Eos: Efficient Private Delegation of zkSNARK Provers

Alessandro Chiesa, *UC Berkeley and EPFL;* Ryan Lehmkuhl, *MIT;*
Pratyush Mishra, *Aleo and University of Pennsylvania;* Yinuo Zhang, *UC Berkeley*

https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

EOS: Efficient Private Delegation of zkSNARK Provers

Alessandro Chiesa
UC Berkeley & EPFL

Ryan Lehmkuhl
MIT[*]

Pratyush Mishra
Aleo & University of Pennsylvania[†]

Yinuo Zhang
UC Berkeley

Abstract

Succinct zero knowledge proofs (i.e. zkSNARKs) are powerful cryptographic tools that enable a prover to convince a verifier that a given statement is true without revealing any additional information. Their attractive privacy properties have led to much academic and industrial interest.

Unfortunately, existing systems for generating zkSNARKs are expensive, which limits the applications in which these proofs can be used. One approach is to take advantage of powerful cloud servers to generate the proof. However, existing techniques for this (e.g., DIZK) sacrifice privacy by revealing secret information to the cloud machines. This is problematic for many applications of zkSNARKs, such as decentralized private currency and computation systems.

In this work we design and implement *privacy-preserving delegation protocols* for zkSNARKs with universal setup. Our protocols enable a prover to outsource proof generation to a set of workers, so that if at least one worker does not collude with other workers, *no private information* is revealed to *any* worker. Our protocols achieve security against malicious workers without relying on heavyweight cryptographic tools.

We implement and evaluate our delegation protocols for a state-of-the-art zkSNARK in a variety of computational and bandwidth settings, and demonstrate that our protocols are concretely efficient. When compared to local proving, using our protocols to delegate proof generation from a recent smartphone (a) reduces end-to-end latency by up to $26\times$, (b) lowers the delegator's active computation time by up to $1447\times$, and (c) enables proving up to $256\times$ larger instances.

1 Introduction

Zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs) are cryptographic proofs that enable a prover $\mathcal{P}$ to convince a (computationally weak) verifier $\mathcal{V}$ of statements of the form "Given a function F and a public input $\mathbb{x}$, there exists a private *witness* $\mathbb{w}$ such that $F(\mathbb{x},\mathbb{w}) = 1$". zkSNARKs satisfy two key properties that make them attractive for applications: *succinctness* and *zero knowledge*. Succinctness ensures that the cryptographic proof is *small* (a few kilobytes) and *easy to verify* (a few milliseconds), regardless of the complexity of F. Zero knowledge ensures that the cryptographic proof reveals *no information* about the witness $\mathbb{w}$. Together, these properties have motivated a number of zkSNARK constructions [27, 25, 6, 37, 4, 3, 10, 2, 33, 24, 11, 14, 15, 42], new applications relying on zkSNARKs for privacy and efficiency [1, 31, 34, 45, 20, 43, 8, 9, 32, 30], as well as industrial deployments of zkSNARKs [47, 35, 23].

The attractive properties of zkSNARKs come at a cost: proving correct even a simple computation requires significantly more time and memory than is required for the computation itself. This overhead arises for two reasons. First, the prover $\mathcal{P}$ "arithmetizes" the computation F, which (often) involves expressing it as an arithmetic circuit C that is much larger than the description of F. Next, for many popular zkSNARKs [26, 24, 14], $\mathcal{P}$ must perform expensive operations whose time and space complexity grow at least linearly in $|C|$.

One way to prove large computations would be to outsource the prover computation to more powerful machines on cloud platforms (such as Amazon EC2 or Microsoft Azure). However, this approach comes at the cost of *privacy*: the machines in the cloud learn the witness $\mathbb{w}$. This is problematic for privacy-focused applications of zkSNARKs, such as private payments [1], private smart contracts [31, 9], and anonymous credentials [20, 40]. In these applications, revealing the witness (which contains private information) to cloud machines could hurt the privacy and anonymity of users.

A concrete illustration of this efficiency-privacy dilemma is easily seen in a number of recent decentralized ledger systems [9, 38] that leverage the privacy and succinctness properties of zkSNARKs to enable users to privately prove that they correctly executed a smart contract over their private data (such as sender and recipient identities, currency amounts, type of computation, and so on). However, the overhead of generating these zkSNARKs has limited users to using only simple smart contracts that can be proven on users' machines, thus hindering the adoption of these systems.

There is thus a need for methods that enable users to generate zkSNARK proofs cheaply and privately. In this work, we investigate one such approach, and ask the following question:

Can users privately and efficiently outsource zkSNARK proving to untrusted machines?

1.1 Contributions

We provide a positive answer to this question by designing and implementing protocols that enable *private delegation* of the

[*]Work partially done while at UC Berkeley.
[†]Work partially done while at UC Berkeley.

prover in state-of-the-art universal-setup zkSNARKs [33, 24, 14]. Our protocols are secure in a strong malicious-security threat model and, when applied to the recent zkSNARK of [14], enable proving larger instances while also reducing the end-to-end proving time. We detail our contributions below.

Delegating PIOP-based zkSNARKs. We construct a delegation protocol for a popular class of zkSNARKs that are constructed from two components: *polynomial interactive oracle proofs* (PIOPs) [24, 14, 11] and *polynomial commitment schemes* (PC schemes) [29] (we review these below). This class contains many efficient zkSNARKs [33, 24, 14, 13]. We design our protocol in two steps. First, we construct an efficient, delegation-specific MPC protocol that uses a new low-overhead technique for checking that the output proof corresponds to the delegated witness. Then, we design efficient arithmetic circuits for the various components of provers of PIOP-based zkSNARKs. We illustrate the efficiency of our construction by instantiating it with subcircuits for the PIOP and PC scheme underlying the MARLIN zkSNARK [14] to obtain a delegation protocol for this zkSNARK that is significantly more lightweight than even the closely related protocol of [36] that uses off-the-shelf MPC protocols.

Implementation and evaluation results. We contribute a Rust library EOS[1] that implements our delegation protocols. We minimize implementation effort and code duplication by designing abstractions for secret-shared field elements and polynomials, that allow us to implicitly construct subcircuits for PIOP and PC schemes by utilizing existing *plaintext* implementations of the same. Furthermore, we construct our zkSNARK prover circuits in a generic manner, allowing the underlying PIOP and PC scheme subcircuits to be swapped out easily. When combined with our generic MPC protocol, this allows our library to delegate any PIOP-based zkSNARK given just the corresponding plaintext implementations. We provide concrete instantiations for our interfaces by implementing a delegation protocol for the zkSNARK of [14].

We conduct comprehensive experiments to evaluate the performance of our implementation. In short, our experiments demonstrate that in a variety of bandwidth and computational settings, our delegation protocols enable proving *larger instances* in *less time* and with *less memory*. For example, when delegating from a smartphone to powerful cloud machines, our protocols reduce proof generation time by up to $26\times$, lower the active computation time of the smartphone by up to $1447\times$, and enable proving $256\times$ larger instances. These benefits kick in for circuits as small as 2^{15} gates. See Sections 7 and 8 for a detailed discussion.

1.2 Related work

Trinocchio [41] studies the problem of *privacy-preserving verifiable computation*, where a delegator wishes to privately outsource a computation to a set of workers. Their protocol consists of two components: (a) an MPC protocol that performs the computation; and (b) a zkSNARK delegation protocol that uses the result of the MPC to generate a proof that the computation was performed correctly. There are several differences between their delegation subprotocol and our protocols: (a) they target a zkSNARK with circuit-specific setup [37], while we target a recent class of zkSNARKs with universal setup; (b) they use Shamir secret sharing and provide privacy against $n/2$ corruptions, while we use simpler additive secret sharing and provide security against $n-1$ corruptions; (c) they require the delegator to reconstruct the Shamir secret shares of the final proof at the end, while our collaborative setting does not require this additional step; (d) their protocol has semi-honest security, while ours has malicious security.

Kanjalkar et. al. [28] design a protocol for *auditable MPC*, where the goal is to prove to a third-party auditor that an execution of an MPC protocol was performed correctly. To achieve auditability, this protocol uses a subprotocol that generates a MARLIN zkSNARK [14] when the witness is secret-shared across the MPC participants. While some techniques in this subprotocol are similar to those in our delegation protocol (when specialized to the zkSNARK of [14]), our protocols are more general, and work with other PIOPs and PC schemes. From a technical perspective, our protocol uses additive secret sharing, while that of [28] (like Trinocchio) uses Shamir secret sharing (SSS) [44], and therefore is secure only against $n/2$ corruptions. Changing these systems to support $n-1$ corruptions would change their performance profile, as they rely on properties of SSS to implement secret multiplications.

The recent work of Ozdemir and Boneh [36] (henceforth OB22) constructs "collaborative proving" protocols that allow a set of parties that have secret shares of an NP witness to jointly generate zkSNARK proofs with respect to that witness. They present protocols for three zkSNARKs [26, 24, 14]. To design efficient protocols, they leverage similar insights as we do. For example, both protocols rely on additive homomorphisms to minimize the overhead of computing polynomial commitments in a distributed setting. However, despite high-level similarities, the protocols provide different efficiency and security guarantees:

- **Protocol design:** Our protocols leverage the honest delegator to minimize the cost of preprocessing in the MPC protocol, while OB22 must rely on heavyweight cryptography to do the same. Furthermore, our protocols minimize overhead of malicious security by introducing the notion of *PIOP consistency checkers*, while OB22's use of information-theoretic MACs leads to a $2\times$ increase in witness-dependent computation and worker communication, and a $(2n-1)\times$ increase in delegator communication.
- **Implementation differences:** The implementation of OB22 does not describe or implement many of the optimizations in Section 7.1. On the other hand, they implement distributed proving for more zkSNARKs than we do.

[1]EOS stands for **E**fficient **O**utsourcing of **S**NARKs.

Overall, these differences result in concrete performance benefits: our protocols are 6–8× faster and require 3–5× less communication. See Section 8.4 for a detailed comparison.

Another recent work [19] constructs a collaborative prover for IOP-based SNARKs [3]. Their protocol for collaborative IOP proving is similar to our protocol for PIOP delegation, but, as with [36], the different application settings (distributed computation vs delegation) results in concretely different protocols and optimization choices. For example, they too must rely on heavyweight cryptography to achieve malicious security. As they do not implement or evaluate their protocols, we cannot provide a quantitative comparison. However, due to the similarities of their setting with that of [36], it is likely that their protocol suffers from similar overheads.

Block and Garman [7] introduce protocols for distributing the prover's work in the honest-majority setting, where at least $\frac{n}{2}+1$ out of n workers are assumed to be honest. Their protocols achieve an asymptotic speedup of $n\times$ compared to the single-prover solution. However, they achieve this while sacrificing privacy: even honest workers can see (portions of) the witness. Furthermore, they neither implement nor evaluate their protocol, making quantitative comparisons difficult. It would be interesting to investigate whether their protocol can be adapted to provide privacy-preserving delegation.

DIZK [46] distributes the prover computation for the zkSNARK in [26] across a cluster of machines, with the aim of optimizing the space complexity of each machine, and without hiding the witness from these machines. DIZK is complementary to our work, as we can improve scalability of our protocol by having each worker use DIZK's techniques to outsource its computations. This preserves privacy because the compute cluster would only see the worker's secret shares.

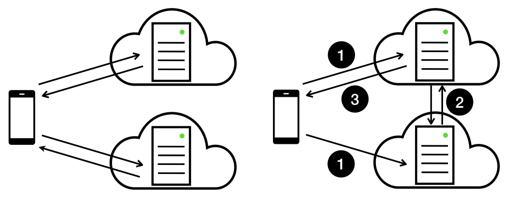
2 Construction overview

We consider delegation protocols where the zkSNARK prover $\mathcal{P}$ delegates its proof computation to n different workers. We focus on zkSNARKs constructed via the methodology in [14], i.e., by composing a PIOP and PC scheme.

