TurboIKOS: Improved Non-interactive Zero Knowledge with Sublinear Memory

No Author Given

No Institute Given

Abstract. In this work, we present a zero knowledge argument for general arithmetic circuits. Our construction is public coins and constant rounds, so it can be made non-interactive and publicly verifiable with the Fiat-Shamir heuristic. The construction is based on the MPC-in-the-head paradigm and requires 3 field elements per multiplication gate, improving upon the 4 field elements required by Baum and Nof [PKC 2020]. We provide two variants of our construction, one that can execute in RAM-constrained environments with a working memory equal to the circuit width (which is necessary to evaluate the circuit) plus a small constant, and another that achieves faster prover runtime. We implement the latter protocol and validate experimentally the expected relationship between proof size and prover runtime.

1 Introduction

Zero knowledge proofs are a useful cryptographic primitive for verifiable yet confidential computing that have found applications in the design of anonymous cryptocurrencies [7, 56] and identification schemes [22]. They are also used as a component within other cryptographic protocols like digital signature schemes [5, 48] and malicious-secure multiparty computation protocols [37, 51]. Both the interactive [38, 39] and non-interactive [15, 29] variants of zero knowledge (ZK) proofs (respectively, arguments) allow an unbounded (resp., polynomially-bounded) prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that a relation C is satisfiable while hiding the witness to this fact. We focus on ZK arguments in this work.

There have been substantial advances over the past decade to improve the efficiency of ZK arguments along several metrics. We categorize these advances into three groups based on their tradeoffs between proof size (or total communication for interactive protocols), RAM requirements, and whether the proofs are verifiable to the general public or a single designated verifier.

First, ZK-SNARKs and ZK-STARKs offer sublinear proof size and verification time (between logarithmic and square root of the circuit size $|C|$) but require the prover to use enormous amounts of memory. There is a long line of research into ZK succinct interactive arguments of knowledge (SNARKs), building upon the work of Killian [50]. Initial constructions required superlinear prover time and per-circuit structured setup [8, 11, 14, 16, 26, 28, 35, 41, 42, 52, 59], and subsequent work achieved linear prover time and permitted universal structured

setup [17, 25, 32, 33, 43, 54, 65]. The newest ZK-SNARKs and ZK scalable transparent arguments of knowledge (STARKs) leverage ideas from interactive oracle proofs [10, 60] or the sumcheck protocol [24, 53] to remove structured setup altogether but have slightly higher proof size [1, 9, 13, 20, 21, 61–63, 67]. Moreover, the large RAM requirement remains.

Second, there exist ZK arguments that scale to large statements due to their moderate RAM requirements (approximately security parameter $\times$ circuit size) and linear prover and verifier runtime, but that sacrifice public verification because they need a designated verifier to maintain secret randomness. ZK proofs based on privacy-free garbled circuits [31, 34, 44, 47, 66] require a designated verifier to garble the circuit and keep the wire labels hidden until the end of the protocol. A separate line of research [3, 64] uses vector oblivious linear evaluation (VOLE) [18, 19, 57] to build proofs with a highly efficient (and optionally non-interactive) online phase, after a one-time interactive preprocessing phase is used to establish correlated randomness between the prover $\mathcal{P}$ and verifier $\mathcal{V}$.

The focus of this work is the remaining situation: when both public verifiability and low RAM utilization are required and a linear proof size is acceptable, the best available constructions are based on the "MPC-in-the-head" paradigm developed by Ishai et al. [45]. These proofs are constructed by executing a secure multiparty computation (MPC) protocol, which only requires fast symmetric key crypto operations and is amenable to the Fiat-Shamir transform [30]. As a result, proofs in the MPC-in-the-head paradigm form the basis of the Picnic digital signature scheme that is currently an "alternate candidate" in round 3 of the NIST post-quantum crypto competition [23, 40, 48, 55].

1.1 Our contributions

In this work, we contribute a new zero knowledge proof in the MPC-in-the-head paradigm that provides *concretely smaller proof sizes* than prior work. Our construction, called TurboIKOS, retains the benefits of all constructions in the MPC-in-the-head paradigm: low RAM utilization (potentially *sublinear* in the circuit size), public verifiability, avoiding structured setup, prover and verifier runtime that are linear in the circuit size $|C|$, and the ability to make the proof non-interactive via the Fiat-Shamir transform.

We describe two variants of TurboIKOS, both of which operate over an NP relation encoded as an arithmetic circuit C over a large field $\mathbb{F}$. The first version is an improvement over Baum-Nof [4], and it is intended for scenarios where it is acceptable for the prover $\mathcal{P}$'s RAM utilization to be linear in the circuit size (Section 3). The second version is intended for RAM-constrained provers; it only requires working memory equal to the circuit width (which is required to execute the circuit at all) plus a small constant (Appendix A). The proof itself is identical in both scenarios, and thus the verification algorithm $\mathcal{V}$ is unchanged. We also implement the first variant and report on our experimental results (Section 4). We defer the security analysis to Appendix B.

Table 1: Proof size (in # of field elements) and soundness error for several MPC-in-the-head protocols. Some lower-order terms are omitted for legibility. Note that ZKBoo and ZKB++ are only constructed for $N = 3$.

