abstractions for secret-shared field elements/polynomials.
- **Performance Optimizations:**
  - **Improved Parallelization:** For FFTs & MSMs (bottlenecks), Eos runs multiple independent operations in parallel, each with an optimal number of threads. (Reduces polynomial commit time by up to 3x and PIOP prover time by up to 4x).
  - **Reduced Delegator Communication (Secret Sharing):** For a vector, send full share to one worker, PRG seeds to others.
  - **Lower Delegator Memory (Triple Generation):** Delegator processes elements in batches (streaming) for constant additional memory use.
  - **Faster Secret Multiplications (Polynomial Products):** Algebraic rewriting of $z_A \cdot z_B = (z'_A + r_A v_H)(z'_B + r_B v_H)$ to reduce FFT sizes needed from 4|H| to 2|H|.
  - **Efficient Scalar-Vector Products:** Specialized MPC triples when one factor is a repeated scalar (reduces per-worker communication from 8|H| to 6|H| field elements with two workers).

Evaluation Setup (Section 8.1)

- **Research Questions:**
  - Q1: Can Eos prove R1CS instances larger than local proving?
  - Q2: What is the overhead (time, communication) of Eos for locally-provable instances?
- **Delegator Setups:**
  - **LAPTOPHB:** Mid-grade laptop, strong network (AWS r4.xlarge, 32GB RAM, 4 cores, 3Gbps network).
  - **LAPTOPLB:** Mid-grade laptop, average network (same HW, but 350Mbps down / 13Mbps up network).
  - **MOBILE:** Smartphone, average Wi-Fi (Google Pixel 4a, 6GB RAM, 350Mbps down / 13Mbps up Wi-Fi).
- **Workers:** Two powerful AWS c5.24xlarge instances (192GB RAM, 96-core CPU) in different regions (simulating trust domains).
- **Baselines for Comparison:**
  - **DEL:** Delegator generates zkSNARK locally (private but slow).
  - **WORKER:** A single worker generates zkSNARK locally (fast but not private).

Evaluation Results: Proving Larger Instances (Q1 - Section 8.2, Table 1)

- **Key Finding:** Eos consistently enables proving larger instances within the same delegator time/memory budgets.
- **Memory Budget (e.g., 3GB for Delegator):**
  - All setups (LAPTOPHB, LAPTOPLB, MOBILE): Eos proves up to **256x larger** instances (e.g., local 2^{17} vs Eos 2^{25}).
- **Time Budget (e.g., 100s for Delegator):**
  - MOBILE (Collaborative): **32x larger** instances (2^{16} vs 2^{21}).
  - LAPTOPHB (Collaborative/Isolated): **8x larger** instances (2^{18} vs 2^{21}).
- **Collaborative vs. Isolated:** Collaborative often supports larger sizes than isolated if delegator communication is the bottleneck (LAPTOPLB, MOBILE).

Evaluation Results: End-to-End Latency (Q2 - Section 8.3.1, Fig. 5)

- **LAPTOPHB (High Bandwidth):**
  - Isolated: 5.5x–8.5x faster than DEL.
  - Collaborative: 5.5x–9x faster than DEL.
  - Overhead vs. WORKER (non-private baseline): Only 1.1x–1.9x, demonstrating low computational overhead with sufficient bandwidth.
- **LAPTOPLB (Low Upload Bandwidth):**
  - Isolated: 1.7x faster than DEL.
  - Collaborative: 5.7x faster than DEL (delegator online cost lower as some communication moves to preprocessing).
- **MOBILE (Low Bandwidth & CPU):**
  - Isolated: 7.6x–8.5x faster than DEL.
  - Collaborative: **22x–26x faster** than DEL.
- **Summary:** Eos provides speedups in all setups. Collaborative generally better if preprocessing is feasible.

Evaluation Results: Delegator Online Time & Communication (Q2)

- **Delegator Online Time (Active Participation Time - Fig. 6):**
  - Significantly lower in Eos than DEL baseline.
  - LAPTOPHB (Collaborative): Reduces delegator online time by at least 592x.
  - MOBILE (Collaborative): Reduces delegator online time by at least 96x.
  - (Abstract/Intro also states up to 1447x reduction in delegator's active computation time for a smartphone).
  - Collaborative mode generally has much lower delegator online time than isolated.
- **Cost of Preprocessing (Collaborative Mode - Fig. 7):**
  - Dominated by communication if delegator bandwidth is low for large instances.
  - Can be done opportunistically (e.g., when on a better network).
- **Communication Overhead (Delegator-Worker - Fig. 8):**
  - Grows linearly with instance size.
  - Isolated mode incurs higher online communication cost than collaborative (even when accounting for preprocessing).

Evaluation Results: Comparison with OB22 (Section 8.4, Fig. 9)

- Comparison done on LAPTOPHB setup (most favorable for OB22 due to bandwidth).
- **Eos Performance vs. OB22:**
  - Latency: Eos is **6–8x faster**.
  - Communication:
    - Delegator-Worker: Eos requires ~3x less.
    - Worker-Worker: Eos requires ~5x less.
- **Reason for Improvement:** Eos's delegation-specific design choices (leveraging honest delegator, PIOP consistency checkers) are more efficient in this setting than general collaborative proving tools.

Conclusion & Future Work

- **Eos successfully addresses the challenge of private and efficient delegation of zkSNARK provers**.
- **Key Achievements:**
  - Significant performance improvements (latency, delegator cost, scalable instance size) across varied settings.
  - Strong privacy (if ≥ 1 worker honest) and malicious security guarantees without heavyweight crypto tools.
- **Novel Techniques Introduced:**
  - Specialized MPC leveraging the honest delegator.
  - PIOP consistency checkers for efficient malicious security.
  - Optimized arithmetic circuits for zkSNARK prover components.
- **Impact:** Makes advanced cryptographic privacy practical for a wider range of applications and devices.
- **(Optional: Future Work ideas, if any mentioned or obvious extensions)**
  - Investigating adaptation of honest-majority protocols for privacy-preserving delegation.
  - Further exploring streaming witness generation for unbounded instance sizes (Remark 8.1).

difference between zkSNARK & zkSTARK

特性	zkSNARK	zkSTARK
全称	Zero-Knowledge Succinct Non-interactive Argument of Knowledge	Zero-Knowledge Scalable Transparent Argument of Knowledge
信任设置	需要可信预设（Trusted Setup）	不需要可信预设
证明大小	几百字节（较小）	数十KB（较大）
验证速度	快	非常快（更快）
生成速度	较慢（尤其对复杂电路）	更快，尤其适合大规模证明
抗量子性	不抗量子攻击	抗量子攻击
加密原语	椭圆曲线密码学（如pairing-based cryptography）	基于哈希函数（如Merkle树、FRI）
适用场景	常用于以太坊隐私协议（如Zcash、Aztec）	常用于可扩展性方案（如StarkNet、zkRollup）

特性	zkSNARK	zkSTARK
可信设置	需要可信设置（Trusted Setup）	✅ 无需可信设置（透明）
数学基础	椭圆曲线、对数硬问题（需复杂密码学工具）	基于哈希函数（如 Merkle Tree），更量子安全
证明大小	小（几百字节）	较大（几十 KB 到几 MB）
验证速度	极快	快，但比 zkSNARK 稍慢
证明生成速度	较慢	更快，支持大规模并行
量子抗性	❌ 不抗量子攻击	✅ 抗量子攻击

Thank You / Q&A

GitHub: https://github.com/coperlm

Email: TyWang2005@outlook.com