Protocol setting. Our protocols are in the preprocessing model. That is, the delegating prover (henceforth $\mathcal{D}$) outsources its work to a set of n remote workers $\mathcal{W}_1,\dots,\mathcal{W}_n$ in two phases: a preprocessing phase that is independent of the witness for which a proof is being generated, and an online phase that is witness-dependent. In the latter phase, $\mathcal{D}$ sends to each worker the (short) public input $\mathbb{x}$ and secret shares of the (large) private witness $\mathbb{w}$, and the parties then use the preprocessing material to jointly compute the zkSNARK proof. Our delegation protocols work in two different modes (Fig. 1):

- *Isolated mode:* Each honest worker communicates only with the delegator $\mathcal{D}$, and not with other workers. $\mathcal{D}$ is online throughout the protocol execution.
- *Collaborative mode:* Workers communicate directly with each other, and with the delegator $\mathcal{D}$.

Isolated mode provides stronger security guarantees than collaborative mode, as workers are not even aware of each other, but pays for this by incurring concretely higher communication and latency costs (see Section 8). In both modes, all communication occurs over authenticated secure channels.



(a) (b)

Figure 1(a) Isolated delegation (b) Collaborative delegation

Threat model. Our protocols guarantee that the private witness $\mathbb{w}$ is completely hidden from *all workers* if at least *one* worker is honest and does not collude with the others. Dishonest workers can deviate arbitrarily from the protocol.

Indexed relations. An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathbb{i},\mathbb{x},\mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness. In this paper, we consider zkSNARKs for the popular R1CS relation $\mathcal{R}_{\mathrm{R1CS}}$, which is the set of triples $(\mathbb{i},\mathbb{x},\mathbb{w}) = ((\mathbb{F},M,A,B,C),x,w)$ where $\mathbb{F}$ is a finite field, and A,B,C are $M\times M$ matrices over $\mathbb{F}$ such that $Az\circ Bz = Cz$ for $z := (x,w) \in \mathbb{F}^M$. (Here "$\circ$" denotes the entry-wise product.)

Remark 2.1 (witness reduction)**.** zkSNARK proving contains a "witness reduction" step that converts a high-level NP witness (e.g., a hash preimage) to a low-level witness (e.g., wire assignments in the hash circuit) that is suitable for proving. In our delegation protocols, the delegator performs witness reduction and secret shares the resulting low-level witness among the workers. An alternative definition could require the workers to carry out witness reduction themselves via MPC. However, we avoid this approach because, for many computations of interest, the cost of executing witness reduction in MPC will outstrip the cost of doing it on the delegator.

2.1 Background: universal-setup zkSNARKs

The methodology of [14] constructs zkSNARKs from two components: information-theoretic *polynomial interactive oracle proofs* (PIOPs), and cryptographic *polynomial commitment schemes* (PC schemes). Below we review these components and how they are combined to obtain zkSNARKs, focusing on the computations of the zkSNARK prover.

A polynomial interactive oracle proof (PIOP) for an indexed relation $\mathcal{R} = \{(\mathbb{i},\mathbb{x},\mathbb{w})\}$ is an interactive protocol between a prover **P** and a verifier **V** that allows **P** to convince **V** that it "knows" a valid witness $\mathbb{w}$ such that $(\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}$. **P** does so by sending the verifier *polynomial oracles*. The verifier can then query these at points of its choice, and decide to accept or reject based on the answers.[2]

A polynomial commitment (PC) scheme is a cryptographic primitive that allows a committer to commit to a polynomial, and then later prove to another party that the committed polynomial evaluates to a claimed evaluation at a challenge point.

Constructing zkSNARKs from PIOPs and PC schemes. A zkSNARK is a tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$. Below we focus on the prover $\mathcal{P}$ and the verifier $\mathcal{V}$. The methodology of [14] obtains a zkSNARK from a PIOP and a PC scheme by first constructing an *interactive argument* as follows. The interactive argument prover $\mathcal{P}$ and verifier $\mathcal{V}$ respectively invoke the PIOP prover **P** and verifier **V**, but in each round, instead of directly sending the polynomial oracles output by **P**, $\mathcal{P}$ instead commits to these polynomials using the PC scheme, and sends the resulting commitments to $\mathcal{V}$. After the interaction, $\mathcal{V}$ declares its queries to the committed polynomials, and $\mathcal{P}$ replies with the desired evaluations along with an evaluation proof attesting to their correctness relative to the commitments. To obtain a zkSNARK, one can apply the Fiat–Shamir transform [22] to this interactive argument.

2.2 Delegating zkSNARKs

Strawman: use off-the-shelf MPC protocols. A straightforward approach to delegating proving would be for the delegator to secret share its witness with the workers, who would then invoke a suitable MPC protocol to securely evaluate the prover. Unfortunately, this approach has significant shortcomings in terms of concrete efficiency for two reasons.

First, existing state-of-the-art MPC protocols achieving malicious security against a dishonest majority of workers rely on relatively heavyweight public-key cryptography, which has a non-trivial computational overhead. Second, these MPC protocols require expressing the computation as an arithmetic circuit. A straightforward translation of the zkSNARK prover algorithm for popular zkSNARKs [24, 14] to circuit format would require expressing complex operations such as elliptic curve multi-scalar multiplications and polynomial arithmetic as circuits, and this would be prohibitively expensive.

We show how to overcome both these issues below. Our starting point is the relatively efficient SPDZ protocol [18].

Step 1: Designing specialized protocols for delegation. The SPDZ protocol suffers from overheads primarily due to two reasons: the need for expensive public-key cryptography to generate correlated randomness for multiplication gates, and the use of authenticated shares for malicious security.

We tackle the first cost by leveraging the fact that in the delegation setting, there is a trusted party that is guaranteed to be honest: the delegator. We can use the delegator to either generate the correlated randomness (in collaborative mode), or directly implement the multiplication functionality (in isolated mode), thus eliminating expensive public-key cryptography.

To eliminate the cost associated with authenticated shares, we notice that our computation is "error-detecting": we are generating a zkSNARK. We expand on this idea and develop new techniques to ensure malicious security next.

Step 2: Enforcing malicious security. A natural first step is to use the zkSNARK to succinctly check that the MPC execution is correct by having the delegator verify the zkSNARK produced by the workers, and reject if it is invalid. Roughly, the security argument would be that because the zkSNARK is knowledge-sound, the adversary cannot produce an invalid proof by deviating from the protocol (and hence using an invalid witness). This idea *almost* works, but is insufficient by itself. The adversary can attempt to *malleate* its shares of the delegator's valid witness w to produce a proof of a *related* statement. Even if the resulting proof is invalid, it can leak information about w. Consider the example of a booleanity constraint $b \cdot (1 - b) = 0$ for a variable b, and an adversarial malleation that adds 1 to b: if b's original value was 0, then the constraint is still satisfied, and the proof is valid, whereas if the original value was 1, then the constraint is violated, and the proof is invalid. Thus by observing whether the proof verifies or not, the adversary can learn the value of b.

However, while we cannot leverage the succinct verification property of the zkSNARK, we can still try to use the succinct verification properties of the *underlying components* of the zkSNARK, the PIOP and the PC scheme. We do so by introducing the notion of a *consistency checker* for the PIOP, and combine this with the PC scheme to enable the delegator to efficiently check that the polynomials computed during the MPC execution are consistent with those that an honest prover would have computed. We formalize this intuition and construct an MPC scheme that does not rely on authenticated triples at all. See Sections 5 and 6.3 for details.

Step 3: Designing efficient circuits for zkSNARK provers. To design efficient circuits for proving PIOP-based zkSNARKs, we leverage the fact that the underlying building blocks, namely PIOPs and PC schemes, are highly algebraic:

- *PIOP:* The operations performed by the provers of many popular PIOPs [33, 24, 14, 13] consist of a few common operations on polynomials. By optimizing the number of multiplications in, and the multiplicative depth of, circuits for these operations, we can improve the corresponding parameters of the PIOP prover circuit. We now describe some of these operations and how we optimize their circuits.
  - *Multipoint evaluation and interpolation* of polynomials over smooth multiplicative subgroups (see Section 3) are key building blocks of PIOP provers. Efficient algorithms for these can be derived from FFTs (resp. IFFTs), and since the latter is a linear operation with respect to the

[2]The PIOPs we consider in this work possess an additional algorithm called an *indexer* that preprocesses the NP index into *index* polynomials. The verifier has oracle access to these polynomials also.

coefficients (resp. evaluations) of the polynomial, the corresponding circuit contains *no multiplication gates*.

– *Affine polynomial operations* use no multiplication gates.
– *Division by a public polynomial:* Dividing a polynomial p by a public polynomial g is a linear operation on the coefficients of p, and thus requires no multiplication gates.
– *Multiplication of two polynomials:* To multiply two polynomials $p, q \in \mathbb{F}^{\leq d}[X]$, one can rely on efficient FFT-based methods as follows: (a) evaluate p and q on a set S of size greater than $2d$ via the aforementioned techniques; (b) compute the point-wise product of the evaluations, to obtain the evaluations of the product polynomial $r \in \mathbb{F}^{\leq 2d}[X]$; and finally, (c) invoke the IFFT to interpolate the evaluations and recover r.
  Since the FFT and IFFT steps are linear, we only require $\sim 2d$ multiplication gates, for Step (b). Moreover, because this point-wise product can be computed in parallel, the multiplicative depth of this circuit is 1.

• *PC schemes:* Unlike PIOP provers, the algorithms in PC schemes require operations over elliptic curves. At first glance, this is problematic as it requires us to express these operations as circuits. However, we leverage and extend past abstractions for *elliptic-curve* circuits [36] that exploit the fact that addition in elliptic curve groups corresponds to addition in the corresponding scalar field. This allows us to express the commitment and opening algorithms of popular PC schemes such as that of [29] as low-depth circuits.

By combining these optimizations, we are able to obtain circuits for the zkSNARK prover whose computational overhead nearly matches native execution. See Section 4 for details. Overall, these optimizations lead to excellent concrete efficiency, as we demonstrate in Section 8.

3 Preliminaries

We assume that all public parameters have length at least λ, so that algorithms that receive these can run in time poly(λ).

Algebraic preliminaries. For a finite field $\mathbb{F}$, a *smooth multiplicative subgroup* of $\mathbb{F}$ is a subgroup of the multiplicative group $\mathbb{F}^*$ having order 2^k for some $k \in \mathbb{N}$. The structure of such a subgroup enables efficient polynomial arithmetic. For a function $f\colon H \to \mathbb{F}$ we denote by $\hat{f}$ the univariate polynomial over $\mathbb{F}$ with degree less than $|H|$ such that $\hat{f}(a) = f(a)$ for every $a \in H$. $\hat{f}$ is then called the *low-degree extension* of f.

Random oracles. A *random oracle* ρ is a function sampled uniformly from the set of functions from $\{0,1\}^*$ to $\{0,1\}^\lambda$.

3.1 Circuit model

Most efficient MPC protocols require expressing the computation as a circuit. In our case, the computation being performed (zkSNARK proving) requires not only standard addition and multiplication operations, but also operations such as (elliptic curve) group arithmetic and random oracle calls. To capture this richer functionality, we extend prior work [36] and consider an extended circuit model that includes these operations.