Protocol	**Proof size**	**Soundness error**
IKOS+SPDZ [27, 46]	$R \cdot (6MN + (I + O)N)$	$(1/N)^R$
ZKBoo [36]	$R \cdot (2M + 2I + 2O)$	$(2/3)^R$
ZKB++ [23]	$R \cdot (M + I)$	$(2/3)^R$
Katz et al. [48]	$R \cdot (2M + I + \log N + \log_R(P))$	$\max_{0 \le i \le R} \frac{\binom{P-R+i}{P-R}}{\binom{P}{P-R} N^i}$
Baum-Nof [4]	$R \cdot (4M + I + \log N)$	$(1/N)^R$
TurboIKOS (this work)	$R \cdot (3M + I + \log N)$	$(1/N)^R$

1.2 The MPC-in-the-head paradigm

MPC-in-the-head is a method to construct a zero knowledge proof from a secure multiparty computation (MPC) protocol. Given an NP relation encoded as a circuit C, the prover $\mathcal{P}$ runs all parties in a secure computation of C beginning with a sharing of the witness, and the verifier $\mathcal{V}$ challenges $\mathcal{P}$ to open some of the views. Zero knowledge follows from the privacy of the MPC protocol, and soundness is achieved because a malicious $\mathcal{P}$ must have created inconsistent views and $\mathcal{V}$ finds them with noticeable probability. The seminal work of Ishai et al. [45] (also referred to as "IKOS") demonstrated that this transformation works for any MPC protocol, and a subsequent line of works designed specific protocols with increasingly smaller proof size: ZKBoo [36], ZKB++ [23], Katz et al. [48], and Baum-Nof [4].

Table 1 shows proof sizes for MPC-in-the-head constructions in which the prover $\mathcal{P}$ runs R iterations of an MPC protocol, each of which involves N parties securely evaluating a circuit C with I input wires, O output wires, and M multiplication gates. When using an ordinary MPC protocol like SPDZ [27], a multiplication gate requires all parties to broadcast one message that is stored in the resulting proof, yielding in a proof size of $\Omega(MNR)$. To do better, MPC-in-the-head constructions make optimizations that are not acceptable for "normal" MPC protocols: they design *circuit decompositions* that look like MPC party views, yet can only be computed when a single entity $\mathcal{P}$ knows the inputs of all MPC parties. In circuit decompositions, the emulated MPC parties don't communicate to compute the views, but rather only to check their consistency.

We briefly survey the main ideas in each construction and the impact they have on the proof size per multiplication gate, which tends to be the largest contributor to the proof size.

- ZKBoo [36] and ZKB++ [23] are based on the $N = 3$ party replicated secret sharing MPC protocol of Araki et al. [2]; they do not generalize to arbitrary choices of N. All data is secret shared using 3-out-of-3 additive sharing, and addition can be done locally. Multiplication requires sending 3 messages, each of which is a function of a different subset of 2 of the 3 shares of the

input wires. The verifier $\mathcal{V}$ receives two shares, and therefore can verify 1 of the 3 messages sent during each multiplication.
- ZKB++ (and all subsequent works) samples shares pseudorandomly. Given a seed σ_p for each party p, to share a value v_w on wire w, only the *offset* $e_w = v_w + \sum_p \mathsf{PRF}(\sigma_p, w)$ is recorded in the proof, reducing the cost per multiplication gate but requiring a (cheap) initial setup to distribute seeds.
- Katz et al. [48] extends MPC-in-the-head to accommodate MPC protocols with preprocessing. They build Beaver triples using a cut-and-choose approach, where some triples are opened and checked during preprocessing. The proof size ($R \log_R(P)$) required to assist $\mathcal{V}$ in the preprocessing step is independent of the circuit size. The remaining Beaver triples are assumed to be valid and used to verify the real execution.
- Baum-Nof [4] also uses pseudorandom shares and Beaver triples in a variant of the SPDZ protocol, but avoids cut-and-choose in favor of *sacrificing* one Beaver triple to check the validity of each multiplication gate.

Comparing proof sizes. For each multiplication gate: ZKB++ requires 1 field element to represent the offset e_w for the output value, Katz et al. requires 1 more field element to represent the offset for the Beaver triple value, and Baum-Nof requires 2 more field elements to test whether the sacrificed Beaver triple and the circuit values are consistent. So at first glance, it might appear as though proof sizes are getting bigger (i.e., worse) as we go down Table 1. But actually, the newer protocols are better for two reasons:

1. ZKBoo and ZKB++ are restricted to $N = 3$ parties, whereas the subsequent constructions allow for larger N with only a small effect on the proof size. By choosing a larger N, Katz et al. and (especially) Baum-Nof achieve negligible soundness error with a smaller number of iterations R.
2. The Katz et al. protocol has a parameter P that can be chosen arbitrarily, but it turns out that this value must be very large to achieve negligible soundness error and the resulting $R \log_R(P)$ term in the proof size becomes noticeable. Baum-Nof avoids this cost.