Definition 3.1 (Oracle elliptic-curve circuit)**.** *Let $\mathbb{F}$ be a finite field of prime order q, and let $\mathbb{G}$ be the q-order subgroup of an elliptic curve. Then, an* **oracle elliptic-curve circuit** $C^{\rho}_{\mathbb{F},\mathbb{G}}$ *is an arithmetic circuit where (a) each wire takes on values either in $\mathbb{F}$ or $\mathbb{G}$, and has either public or private visibility, and (b) each gate is one of the following:*

• $\mathsf{Add}_{\mathbb{F}}(w_i \in \mathbb{F}, w_j \in \mathbb{F}) \to w_k \in \mathbb{F}$ *Set $w_k := w_i + w_j$, where "$+$" denotes addition in $\mathbb{F}$.*
• $\mathsf{Mul}_{\mathbb{F}}(w_i \in \mathbb{F}, w_j \in \mathbb{F}) \to w_k \in \mathbb{F}$ *sets $w_k := w_i \cdot w_j$, where "$\cdot$" denotes multiplication in $\mathbb{F}$.*
• $\mathsf{Add}_{\mathbb{G}}(w_i \in \mathbb{G}, w_j \in \mathbb{G}) \to w_k \in \mathbb{G}$ *sets $w_k := w_i + w_j$, where "$+$" denotes addition in $\mathbb{G}$.*
• $\mathsf{Mul}_{\mathbb{G}}(w_i \in \mathbb{F}, w_j \in \mathbb{G}) \to w_k \in \mathbb{G}$ *sets $w_k := w_i \cdot w_j$, where "$\cdot$" denotes scalar multiplication in $\mathbb{G}$. At least one of w_i or w_j must have public visibility.*
• $\mathsf{RO}(\{w_i\}_i) \to w_k \in \mathbb{F}$ *sets $w_k := \rho(\{w_i\}_i)$. Each w_i must be public.*
• $\mathsf{Reveal}(w_i) \to w_k$ *sets $w_k := w_i$ and makes w_k public.*

For every gate except Reveal, *the output w_k is public if and only if all input wires are public.*

When $\mathbb{F}$, $\mathbb{G}$, and ρ are obvious from context, we will omit them, and instead write C. The *circuit depth* $\textsc{depth}(C)$ is the maximum number of $\mathsf{Mul}_{\mathbb{F}}$ and Reveal gates in a path from an input wire to an output wire, when all input wires to these gates are private. This is strictly larger than traditional notions of *multiplicative depth*. We use this more general notion because the round complexity of our protocol scales with circuit depth, and not multiplicative depth.

3.2 Additive secret sharing

Additive secret sharing enables sharing a message m among n parties so that obtaining $n-1$ shares reveals no information about m. Formally, a *secret sharing* scheme for a finite field $\mathbb{F}$ is a pair $\mathsf{SS} = (\mathsf{Share}, \mathsf{Combine})$ with the following syntax.

• *Sharing:* On input a message $m \in \mathbb{F}$, and a number of parties n, Share outputs n secret shares $[[\![m]\!]_i]_{i=1}^n$.
• *Combining:* On input secret shares $[[\![m]\!]_i]_{i=1}^n$, Combine combines these shares to compute the message $m \in \mathbb{F}^n$.

When the number of parties n is clear from context, we omit it when invoking SS.Share. We abuse notation and denote component-wise secret shares of the coefficients of a polynomial $p \in \mathbb{F}^{\leq d}[X]$ by $[[\![p]\!]_i]_{i=1}^n$. Additive secret sharing preserves additive homomorphisms: for messages $m_1, m_2 \in \mathbb{F}$, for all $i \in n$, $[\![m_1 + m_2]\!]_i = [\![m_1]\!]_i + [\![m_2]\!]_i$.

3.3 Polynomial commitments

A **polynomial commitment scheme** enables a sender to commit to a polynomial p and then later prove the correct evaluation of p at a desired point. It consists of a tuple of algorithms $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ satisfying completeness, extractability, and hiding (see [14] for definitions of these). We are interested in Commit and Open:

- $\mathsf{PC.Commit}^{\rho}(\mathsf{ck}, p; \bar{p}) \to C$. On input the commitment key ck, a polynomial p over the field $\mathbb{F}$, PC.Commit outputs a commitment C to the polynomial p. The randomness $\bar{p}$ is used if the commitment C is hiding.
- $\mathsf{PC.Open}^{\rho}(\mathsf{ck}, C, p, z; \bar{p}) \to \pi$. On input the commitment key ck, a commitment C, the polynomial p committed inside C, an evaluation point $z \in \mathbb{F}$, and commitment randomness $\bar{p}$, PC.Open outputs an evaluation proof π.

3.4 Polynomial interactive oracle proof

A **polynomial interactive oracle proof** (PIOP) for an indexed relation $\mathscr{R}$ is an interactive protocol specified by a tuple $\mathsf{PIOP} = (\mathbb{F}, \mathsf{k}, \mathsf{s}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ where $\mathbb{F}$ is a finite field, k is the number of rounds, $\mathsf{s}(j)$ is the number of prover polynomials in the j-th round, and $\mathbf{I}$, $\mathbf{P}$, $\mathbf{V}$ are algorithms described next.

In an offline phase, the indexer $\mathbf{I}$ preprocesses the NP index $\mathbb{i}$ into a set of *indexed polynomials* that are made available to the prover $\mathbf{P}$ (in full) and to the verifier $\mathbf{V}$ (as oracles).

During the online phase, $\mathbf{P}(\mathbb{F}, \mathbb{i}, \mathbb{x}, \mathbb{w})$, in each round $j \in [\mathsf{k}]$, receives a message $\mu_j \in \mathbb{F}^*$ from $\mathbf{V}(\mathbb{x})$ and replies with $\mathsf{s}(j)$ oracle polynomials $p_{j,1}, \ldots, p_{j,\mathsf{s}(j)} \in \mathbb{F}[X]$. $\mathbf{V}$ can query these polynomials and the indexed polynomials. A query consists of a location $z \in \mathbb{F}$ for an oracle $p_{i,j}$, and its corresponding answer is $p_{i,j}(z) \in \mathbb{F}$. After the interaction, the verifier accepts or rejects. Every PIOP we consider in this paper is required to achieve perfect completeness, negligible knowledge soundness error, and zero knowledge. See [14] for details.

3.5 zkSNARKs

A *succinct preprocessing non-interactive argument of knowledge* in the random oracle model (ROM) for an indexed relation $\mathscr{R}$ is a tuple of algorithms $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ satisfying completeness, knowledge soundness, succinctness, and zero knowledge. The indexer $\mathcal{I}$ preprocesses the NP index $\mathbb{i}$ into index-specific proving (ipk) and verification (ivk) keys. The prover $\mathcal{P}$, on input ipk, an instance $\mathbb{x}$, and a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathscr{R}$, outputs a proof π which can be checked by the verifier $\mathcal{V}$ when given as input ivk and $\mathbb{x}$.

Constructing zkSNARKs from PIOPs and PC schemes. [14] constructs a zkSNARK from a PIOP and a PC scheme as follows. First, the argument indexer $\mathcal{I}$, on input $\mathbb{i}$, invokes the PIOP indexer $\mathbf{I}$ to obtain the indexed polynomials, and commits to these using PC.Commit. It then constructs ipk out of these polynomials and ck, and sets ivk to be the commitments.

The interactive argument prover $\mathcal{P}$ and verifier $\mathcal{V}$ respectively invoke the PIOP prover $\mathbf{P}$ and verifier $\mathbf{V}$. In each round, instead of directly sending the polynomial oracles output by $\mathbf{P}$, $\mathcal{P}$ instead commits to these polynomials via PC.Commit, and sends the resulting commitments to $\mathcal{V}$, which invokes $\mathbf{V}$ to generate its next message. After the interaction, $\mathcal{V}$ invokes $\mathbf{V}$ to generate its queries to the committed polynomials. It sends these to $\mathcal{P}$, who replies with the desired evaluations along with an evaluation proof attesting to their correctness relative to the commitments. To obtain a zkSNARK, the Fiat–Shamir transform [22] is applied to this interactive argument.

4 Circuits for common operations

We now describe efficient circuits for operations that are commonly found in zkSNARK provers. These circuits will be used as building blocks for the PIOP prover circuits and for the PC scheme circuits in Section 4.3. When self-evident, we will omit proofs of claims about circuit depth.

4.1 Circuits for polynomial arithmetic

The fundamental objects in PIOP-based zkSNARKs are polynomials, and so it is natural that provers for such zkSNARKs make heavy use of polynomial arithmetic. We now describe efficient circuits for common operations on polynomials.

```
PolyAdd$(p_1, p_2) \to p_3$:
1. For $i \in \{0, \ldots, d\}$, set coefficient $p_{3,i} := \mathsf{Add}_{\mathbb{F}}(p_{1,i}, p_{2,i})$.
```

Claim 4.1. *The circuit depth of* PolyAdd *is* 0.

```
FFT(polynomial $p$, pub subgroup $H$) $\to \{p(\omega^i)\}_{i=0}^{|H|-1}$:
1. Compute the FFT via the standard algorithm [17], using
   only additions and multiplications by public values.
```

Claim 4.2. *The circuit depth of* FFT *is* 0.

```
IFFT(evaluations $\{p(\omega^i)\}_{i=0}^{|H|-1}$, pub subgroup $H$) $\to$ poly. $p$:
1. Compute the IFFT via the standard algorithm [17], using
   only additions and multiplications by public values.
```

Claim 4.3. *The circuit depth of* IFFT *is* 0.

```
PolyEval(priv $p$, pub point $z$) $\to$ priv $v$:
1. Compute $v := \sum_{i=0}^{d} p_i \cdot z^i$.
```

Claim 4.4. *The circuit depth of* PolyEval *is* 0.

```
PolyMul(priv $p_1$, priv $p_2$) $\to$ priv $p_3$:
1. Construct domain $H$ over which $p_1$, $p_2$ will be evaluated.
2. Compute $e_1 :=$ FFT$(p_1, H)$ and $e_2 :=$ FFT$(p_2, H)$.
3. Compute $e_3$ as the element-wise product of $e_1$ and $e_2$.
4. Interpolate these to obtain priv $p_3 :=$ IFFT$(e_3, H)$.
```

Claim 4.5. *The circuit depth of* $\mathsf{PolyMul}$ *is* 1.

Proof. FFT and IFFT have depth 0, and the $|H|$ multiplications are independent of each other. □

```
$\mathsf{PolyDiv}(p, \texttt{pub}\ \text{divisor}\ d) \to (\text{quotient}\ q, \text{remainder}\ r)$:
1. Obtain quotient and remainder via Euclidean division.
```

Claim 4.6. *The circuit depth of* $\mathsf{PolyDiv}$ *is* 0.

Proof. This follows because polynomial division is linear when the divisor d is public. Consider polynomials p_1 and p_2 such that $(q_1, r_1) := \mathsf{PolyDiv}(p_1, d)$, and $(q_2, r_2) := \mathsf{PolyDiv}(p_2, d)$. Consider $p_3 := \alpha p_1 + \beta p_2$ for arbitrary $\alpha, \beta \in \mathbb{F}$, and let $(q_3, r_3) := \mathsf{PolyDiv}(p_3, d)$. Then we have that

$$\begin{aligned} p_3 = q_3 \cdot d + r_3 &= \alpha(q_1 \cdot d + r_1) + \beta(q_2 \cdot d + r_2) \\ &= (\alpha q_1 + \beta q_2) \cdot d + (\alpha r_1 + \beta r_2) \end{aligned}$$

This means that $q_3 = \alpha q_1 + \beta q_2$ and $r_3 = \alpha r_1 + \beta r_2$, and hence $\mathsf{PolyDiv}(\alpha p_1 + \beta p_2, d) = \alpha \cdot \mathsf{PolyDiv}(p_1, d) + \beta \cdot \mathsf{PolyDiv}(p_2, d)$, which implies that $\mathsf{PolyDiv}$ is linear. □

4.2 Circuits for group arithmetic

```
$\mathsf{MSM}(\texttt{priv}\ \boldsymbol{c} \in \mathbb{F}^n, \texttt{pub}\ \boldsymbol{G} \in \mathbb{G}^n) \to \texttt{priv}\ R \in \mathbb{G}$:
1. Output result $\texttt{priv}\ R := \sum_{i=1}^{n} \mathsf{Mul}_{\mathbb{G}}(c_i, G_i)$.
```

Claim 4.7. *The circuit depth of* MSM *is* 0.

4.3 Circuits for PC schemes

A circuit C_{PC} for a PC scheme consists of subcircuits for two operations: committing to polynomials, and producing an evaluation proof for a committed polynomial at a given point. In this section we will describe efficient subcircuits for the popular pairing-based $\mathsf{PC}_{\mathsf{KZG}}$ polynomial commitment scheme [29, 14]. Later we will use versions of these circuits that allow committing to multiple polynomials with strict degree bounds, and opening these at multiple points. Circuits for these can be constructed from our simpler circuits via the transformations in [14] without increasing circuit depth.

4.3.1 Circuit for $\mathsf{PC}_{\mathsf{KZG}}$

```
$C_{\mathsf{KZG}}.\mathsf{Commit}(\texttt{pub}\ \mathsf{ck}, p; \bar{p}) \to C$:
1. Parse ck as $(\{\alpha^i G\}_{i=0}^{D}, \{\alpha^i \gamma G\}_{i=0}^{D})$.
2. $C := \mathsf{MSM}(p, \{\alpha^i G\}_{i=0}^{d}) + \mathsf{MSM}(\bar{p}, \{\alpha^i \gamma G\}_{i=0}^{d})$.
```

Claim 4.8. *The circuit depth of* $C_{\mathsf{KZG}}.\mathsf{Commit}$ *is* 0.

```
$C_{\mathsf{KZG}}.\mathsf{Open}(\texttt{pub}\ \mathsf{ck}, C, p, \texttt{pub}\ z; \bar{p}) \to \text{proof}\ \pi$:
1. Compute random evaluation $\texttt{priv}\ \bar{v} := \mathsf{PolyEval}(\bar{p}, z)$.
2. Compute witness and randomized witness polynomials:
   $(w, \_) := \mathsf{PolyDiv}(p, X - z)$; $(\bar{w}, \_) := \mathsf{PolyDiv}(\bar{p}, X - z)$.
3. Commit to $w$: $W := C_{\mathsf{KZG}}.\mathsf{Commit}(\mathsf{ck}, w; \bar{w})$.
4. Output $\pi := (W, \bar{v})$.
```

Claim 4.9. *The circuit depth of* $C_{\mathsf{KZG}}.\mathsf{Open}$ *is* 0.

5 Consistency checkers for PIOPs

Recall from Section 2.2 that the goal of a consistency checker is to efficiently check that the polynomial oracles produced by a (potentially malicious) PIOP prover match those that an honest prover would have produced when interacting with the same verifier messages. We first formally define consistency checkers in Section 5.1, and then, in Section 5.2, describe an efficient consistency checker for the PIOP of [14].

5.1 Definition

Ch is a **consistency checker** for a $\mathsf{PIOP} = (\mathsf{k}, \mathsf{s}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ for $\mathcal{R}$ if the following properties hold.

- **Completeness:** For all size bounds $N \in \mathbb{N}$, and for all efficient adversaries $\mathcal{A}$, the following probability equals one:

$$\Pr\left[\begin{array}{l|l} \begin{array}{l} \mathsf{st}_{\mathbf{Ch}} := (\mathbb{i}, \mathbb{x}, \mathbb{w}) \\ \text{for } j \in \{1, \dots, \mathsf{k}\}: \\ \quad (\mathsf{st}_{\mathbf{Ch}}, b_j) \leftarrow \mathbf{Ch}^{\boldsymbol{p}_j}(\mathsf{st}_{\mathbf{Ch}}) \\ \quad b_j = 1 \end{array} & \begin{array}{l} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(N) \\ \mathsf{st}_{\mathbf{P}} := (\mathbb{i}, \mathbb{x}, \mathbb{w}) \\ \mathsf{st}_{\mathbf{V}} := \mathbb{x} \text{ and } \boldsymbol{p}_{\mathbf{I}} := \mathbf{I}(\mathbb{i}) \\ \hline \text{for } j \in \{1, \dots, \mathsf{k}\}: \\ \quad (\mathsf{st}_{\mathbf{V}}, c_j) \leftarrow \mathbf{V}(\mathsf{st}_{\mathbf{V}}) \\ \quad (\mathsf{st}_{\mathbf{P}}, \boldsymbol{p}_j) := \mathbf{P}(\mathsf{st}_{\mathbf{P}}, c_j) \end{array} \end{array}\right]$$

- **Soundness:** For all size bounds $N \in \mathbb{N}$, and all adversaries $\widetilde{\mathbf{P}} = (\widetilde{\mathbf{P}}_1, \widetilde{\mathbf{P}}_2)$, the following probability is negligible:

$$\Pr\left[\begin{array}{l|l} \begin{array}{l} \mathsf{st}_{\mathbf{Ch}} := (\mathbb{i}, \mathbb{x}, \mathbb{w}) \\ \mathsf{st}_{\mathbf{P}} := (\mathbb{i}, \mathbb{x}, \mathbb{w}) \\ \mathbf{V}^{\widetilde{\boldsymbol{p}}, \boldsymbol{p}_{\mathbf{I}}}(\mathsf{st}_{\mathbf{V}}) = 1 \\ \text{for } j \in \{1, \dots, \mathsf{k}\}: \\ \quad (\mathsf{st}_{\mathbf{P}}, \boldsymbol{p}_j) := \mathbf{P}(\mathsf{st}_{\mathbf{P}}, c_j; r_j) \\ \quad (\mathsf{st}_{\mathbf{Ch}}, b_j) \leftarrow \mathbf{Ch}^{\widetilde{\boldsymbol{p}}_j}(\mathsf{st}_{\mathbf{Ch}}) \\ \quad (b_j = 1) \wedge (\boldsymbol{p}_j \neq \widetilde{\boldsymbol{p}}_j) \end{array} & \begin{array}{l} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R} \leftarrow \widetilde{\mathbf{P}}_1(N) \\ \mathsf{st}_{\mathbf{P}} := (\mathbb{i}, \mathbb{x}, \mathbb{w}) \\ \mathsf{st}_{\mathbf{V}} := \mathbb{x} \text{ and } \boldsymbol{p}_{\mathbf{I}} := \mathbf{I}(\mathbb{i}) \\ \hline \text{for } j \in \{1, \dots, \mathsf{k}\}: \\ \quad (\mathsf{st}_{\mathbf{V}}, c_j) \leftarrow \mathbf{V}(\mathsf{st}_{\mathbf{V}}) \\ \quad (\mathsf{st}_{\mathbf{P}}, \boldsymbol{p}_j) := \mathbf{P}(\mathsf{st}_{\mathbf{P}}, c_j) \\ \quad (\widetilde{\boldsymbol{p}}_j, r_j) := \widetilde{\mathbf{P}}_2(c_j) \end{array} \end{array}\right]$$

We will assume that **Ch** can be decomposed into two sub-algorithms $\mathbf{Ch}_{\mathbf{Q}}$ and $\mathbf{Ch}_{\mathbf{D}}$, where $\mathbf{Ch}_{\mathbf{Q}}$ outputs the queries it wishes to make to its oracles, along with the expected answers, and $\mathbf{Ch}_{\mathbf{D}}$ checks that the responses to these queries are consistent with the expected answers.

5.2 Consistency checker for MARLIN's PIOP

We now provide an overview of our consistency checker for the MARLIN PIOP of [14], and leave details to Appendix B.

Background on the MARLIN PIOP. Let us recall the structure of the MARLIN PIOP for the R1CS relation (described in Section 2), with an eye towards aspects relevant towards constructing a consistency checker. The PIOP uses as a building block a subPIOP for a holographic lincheck [14] that is used to prove a matrix-vector product. It batches three such subPIOPs (one for each R1CS matrix) together, resulting in a protocol with three rounds. In the first round, the prover sends polynomial oracles that are LDEs of the witness w, and of Az and Bz, and sends a randomized masking polynomial. In the second round, it sends two polynomials relating to the univariate sumcheck lemma [2]. The third round is witness-independent, and does not need to be checked for consistency.

Consistency checker. Our consistency checker, when given access to the first round polynomial oracles, proceeds by (1) sampling a random query point s, (2) locally evaluating the LDE of the witness w at s, and (3) querying the LDE of w (which is given as oracle) at s, and checking that the result matches the local evaluation. This checker is exceedingly simple and efficient. In particular, it only performs $O(|w|)$ field multiplications operations. (In our application to delegation, this corresponds to the computation performed by the delegator to enforce worker honesty.) These simple checks suffice because, by the soundness of the holographic lincheck subPIOP, if the adversary changes any of the other polynomials, the PIOP verifier's checks will fail. (In our application, this corresponds to the final delegated proof being invalid.)

6 Delegated SNARKs

A *delegation protocol* Π_{SNARK} for a zkSNARK $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ is a protocol between a computationally weak $\mathcal{D}$ and n powerful workers $[\mathcal{W}_i]_{i=1}^n$ which allows $\mathcal{D}$ to outsource the computations involved in $\mathcal{P}$ to the workers. In this section we formally define Π_{SNARK}, and construct a delegation protocol for the class of PIOP-based zkSNARKs.

6.1 Notation for MPC protocols

Protocol participants. We consider two kinds of participants: (a) an honest delegator $\mathcal{D}$ that wishes to outsource zkSNARK proof generation; and (b) n workers $[\mathcal{W}_i]_{i=1}^n$ that collectively perform the outsourced computation.

Protocol communication. Parties in our protocols communicate in one of two modes:

- *Isolated mode:* Honest workers communicate only with $\mathcal{D}$, and not with each other, over secure authenticated channels.
- *Collaborative mode:* Workers communicate with each other and with $\mathcal{D}$ over secure authenticated channels.

6.2 Definition

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a SNARK for an indexed NP relation $\mathcal{R}$. Then Π_{SNARK} is a *delegation protocol for* ARG with respect to the ideal functionality $\mathcal{F}_{\mathsf{SNARK}}$ (Fig. 2) if it is a protocol between a delegator $\mathcal{D}$ and n workers $[\mathcal{W}_i]_{i=1}^n$ such that, for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and for every efficient real-world adversary $\mathcal{A}$, there exists an efficient ideal-world simulator $\mathcal{S}_{\mathcal{A}}$, such that the output of the real execution is indistinguishable from the output of the ideal execution. That is, the following probability distributions are indistinguishable:

$$\{\textsc{Real}_{\Pi,\mathcal{A},P}((\mathsf{ipk}, \mathbb{x}, \mathbb{w}), \perp)\} \approx \{\textsc{Ideal}_{\mathcal{F},\mathcal{S},P}((\mathsf{ipk}, \mathbb{x}, \mathbb{w}), \perp)\}$$

Here $\mathcal{P} := \{\mathcal{D}\} \cup [\mathcal{W}_i]_{i=1}^n$ denotes the set of all involved parties, and ipk is the proving key specific to the index $\mathbb{i}$.

1. Receive $(\mathsf{ipk}, \mathbb{x}, \mathbb{w}, r)$ from $\mathcal{D}$
2. Compute $\pi \leftarrow \mathcal{P}^{\rho}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}; r)$.
3. Send $(\mathsf{ipk}, \mathbb{x}, \pi)$ to all workers (and hence to $\mathcal{S}$).
4. If $\mathcal{S}$ sends `reject`, output $\perp$; else, output π to $\mathcal{D}$.

Figure 2: Ideal functionality $\mathcal{F}_{\mathsf{SNARK}}$.

We require the delegator's computation in Π_{SNARK} to take time $O(|\mathbb{w}|)$. This precludes operations like MSMs and FFTs.

6.3 Delegating PIOP-based zkSNARKs

We construct a delegation protocol for SNARKs produced by the compiler of [14]. In more detail, we achieve the following theorem (we provide a proof in Appendix A).

Theorem 6.1 (delegating PIOP-based SNARKs)**.** *Let $\mathcal{R}$ be an indexed relation. Consider the following components:*

- $\mathsf{PIOP} = (\mathsf{k}, \mathsf{s}, \mathbf{I}, \mathbf{P}, \mathbf{V})$ *is a polynomial IOP for $\mathcal{R}$ (see Section 3.4), and C_{PIOP} is a circuit for $\mathbf{P}$;*
- $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Trim}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Check})$ *is a polynomial commitment scheme (see Section 3.3), and C_{PC} is a circuit for* $(\mathsf{PC.Commit}, \mathsf{PC.Open})$*; and*
- **Ch** *is a consistency checker for* PIOP.

Let $\mathsf{ARG} = (\mathcal{G}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ be a zkSNARK for $\mathcal{R}$ obtained by invoking the compiler of [14] on PIOP *and* PC*. Then C_{SNARK} (Fig. 3) is a circuit for $\mathcal{P}$, and Π_{SNARK} (Fig. 4), when instantiated with C_{SNARK}, is a delegation protocol for* ARG.

7 Implementation

We provide EOS, a Rust library that realizes our delegation protocols for zkSNARKs, and includes components of independent interest. EOS relies on, and contributes to, the state-of-the-art `arkworks` libraries [16]. We generalize the existing `arkworks` implementations of PIOP and PC schemes of [14] to support our new abstractions for secret-shared field elements and polynomials, thus avoiding duplicated effort.

7.1 Optimizations

Improved parallelization. For the zkSNARKs that we consider in this work, the majority of the computational overhead

$C_{\mathsf{SNARK}}.\mathsf{Init}^{\rho}(\texttt{pub}\ \mathrm{ipk}, \texttt{pub}\ \mathbb{x}, \texttt{priv}\ \mathbb{w}; \texttt{priv}\ r)$:

1. Obtain from ipk the index $\mathbb{i}$ and the verification key ivk.
2. Initialize random oracle state: $\mathsf{st}_{\rho} := \mathsf{RO}(\mathrm{ivk}, \mathbb{x})$.
3. Initialize PIOP prover state: $\texttt{priv}\ \mathsf{st}_{\mathbf{P}} := (\mathbb{i}, \mathbb{x}, \mathbb{w})$.
4. Initialize PIOP verifier state: $\texttt{pub}\ \mathsf{st}_{\mathbf{V}} := \mathbb{x}$.
5. Set round number $j := 1$.
6. Set PIOP verifier randomness $\mu_j := \bot$.
7. Setup state: $\mathsf{st}_{\mathcal{P}} := (\mathrm{ipk}, \mathsf{st}_{\rho}, \mathsf{st}_{\mathbf{P}}, \mathsf{st}_{\mathbf{V}}, j, \mu_j)$.

$C_{\mathsf{SNARK}}.\mathsf{Round}^{\rho}$:

1. Parse $\mathsf{st}_{\mathcal{P}} = (\mathrm{ipk}, \mathsf{st}_{\rho}, \mathsf{st}_{\mathbf{P}}, \mathsf{st}_{\mathbf{V}}, j, \mu_j)$.
2. Run PIOP verifier: $(\mathsf{st}_{\mathbf{V}}, \mathsf{msg}_j) := \mathbf{V}(\mathsf{st}_{\mathbf{V}}, \mu_j)$.
3. Set $(\mathsf{st}_{\mathbf{P}}, \boldsymbol{p}_j) := \mathbf{P}(\mathsf{st}_{\mathbf{P}}, \mathsf{msg}_j, r_j)$.
4. Compute $\texttt{pub}\ \boldsymbol{C}_j := \mathsf{Reveal}(C_{\mathsf{PC}}.\mathsf{Commit}(\mathsf{ipk.ck}, \boldsymbol{p}_j; \bar{\boldsymbol{p}}_j))$.
5. Compute $\texttt{pub}\ (\mathsf{st}_{\rho}, \mu_{j+1}) := \mathsf{RO}(\mathsf{st}_{\rho}, \boldsymbol{C}_j)$.
6. Set $\mathsf{st}_{\mathcal{P}} := (\mathsf{st}_{\rho}, \mathsf{st}_{\mathbf{P}}, \mathsf{st}_{\mathbf{V}}, \mathrm{ipk}, j+1, \mu_{j+1})$.
7. If the round $j+1$ is witness-independent, $\mathsf{Reveal}(\mathsf{st}_{\mathcal{P}})$.

$C_{\mathsf{SNARK}}.\mathsf{Finalize}^{\rho} \to \texttt{pub}\ \pi$:

1. Denote by $\boldsymbol{p}$ the set of all index and prover polynomials, and by $\boldsymbol{C}$ and $\bar{\boldsymbol{p}}$ the corresponding commitments and random polynomials.
2. Compute the query set $Q := \mathbf{Q}_{\mathbf{V}}(\mathsf{st}_{\mathbf{V}})$.
3. Evaluate $\boldsymbol{p}$ at Q via $\mathsf{PolyEval}$ to obtain evaluations $\boldsymbol{v}$, and send these to $\mathcal{D}$.
4. Compute evaluation proof $\pi_{\mathsf{PC}} := C_{\mathsf{PC}}.\mathsf{Open}^{\rho}(\mathsf{ck}, \boldsymbol{C}, \boldsymbol{p}, Q; \bar{\boldsymbol{p}})$.
5. Assemble and reveal to $\mathcal{D}$ the final proof $\pi := (\boldsymbol{C}, \boldsymbol{v}, \pi_{\mathsf{PC}})$.

Figure 3: Circuit C_{SNARK} for zkSNARK prover.

$\mathsf{Preprocess}(C)$:
If In isolated mode, output nothing. Otherwise, $\mathcal{D}$ samples multiplication triples for the workers as follows.

1. For each $k \in [|C|_{\mathsf{Mul}_{\mathbb{F}}}]$:
    (a) For each $i \in [n-1]$, $\mathcal{D}$ samples seed $s_i \leftarrow \{0,1\}^{\lambda}$ and computes $\mathcal{W}_i$'s share of the k-th triple: $([\![\alpha_k]\!]_i, [\![\beta_k]\!]_i, [\![\gamma_k]\!]_i) := \mathsf{PRG}(s_i)$.
    (b) $\mathcal{D}$ computes $([\![\alpha_k]\!]_n, [\![\beta_k]\!]_n) := \mathsf{PRG}(s_n)$.
2. $\mathcal{D}$ sets $[\![\gamma_k]\!]_n := (\sum_{i=1}^{n} [\![\alpha_k]\!]_i \cdot \sum_{i=1}^{n} [\![\beta_k]\!]_i) - \sum_{i=1}^{n-1} [\![\gamma_k]\!]_i$, and sends to worker $\mathcal{W}_n$ $(s_n, [[\![\gamma_k]\!]_n]_{k=1}^{m})$, and sends to the other workers s_i.
3. For each $i \in [n-1]$, $\mathcal{W}_i$ computes $([\![[\alpha_k]_{k=1}^{m}]\!]_i, [\![[\beta_k]_{k=1}^{m}]\!]_i, [\![[\gamma_k]_{k=1}^{m}]\!]_i) := \mathsf{PRG}(s_i)$.
4. $\mathcal{W}_n$ additionally computes $([\![[\alpha_k]_{k=1}^{m}]\!]_n, [\![[\beta_k]_{k=1}^{m}]\!]_n) := \mathsf{PRG}(s_n)$.

Online:

1. The delegator $\mathcal{D}$ has input $(\texttt{pub}\ \mathrm{ipk}, \texttt{pub}\ \mathbb{x}, \texttt{priv}\ \mathbb{w}, \texttt{priv}\ r)$. It sends to the workers ipk and $\mathbb{x}$ in the clear, and secret shares $\mathbb{w}$ and r.
2. $\mathcal{D}$ initializes the PIOP checker state: $\mathsf{st}_{\mathbf{Ch}} := (\mathbb{i}, \mathbb{x}, \mathbb{w})$.
3. Initialize the SNARK prover: $\mathsf{ExecCircuit}(C_{\mathsf{SNARK}}.\mathsf{Init}^{\rho})$.
4. For each round $j \in \{1, \ldots, \mathsf{k}\}$:
    (a) The parties invoke $\mathsf{ExecCircuit}(C_{\mathsf{SNARK}}.\mathsf{Round}^{\rho})$ to execute the j-th round of the SNARK prover, and $\mathcal{D}$ receives $(\mathsf{st}_{\rho}, \mathsf{st}_{\mathbf{V}}, \boldsymbol{C}_j)$.
    (b) $\mathcal{D}$ invokes $\mathbf{Ch_Q}$ to obtain queries to $\boldsymbol{p}_j$ (the polynomials in $\boldsymbol{C}_j$), and the expected answers: $(\mathsf{st}_{\mathbf{Ch}}, Q, \boldsymbol{v}_j) \leftarrow \mathbf{Ch_Q}(\mathsf{st}_{\mathbf{Ch}})$.
    (c) The workers receive Q from $\mathcal{D}$, and invoke $\mathsf{ExecCircuit}(C_{\mathsf{PC}}.\mathsf{Open})$ to reveal to $\mathcal{D}$ the evaluation proof π_j.
    (d) $\mathcal{D}$ verifies π_j with respect to $(\boldsymbol{C}_j, Q, \boldsymbol{v}_j)$, and the workers to proceed if it is valid. (This corresponds to the check of $\mathbf{Ch_D}$.)
5. $\mathcal{D}$ obtains the final proof $\pi \leftarrow \mathsf{ExecCircuit}(C_{\mathsf{SNARK}}.\mathsf{Finalize}^{\rho})$.

$\mathsf{ExecCircuit}(\text{circuit } C)$: Process each gate of C as follows:

- $\mathsf{Add}_{\mathbb{F}}(w_a, w_b) \to w_c$:
  - If w_a and w_b are both public, then the workers just set $\texttt{pub}\ w_c := w_a + w_b$.
  - If w_a and w_b are both private, then each worker $\mathcal{W}_i$ sets $[\![w_c]\!]_i := [\![w_a]\!]_i + [\![w_b]\!]_i$.
  - Else, assuming wlog that w_a is public, $\mathcal{W}_1$ sets $[\![w_c]\!]_1 := w_a + [\![w_b]\!]_1$, while every other worker $\mathcal{W}_i$ sets $[\![w_c]\!]_i := [\![w_b]\!]_i$.
- $\mathsf{Mul}_{\mathbb{F}}(w_a, w_b) \to w_c$:
  - If w_a and w_b are both public, then the workers just set $\texttt{pub}\ w_c := w_a \cdot w_b$.
  - If w_a and w_b are both private, then:
    * In isolated mode, the workers send shares of w_a and w_b to $\mathcal{D}$, who reconstructs w_a and w_b and reshares $w_c := w_a w_b$.
    * In collaborative mode, the workers use the triples generated in the preprocessing step to evaluate the multiplication.
  - Else, without loss of generality assume w_a is public and w_b is private. Then each worker $\mathcal{W}_i$ sets $[\![w_c]\!]_i := w_a \cdot [\![w_b]\!]_i$.
- $\mathsf{Add}_{\mathbb{G}}(w_a, w_b) \to w_c$: Proceed as in $\mathsf{Add}_{\mathbb{F}}$, but use group addition instead.
- $\mathsf{Mul}_{\mathbb{G}}(w_a, w_b) \to w_c$: Proceed as in $\mathsf{Mul}_{\mathbb{F}}$, but use scalar multiplication in $\mathbb{G}$ instead. (This is linear if w_a or w_b are $\texttt{pub}$).
- $\mathsf{Reveal}(w_a) \to w_b$:
  - If w_a is public, then the workers just set $\texttt{pub}\ w_b := w_a$.
  - Else, in isolated mode, each worker $\mathcal{W}_i$ sends $[\![w_a]\!]_i$ to $\mathcal{D}$, who combines these shares and returns $\texttt{pub}\ w_b := \sum_{i=1}^{n} [\![w_a]\!]_i$.
  - Else, in collaborative mode, each worker $\mathcal{W}_i$ broadcasts $[\![w_a]\!]_i$, and all workers reconstruct $\texttt{pub}\ w_b := \sum_{i=1}^{n} [\![w_a]\!]_i$.
- $\mathsf{RO}(w_{a_1}, \ldots, w_{a_k}) \to w_b$: Set $\texttt{pub}\ w_b := \rho(\mathsf{Reveal}(w_{a_1}), \ldots, \mathsf{Reveal}(w_{a_k}))$.

Figure 4: Our delegation protocol for PIOP-based zkSNARKs

of the prover arises from finite field FFTs and elliptic-curve multi-scalar multiplications (MSMs). This is reflected in our delegation protocols as well: the majority of the worker-time is spent in these two tasks. Thus, achieving an efficient implementation of these tasks is critical for our protocol to be beneficial. One strategy is to leverage parallelism. However, the libraries that we use in our implementation[3] provide parallel implementations for FFTs and MSMs that achieve diminishing returns as the number of threads increases; after a threshold number t of threads, the latency of the FFT and MSM algorithms does not decrease with the number of threads.

To overcome this issue, we leverage the insight that zkSNARK prover perform many *independent* FFTs and MSMs that can be computed concurrently and in parallel. In more detail, we modified the underlying libraries to allocate to each FFT or MSM only t threads, and to then run multiple FFTs/MSMs in parallel. Hence, if a machine has mt threads for some m, then our implementation performs m FFTs/MSMs in parallel, each running with t threads.

In our experiments (Section 8), these optimizations reduce the time required to commit to a batch of polynomials by up to $3\times$ and the time required to run the PIOP prover by up to $4\times$. These improvements are of independent interest, and we plan to upstream them to the relevant libraries.

Reduced delegator communication when secret sharing vectors. Having the delegator naively secret share a vector v to all workers requires communicating $n \cdot |v|$ field elements. We instead adopt a technique from the semi-honest MPC literature [21, 39] and have the delegator send a full share of size $|v|$ only to a single worker while the rest get PRG seeds. This reduces communication to $|v| + O_\lambda(n)$ field elements.

Lower memory usage for PIOP delegation. Our delegator-based triple-generation protocol requires the delegator to process many field elements (proportional to the number M of constraints in the R1CS instance being delegated). A straightforward implementation would have the delegator keep all these elements in memory, resulting in large memory usage for the delegator. Instead, we observe that the delegator only requires *streaming* access to these elements, and so can process them in batches. We implement this idea and enable the delegator to only use a constant amount of additional memory (beyond that required to produce the initial witness).

Reducing latency for secret multiplications. For an R1CS instance with $M = 2^k$ constraints, the PIOP prover $\mathbf{P}$ of [14, Appendix E] computes the following secret multiplication:

$$\hat{z}_A \cdot \hat{z}_B := (\bar{z}_A + r_A v_H) \cdot (\bar{z}_B + r_B v_H)$$

Here H is a multiplicative subgroup of size M, $z = (x, w) \in \mathbb{F}^M$, $\bar{z}_A$ and $\bar{z}_B$ are the LDEs of $z_A := Az$ and $z_B := Bz$, respectively, over H, v_H is the vanishing polynomial of H (of degree $|H|$), and r_A and r_B are randomly sampled elements in $\mathbb{F}$. As both $\hat{z}_A$ and $\hat{z}_B$ have degree $|H|$, their product has degree $2|H|$ and is specified by $2|H|+1$ evaluations. Multiplying them thus requires FFTs and IFFTs of size $4|H|$ (as we must round to the next power of two), which is expensive for large H. We avoid this by rewriting the multiplication as:

$$\hat{z}_A \cdot \hat{z}_B = \bar{z}_A \bar{z}_B + r_A v_H \bar{z}_B + r_B v_H \bar{z}_A + r_A r_B v_H^2 \quad .$$

Since v_H^2 can be computed efficiently in the clear without FFTs, evaluating this expression now only requires FFTs of size $2|H|$. This enables the following optimizations:

- *Isolated mode:* Workers send their shares of $\mathrm{FFT}(\bar{z}_A)$ and $\mathrm{FFT}(\bar{z}_B)$ to the delegator. The delegator samples r_A and r_B locally, reconstructs $\mathrm{FFT}(\bar{z}_A)$ and $\mathrm{FFT}(\bar{z}_B)$, and then sends shares of $\mathrm{FFT}(\bar{z}_A) \circ \mathrm{FFT}(\bar{z}_B)$, r_A, r_B, and $r_A r_B$ to the workers, who use these shares to compute shares of the final product.
- *Collaborative mode:* The workers locally sample shares of r_A and r_B, use multiplication triples to compute shares of $\{\mathrm{FFT}(\bar{z}_A) \circ \mathrm{FFT}(\bar{z}_B),\ r_A z_B,\ r_B z_A,\ r_A r_B\}$, and then use these to locally compute shares of the final product.

Overall, these improvements reduce worker computation (the FFT is smaller and the additional multiplications can be performed in parallel), and worker communication (only $2|H|$ secret multiplications need to be performed, instead of $4|H|$).

Reducing communication for scalar-vector products. In the foregoing optimization, in collaborative mode, workers have to compute the scalar-vector products $r_A z_B$ and $r_B z_A$ using multiplication triples. Each such product requires $|H|$ triples $\{(a_i, b_i, a_i \cdot b_i) \in \mathbb{F}^3\}_{i \in [|H|]}$. However, all of $a_1, \dots a_{|H|}$ are used to open the same value (for example, r_A). Instead of generating and then wasting $|H| - 1$ of these values, we reduce communication cost in both the preprocessing and online phases by using triples of the form $\{a, b_i, a \cdot b_i\}_{i \in [|H|]} \in \mathbb{F}^3$. When delegating with two workers, this optimization reduces per-worker communication from $8|H|$ to $6|H|$ field elements.

8 Evaluation

We evaluate Eos by using it to delegate the prover of the zkSNARK of [14], with the aim of answering the following questions:

Q1: Do our delegation protocols enable proving R1CS instances *larger* than possible with local proving?

Q2: For locally-provable R1CS instances, what is the overhead (if any) of delegation, in terms of end-to-end time and communication costs, and how do these costs scale with instance size? We answer these questions via a comprehensive evaluation. We answer **Q1** in Section 8.2 by comparing the instance sizes provable via delegation with the corresponding sizes provable locally, within the same memory and time budgets for the delegator. We answer **Q2** in Section 8.3 by comparing the cost of delegated proving to that of proving in both fully-private and non-private baselines. We also evaluate how these costs scale with R1CS instance size. Additionally, in Section 8.4, we provide a detailed comparison with [36].

[3] `https://arkworks.rs/marlin` for PIOPs, and `https://arkworks.rs/poly-commit` for PC schemes

8.1 Experimental setup

Delegator setups. All experiments were evaluated with three different delegator setups.

- Setup LAPTOPHB: The delegator is an AWS `r4.xlarge` instance with 32 GB of RAM and 4 cores of an Intel Xeon E5-2686 CPU at 2.3 GHz. It is located in `us-west-2`, and has a network throughput of 3 Gbps. This setup emulates a midgrade laptop with a strong network connection.
- Setup LAPTOPLB: The delegator is the same as in LAPTOPHB, but with a throttled connection of 350 Mbps down and 13 Mbps up. This setup emulates a midgrade laptop with an average network connection.
- Setup MOBILE: The delegator is a Google Pixel 4a smartphone with 6 GB of RAM and a Qualcomm Snapdragon 730G processor (six cores at 1.8 GHz and two cores at 2.2 GHz). It is located in Northern California and uses a Wi-Fi connection with 350 Mbps download speed and 13 Mbps upload speed. This setup represents a commodity smartphone with an average Wi-Fi connection.

Workers. In each of the foregoing setups, the delegators interacted with two worker machines running on AWS `c5.24xlarge` instances with 192 GB of RAM and 96-core Intel Xeon Platinum 8000 series CPUs at 3.6 GHz. We placed the worker machines in different regions to simulate different trust domains: worker $\mathcal{W}_1$ in the `us-west-1` (Northern California) region, and worker $\mathcal{W}_2$ in the `us-east-1` (Northern Virginia) region. Each worker cost \$0.14/sec.

In all setups, the RTT between (a) $\mathcal{D}$ and $\mathcal{W}_1$ was 21 ms, (b) $\mathcal{D}$ and $\mathcal{W}_2$ was 72 ms, and (c) $\mathcal{W}_1$ and $\mathcal{W}_2$ was 62 ms.

8.2 Can delegation prove large instances?

In Table 1 we evaluate the largest R1CS instances that can be proven locally and via delegation with EOS, within fixed time budgets and memory budgets. Our evaluation shows that in every case, EOS enables proving larger instances. In particular, in all setups, delegation in both isolated and collaborative mode enables proving up to 256× larger instances within the same memory budget of 3 GB; in MOBILE, collaborative mode delegation can prove 32× larger instances within the same time budget of 100 s; finally, in LAPTOPHB, the largest delegatable instance size is the same in isolated and collaborative modes, while in LAPTOPLB and MOBILE, collaborative mode supports larger sizes. This is because in the latter setups, delegator-worker communication is the bottleneck.

Remark 8.1 (witness streaming)**.** Given streaming access to the witness, the delegator's algorithm in our protocols can be executed in a streaming manner, requiring only constant space. Hence, if the witness can itself be generated in a streaming manner, then our protocols enable delegation for instances of *any size* (assuming worker instances with sufficient memory).

setup	prover type	time budget		memory budget	
		size	increase	size	increase
LAPTOPHB	local	2^{18}	—	2^{17}	—
	isolated	2^{21}	8×	2^{25}	256×
	collab.	2^{21}	8×	2^{25}	256×
LAPTOPLB	local	2^{18}	—	2^{17}	—
	isolated	2^{19}	2×	2^{25}	256×
	collab.	2^{21}	8×	2^{25}	256×
MOBILE	local	2^{16}	—	2^{17}	—
	isolated	2^{19}	8×	2^{25}	256×
	collab.	2^{21}	32×	2^{25}	256×

Table 1: Largest instance sizes provable when the delegator has (a) a time budget of 100 s and maximum available memory budget; and (b) a memory budget of 3 GB and no time budget.

8.3 What is the overhead of delegation?

We have established that, for fixed time and memory budgets, our delegation protocols enable proving larger instance sizes than is possible when proving locally. Now we examine the latency and communication costs incurred during delegation.

Baselines. We compare these costs for delegation protocols against the same costs for the following baselines.
- DEL: the delegator locally generates the zkSNARK.
- WORKER: a single worker locally generates the zkSNARK.

These baselines represent two extremes in the trade-off between privacy and proving cost: proving on the delegator is slow but hides the witness, while proving on a worker machine is fast but completely leaks the witness. Our delegation protocol explores the space between these extremes by guaranteeing privacy as long as at least one worker is honest.

8.3.1 End-to-end time for proof generation

In Fig. 5, we compare the end-to-end time (henceforth latency) required to produce a proof in EOS with the latency of Baselines DEL and WORKER. These numbers do *not* include preprocessing time.[4] We find that:

- In LAPTOPHB, latency when delegating in isolated mode is lower than in Baseline DEL by 5.5×–8.5×, while latency in collaborative mode is lower by 5.5×–9×. The speedups are similar as the high bandwidth means that delegator-worker communication is not a bottleneck. Furthermore, the overhead with respect to the *optimal* non-private Baseline WORKER is at most 1.1×–1.9× in both modes, which demonstrates that, given sufficient bandwidth, EOS has low computational overhead.
- In LAPTOPLB, the restricted upload bandwidth hinders latency compared to LAPTOPHB. This is expected because the delegator must secret-share the witness with the workers, and must help compute secret multiplications. While both costs are linear in the instance size, in isolated mode, the

communication for the secret multiplication occurs during proof generation itself, but in collaborative mode it occurs in the preprocessing phase (see Fig. 8), resulting in a lower online delegator cost. Concretely, Fig. 5 shows that isolated mode achieves a 1.7× speedup over Baseline DEL while collaborative mode achieves a 5.7× speedup.

- In MOBILE, as in LAPTOPLB, the restricted bandwidth worsens the latency. However, the smartphone delegator of MOBILE is much less powerful than the laptop delegator of LAPTOPLB, which is illustrated by the larger gap between Baselines DEL and WORKER. This means that isolated mode still provides a 7.6×–8.5× speedup, while collaborative mode achieves a 22×–26× speedup.

In summary, EOS provides a speedup over Baseline DEL in all setups. The most significant improvements occur when the delegator has sufficient bandwidth or is resource constrained. If preprocessing is possible, collaborative mode provides superior performance to isolated mode.

8.3.2 Delegator online time

In Fig. 6, we show the time for which the delegator is actively participating in the proof generation. This metric, which we call "online time", includes computation time in local proving (Baseline DEL) and, in delegation, time spent for computation and communication during the *online phase*.[4]

In general, we see that the delegator online time is lower in collaborative mode than in isolated mode, as in the latter the delegator has to additionally assist with computing secret multiplications and verifier challenges. However, in both cases the online time is significantly lower than the times in the baselines. Concretely, in Fig. 6 we see the following speedups:

- In LAPTOPHB, delegation in isolated mode reduces delegator online time by at least 19×, while delegation in collaborative mode reduces online time by at least 592×.
- In LAPTOPLB, delegation in isolated mode reduces delegator online time by at least 2×, while delegation in collaborative mode reduces online time by at least 12×.
- In MOBILE, delegation in isolated mode reduces delegator online time by at least 10×, while delegation in collaborative mode reduces online time by at least 96×.

Notice that in all cases, the gap between isolated and collaborative mode reduces as the instance size increases. This is because at lower instance sizes, the overhead of the delegator's secret multiplications is larger than the overhead of delegator-worker communication. As instance size increases, the communication overhead starts dominating.

[4]We exclude preprocessing time from these figures as it is independent of the witness, and the preprocessed material can be generated and stored much before it is needed. See Section 8.3.3 for details.

8.3.3 Cost of preprocessing

We evaluate preprocessing cost in Fig. 7. We find that at small instance sizes, the preprocessing cost is similar in all setups, but that as instance size increases, the low bandwidth in the LAPTOPLB and MOBILE setups becomes a bottleneck, leading to much higher times than in LAPTOPHB. Furthermore, we see that the preprocessing time converges in the low-bandwidth setups, which indicates that the communication cost is the dominating factor. Note that because preprocessed material can be stored on the server until it is needed, it can be generated when the delegator's device is connected to a high-bandwidth network, thus lowering preprocessing cost.

8.3.4 Communication overhead of delegation

In Fig. 8, we report the cost of delegator-worker communication and find that it is the same for each setup. As expected, the cost grows linearly with the instance size, and isolated mode incurs a higher cost, even when accounting for preprocessing.

8.4 Comparison with the protocols of OB22

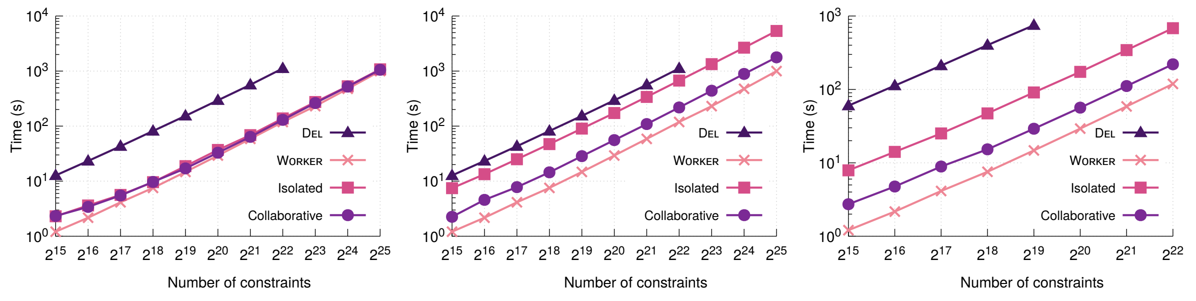
As mentioned in Section 1.2, the distributed proving protocols of OB22 [36] are not specialized for delegation, and do not perform as well as EOS does in this setting. To estimate the actual cost of these differences, we evaluated OB22 on LAPTOPHB using their open-source code.[5] We chose to evaluate in LAPTOPHB as OB22 compares best in this setting: in the other setups, where bandwidth is a bottleneck, OB22 does progressively worse because of its higher communication cost. The results of this evaluation are reported in Fig. 9, where we compare the latency of OB22 to that of our protocol, and find that EOS is 6–8× faster. Additionally, our protocols require 5× less communication between workers and 3× less communication between the delegator and worker machines.

9 Conclusion

In this paper, we define, construct, and implement delegation protocols for zkSNARKs. Our delegation protocols enable speedups in proving time in a variety of bandwidth and computation settings, and achieve malicious security without relying on traditional tools used in malicious MPC.

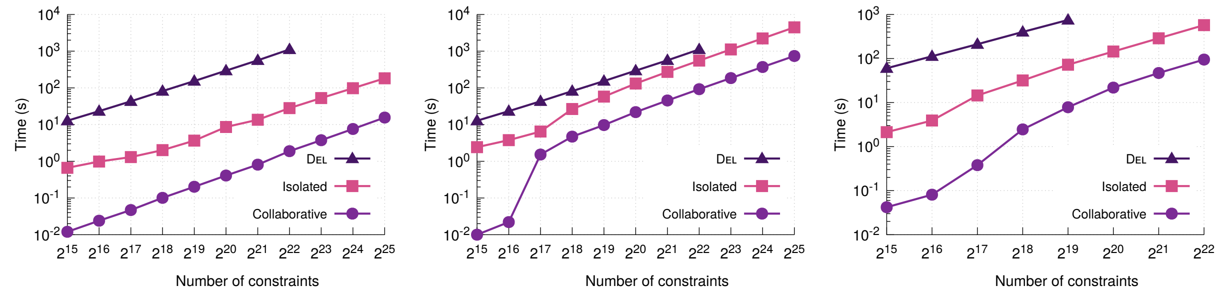
Acknowledgements. We thank the anonymous reviewers as well as our shepherd for their detailed feedback that simplified and improved the presentation of this paper. This work was supported in part by the NSF CISE Expeditions CCF-1730628, NSF Career 1943347, and gifts/awards from the Ethereum Foundation, Sloan Foundation, Bakar Program, Alibaba, Amazon Web Services, Ant Group, Capital One, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk, and VMware.

[5]https://github.com/alex-ozdemir/collaborative-zksnark/



(a) LAPTOPHB **(b)** LAPTOPLB **(c)** MOBILE

Figure 5: Latency of proof generation when delegating the zkSNARK of [14] in the three setups.



(a) LAPTOPHB **(b)** LAPTOPLB **(c)** MOBILE

Figure 6: Delegator online time for the zkSNARK of [14] in the three setups.

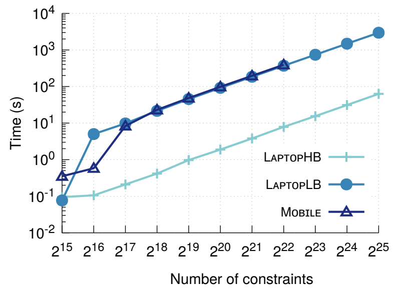


Figure 7: Time required for preprocessing in collaborative mode.

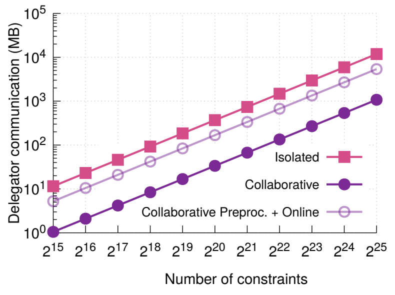


Figure 8: Communication between the delegator and the workers.

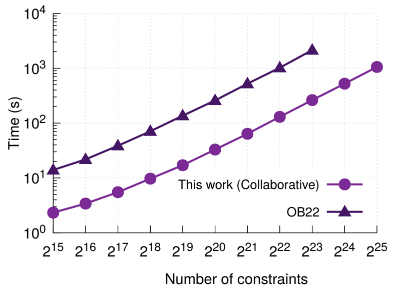


Figure 9: Our work vs. that of [36] in Setup LAPTOPHB.

References

[1] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: S&P ’14.

[2] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward. “Aurora: Transparent Succinct Arguments for R1CS”. In: EUROCRYPT ’19.

[3] E. Ben-Sasson, A. Chiesa, and N. Spooner. “Interactive Oracle Proofs”. In: TCC ’16-B.

[4] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. “Scalable Zero Knowledge via Cycles of Elliptic Curves”. In: CRYPTO ’14.

[5] J.-P. Berrut and L. N. Trefethen. “Barycentric Lagrange Interpolation”. In: *SIAM Review* (2004).

[6] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. “Succinct Non-Interactive Arguments via Linear Interactive Proofs”. In: TCC ’13.

[7] A. R. Block and C. Garman. “Honest Majority Multi-Prover Interactive Arguments”. IACR ePrint Report 2022/557.

[8] J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. “Coda: Decentralized Cryptocurrency at Scale”. IACR ePrint Report 2020/352.

[9] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. “ZEXE: Enabling Decentralized Private Computation”. In: S&P ’20.

[10] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. “Bulletproofs: Short Proofs for Confidential Transactions and More”. In: S&P ’18.

[11] B. Bünz, B. Fisch, and A. Szepieniec. “Transparent SNARKs from DARK Compilers”. In: EUROCRYPT ’20.

[12] V. Buterin. “A Quick Barycentric Evaluation Tutorial”. URL: https://hackmd.io/%5C@vbuterin/barycentric_evaluation.

[13] M. Campanelli, A. Faonio, D. Fiore, A. Querol, and H. Rodriguez. “Lunar: a Toolbox for More Efficient Universal and Updatable zkSNARKs and Commit-and-Prove Extensions”. In: ASIACRYPT ’21.

[14] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. “Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS”. In: EUROCRYPT ’20.

[15] A. Chiesa, D. Ojha, and N. Spooner. “Fractal: Post-Quantum and Transparent Recursive Proofs from Holography”. In: EUROCRYPT ’20.

[16] arkworks contributors. *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[17] J. W. Cooley and J. W. Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Math. Comp.* (1965).

[18] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. “Multiparty Computation from Somewhat Homomorphic Encryption”. In: CRYPTO ’12.

[19] P. Dayama, A. Patra, P. Paul, N. Singh, and D. Vinayagamurthy. “How to prove any NP statement jointly? Efficient Distributed-prover Zero-Knowledge Protocols”. In: PETS ’22.

[20] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. “Cinderella: Turning Shabby X.509 Certificates into Elegant Anonymous Credentials with the Magic of Verifiable Computation”. In: S&P ’16.

[21] D. Demmler, T. Schneider, and M. Zohner. “Ad-Hoc Secure Two-Party Computation on Mobile Devices using Hardware Tokens”. In: USENIX Security ’14.

[22] A. Fiat and A. Shamir. “How to prove yourself: practical solutions to identification and signature problems”. In: CRYPTO ’86.

[23] A. Gabizon, K. Gurkan, P. Jovanovic, G. Konstantopoulos, A. Oines, M. Olszewski, M. Straka, E. Tromer, and P. Vesely. “Plumo: Towards Scalable Interoperable Blockchains Using UltraLight Validation Systems”. In: ZKProof ’20.

[24] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. “PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge”. IACR ePrint Report 2019/953.

[25] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. “Quadratic Span Programs and Succinct NIZKs without PCPs”. In: EUROCRYPT ’13.

[26] J. Groth. “On the Size of Pairing-Based Non-interactive Arguments”. In: EUROCRYPT ’16.

[27] J. Groth. “Short Pairing-Based Non-interactive Zero-Knowledge Arguments”. In: ASIACRYPT ’10.

[28] S. Kanjalkar, Y. Zhang, S. Gandlur, and A. Miller. “Publicly Auditable MPC-as-a-Service with succinct verification and universal setup”. In: EuroS&P ’21 Workshops.

[29] A. Kate, G. M. Zaverucha, and I. Goldberg. “Constant-Size Commitments to Polynomials and Their Applications”. In: ASIACRYPT ’10.

[30] A. Kattis and J. Bonneau. “Proof of Necessary Work: Succinct State Verification with Fairness Guarantees”. IACR ePrint Report 2020/190.

[31] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”. In: S&P ’16.

[32] J. Lee, K. Nikitin, and S. T. V. Setty. “Replicated state machines without replicated execution”. In: S&P ’20.

[33] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. “Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updateable Structured Reference Strings”. In: CCS ’19.

[34] A. Naveh and E. Tromer. “PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations”. In: S&P ’16.

[35] O(1) Labs. “Mina Cryptocurrency”. minaprotocol.org.

[36] A. Ozdemir and D. Boneh. “Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets”. In: USENIX Security ’22.

[37] B. Parno, C. Gentry, J. Howell, and M. Raykova. “Pinocchio: Nearly Practical Verifiable Computation”. In: S&P ’13.

[38] A. Pruden. “zkCloud: Decentralized Private Computing”. `https://aleo.org/post/zkcloud`.

[39] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. “Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications”. In: AsiaCCS ’18.

[40] M. Rosenberg, J. White, C. Garman, and I. Miers. “`zk-creds`: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure”. IACR ePrint Report 2022/878.

[41] B. Schoenmakers, M. Veeningen, and N. de Vreede. “Trinocchio: Privacy-Preserving Outsourcing by Distributed Verifiable Computation”. In: ACNS ’16.

[42] S. Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: CRYPTO ’20.

[43] S. Setty, S. Angel, T. Gupta, and J. Lee. “Proving the correct execution of concurrent services in zero-knowledge”. In: OSDI ’18.

[44] A. Shamir. “How to Share a Secret”. In: *Commun. ACM* (1979).

[45] R. S. Wahby, M. Howald, S. J. Garg, A. Shelat, and M. Walfish. “Verifiable ASICs”. In: S&P ’16.

[46] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. “DIZK: A Distributed Zero Knowledge Proof System”. In: USENIX Security ’18.

[47] “Zcash”. `https://z.cash/`.

A Proof of security for Theorem 6.1

```
Preprocess($C$):
1. If we are in isolated mode, output nothing.
2. If we are in collaborative mode, then generate multiplication triples
   as in $\Pi_{\mathsf{SNARK}}$.
```

```
Online:
1. Receive the index proving key ipk, the NP instance $\mathbb{x}$, and the proof
   $\pi = (\boldsymbol{C}, \boldsymbol{v}, \pi_{\mathsf{PC}})$ from $\mathcal{F}_{\mathsf{SNARK}}$.
2. Set the simulated witness $\widetilde{\mathbb{w}} := 0^{|\mathbb{w}|}$, and sample secret shares for
   it: $[[\![\widetilde{\mathbb{w}}]\!]_i]_{i=1}^n \leftarrow \mathsf{SS.Share}(\mathbb{F}, \widetilde{\mathbb{w}}, n)$.
3. Set the simulated randomness $r := 0 \in \mathbb{F}$, then sample secret shares
   for it: $[[\![r]\!]_i]_{i=1}^n \leftarrow \mathsf{SS.Share}(\mathbb{F}, r, n)$.
4. Set the initial Ch state to $\mathsf{st}_{\mathbf{Ch}} := (\mathbb{i}, \mathbb{x}, \widetilde{\mathbb{w}})$.
5. Send to the adversarial parties their shares of $\widetilde{\mathbb{w}}$ and randomness $r$,
   along with the index proving key ipk, and the NP instance $\mathbb{x}$.
6. Simulate the protocol execution for all the parties using their secret
   shares of $\widetilde{\mathbb{w}}$ and $r$.
7. Handle the execution as follows:
   • To evaluate linear gates, ($\mathsf{Add}_{\mathbb{F}}$, $\mathsf{Add}_{\mathbb{G}}$, $\mathsf{Mul}_{\mathbb{G}}$, and $\mathsf{Mul}_{\mathbb{F}}$ when
     one of the inputs is public), there is no interaction between parties,
     and so computation proceeds locally according to $\Pi_{\mathsf{SNARK}}$.
   • To evaluate $\mathsf{Mul}_{\mathbb{F}}(w_a, w_b)$,
     – Isolated mode: receive from $\mathcal{A}$ its shares of $w_a$ and $w_b$, combine
       these with the honest parties' shares, and share the results
       back to $\mathcal{A}$.
     – Collaborative mode: proceed as in $\Pi_{\mathsf{SNARK}}$.
   • To evaluate Reveal, (a) read from $\pi$ the actual wire value $w$
     revealed at this gate, (b) read from the simulated execution of
     honest parties their shares the expected messages $\widetilde{w}$, and then
     (c) broadcast $w - \widetilde{w}$. Upon receiving $w_{\mathcal{A}} = w_{\mathcal{A}} + \varepsilon$ from $\mathcal{A}$, reconstruct
     the (possibly incorrect) public wire value $\tilde{w} := w + \varepsilon$.
   • To evaluate $\mathsf{RO}(w)$, read the expected output $o$ of $\rho$ at that point
     from the proof $\pi$ by running the SNARK verifier $\mathcal{V}$, and return
     to the adversary this output. Add the mapping $(w \mapsto o)$ to the
     programming $\mu$.
   • To handle checks related to the PIOP consistency checker Ch,
     (a) Invoke $\mathbf{Ch_Q}(\mathsf{st}_{\mathbf{Ch}})$ to obtain the query set $Q$ and the expected answers $\boldsymbol{v}$.
     (b) Send $Q$ to $\mathcal{A}$, and receive the shares of the opening proof
         in return.
     (c) Let $\boldsymbol{C}_j$ be the set of commitments corresponding to the
         polynomials $\boldsymbol{p}_j$ being queried in this round.
     (d) If the proof passes, invoke the PC extractor to obtain the
         polynomials committed inside $\boldsymbol{C}_j$. If these extracted polynomials
         are consistent with the polynomials expected from
         the simulation at this point, continue. Otherwise, send
         reject to $\mathcal{F}_{\mathsf{SNARK}}$.
8. Let $\pi'$ be the final proof obtained at the end of the protocol. If
   $\mathcal{V}(\mathrm{ivk}, \mathbb{x}, \pi') = 0$, then send reject to $\mathcal{F}_{\mathsf{SNARK}}$.
9. Output the programming $\mu$ for the random oracle.
```

Figure 10: Simulator $\mathcal{S}_{\mathcal{A}}$ for Π_{SNARK}.

In Fig. 10 we describe a simulator $\mathcal{S}$ for Π_{SNARK}.

We argue that the output of our simulator is correct in two steps. First, we argue that the all steps prior to a Reveal gate are simulated correctly, and then argue that the outputs of Reveal gates are also simulated correctly.

Input. From the perspective of the corrupted parties, the received shares of $\widetilde{\mathbb{w}}$ and r are indistinguishable from random.

Linear gates do not involve any communication between parties. Multiplication gates require exchanging either partial openings that are perfectly hiding (collaborative mode), or perfectly random shares (isolated mode), and thus reveal no information in either the real or the ideal world.

Multiplication gates are handled either by sending shares to the delegator (in isolated mode), or by opening blinded wire values (in collaborative mode). In both cases, the adversary learns nothing about the shared values.

For random oracle gates, if the input is public, then, assuming the random oracle is sampled uniformly at random from $\mathcal{U}(\lambda)$, then no adversary can distinguish between the programmed random oracle and the actual random oracle.

For reveal gates, our circuits reveal two kinds of values: wires representing commitments and evaluation proofs, and wires representing evaluations of prover polynomials. We will analyze each of these.

- *Commitments and evaluation proofs.* In the ideal world, the revealed shares are hiding commitments corresponding to random polynomials, while in the real world, we reveal hiding commitments to the actual shares of the witness. In both worlds, the commitments sum up to the actual commitment output at that gate. Since the PC scheme is hiding, these are indistinguishable. The same holds for evaluation proofs.
- *Evaluations of prover polynomials.* In the ideal world, we reveal evaluations of shares of random polynomials, while in the real world, we reveal evaluations of shares of prover polynomials. Since honest parties in both cases randomize their polynomials in the same way (using fresh randomness), In both worlds, these shares of evaluations sum up to the actual evaluation output at that gate. Since the PIOP is bounded-query zero-knowledge, the distributions of the revealed evaluations are identical in both worlds.

Leakage in consistency checks. There is no leakage in the consistency checks, as the adversary sees neither the evaluations nor the opening proof that is involved in the consistency check.

Probability of rejection due to consistency checks. If the adversary behaves honestly, then by perfect completeness of the PIOP checker and the PC scheme, as well as correctness of the PC extractor, the consistency checks will always pass. If the adversary deviates from the protocol, then it can lead to a distinguishing advantage if either the real check passes, but the ideal check fails, or vice versa.

The first case happens if either the (a) PIOP checker was accepted a polynomial that is not consistent with its checks, or (b) the PC scheme's evaluation binding is broken, or (c) the PC scheme's randomness binding is broken. The probability of all three is negligible, and so by the union bound this happens with negligible probability.

The second case happens if the extractability of the PC scheme is broken: the adversary was able to produce a proof for an invalid claim (we know the claim is invalid from the soundness of the PIOP checker). This happens with negligible probability.

Hence, overall the distinguishing advantage is negligible.

Probability of rejection due to incorrect output proof. In both worlds the delegator checks if the output proof is correct. In both cases, since the previous consistency checks ensure that some part of the output proof corresponds to the delegator's witness, we can invoke knowledge soundness of the scheme to assert that the probability of the real check passing (but the ideal check failing) is negligible.

B Consistency checker for MARLIN

The algorithm described below is a consistency checker for the MARLIN PIOP of [14].

```
Ch^{p_1}(i, x, w):
1. Parse w as a vector $w \in \mathbb{F}^n$, and let $H$ be a smooth multiplicative subgroup of $\mathbb{F}$ of order $n$.
2. Parse the oracles $\boldsymbol{p}_1$ as $(s, \hat{w}, \hat{z}_A, \hat{z}_B)$.
3. Sample an evaluation point $z \leftarrow \mathbb{F}$
4. Query the oracle $\hat{w}$ at $z$, to obtain $v \in \mathbb{F}$.
5. Use barycentric evaluation [5, 12] to *locally* evaluate $\hat{w}$ at $z$ in $O(n)$ field operations, and check that the result is $v$.
```

We now show that this is a consistency checker for the MARLIN PIOP.

Completeness follows from completeness of the MARLIN PIOP: if the PIOP prover is honest, then its oracle $\hat{w}$ will be consistent with the witness w, and so the evaluations will match.

Soundness follows from soundness of the MARLIN PIOP, and in particular of the subPIOPs for holographic lincheck: if $\hat{w}$ is consistent with the witness w, and the PIOP verifier's checks pass, but the other prover polynomials do not equal those produced in the honest execution, then we know that these polynomials take on values inconsistent with the lincheck relation. We can use this fact to then construct an adversary against the lincheck subPIOP, and hence against the MARLIN PIOP, which is a contradiction.