1.3 Overview of our construction

The simplest way to describe this work is that we combine the techniques used in the Baum-Nof ZK proof with the Turbospeedz MPC protocol [6] so that sacrificing a Beaver triple costs only one field element, while preserving its soundness error. That having been said: combining these techniques is somewhat delicate. In this section, we describe the Turbospeedz construction and then explain the challenge when integrating it into MPC-in-the-head.

SPDZ and Turbospeedz. The SPDZ line of works [12,27,58] is a popular family of MPC protocols that offloads the (expensive) generation of Beaver triples into a preprocessing phase so that the online phase has free additions and only requires broadcasting 2 elements per multiplication gate (1 per input wire). Turbospeedz

[6] saves 1 element per multiplication gate by exploiting a redundancy: when generating shares of an input wire w pseudorandomly such that the shares of the value are $[v_w] = e_w + [\lambda_w]$, the public offsets e_w can also serve "for free" as the broadcast values for the input wires, and the only effort required is to create the new offset for the 1 output wire.

The challenge of TurboIKOS. When SPDZ is used in MPC-in-the-head to check a multiplication gate whose input and output wires are claimed to be a Beaver triple $\langle v_x, v_y, v_z \rangle$, it suffices to use the semi-honest protocol without MAC checks, and for the prover $\mathcal{P}$ to cheaply generate an independent Beaver triple $\langle \hat{\lambda}_x, \hat{\lambda}_y, \hat{v}_z \rangle$. However, with Turbospeedz there is a problem: the protocol transmits 2 field elements in the preprocessing stage, in addition to the 1 field element in the online stage. This is fine from an MPC perspective where preprocessing work might be viewed as "free," but is unacceptable for MPC-in-the-head where all elements add equally to the proof size.

To overcome this issue, we turn to another member of the SPDZ family: Overdrive [49]. The Overdrive protocol includes a clever method for generating a partially-correlated Beaver triple $\langle \lambda_x, \hat{\lambda}_y, \hat{v}_z \rangle$ where the shares $[\lambda_x]$ for the first element of the Beaver triple are the *same* as the shares for the true value v_x. With a common element between the two Beaver triples, all of the setup calculations become linear steps that can be computed locally by the parties. Integrating Turbospeedz's function-dependent preprocessing with Overdrive's Beaver triple generation mechanism is one of the accomplishments of our TurboIKOS protocol.

Comparisons with other works. This work achieves the best communication complexity among the linear MPC-in-the-head style protocols for semi-honest-secure MPC protocols, by achieving per-party communication that is independent of the size of the circuit. This allows adding more parties cheaply, so we can do fewer repetitions of the inner MPC protocol to achieve the same level of soundness.

Additionally, our protocol is even competitive with MAC'n'Cheese [3] and Wolverine [64], two state-of-the-art designated verifier protocols with low RAM requirements based on vector OLE. When proving knowledge of two 512×512 matrices over a 128-bit field that multiply to a public matrix, the memory-optimized variant of our TurboIKOS protocol requires less than 10 MB of RAM, compared to 60 MB for MAC'n'Cheese and 400 MB for Wolverine. Our 50 GB proof size is decently larger than the 5-6 GB proofs of MAC'n'Cheese and Wolverine, but similar in scope even for a circuit of this scale and our ZK argument is publicly verifiable.

2 Preliminaries

2.1 Notation

Throughout this work, $\mathcal{P}$ denotes the prover and $\mathcal{V}$ denotes the verifier. We let C denote an arithmetic circuit corresponding to the NP relation with a

canonical output message corresponding to logical true (i.e., the witness satisfies the relation). We use ADD, MUL to denote addition and multiplication gates, respectively.

The circuit has a set of gates G of which a subset M are MUL gates, as well as a set W of wires of which there are disjoint subsets I of input wires and O of output wires. By abuse of notation, we use the same variables to denote the size of each set; for instance, we let M denote the number of multiplication gates when it is clear from context that we are describing an integer rather than a set.

We consider an MPC-in-the-head protocol execution with N parties that is repeated R times. If a single iteration of a protocol has soundness error δ, then we can run $R = \lceil \frac{\kappa}{\log(1/\delta)} \rceil$ independent iterations to reduce the soundness error to $2^{-\kappa}$ (where all logarithms are taken base-2 in this work).

For computation and equations, we use $\mathbb{F}$ to refer to a finite field and $\mathbb{F}^*$ to refer to the units of that field. We generally use κ as our security parameter and $[v]$ to refer to an additive secret sharing of a value v among the N parties.

2.2 Definitions

Pseudorandom Functions and Commitments. We require the existence of a pseudorandom function PRF and a computationally hiding commitment scheme Com in our security analysis in Appendix B. Our implementation uses hash-based commitments that models the hash function as a random oracle and assumes that AES acts as a PRF. Below we give the formal definitons for PRF and Com:

Definition 1 (Pseudorandom Function). *Let $\mathcal{F}$: $\{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an efficient, length-preserving, and keyed function. $\mathcal{F}$ is a pseudorandom function if $\forall$ p.p.t. adversaries* A*, there is a negligible function negl(n) such that* $\Pr[PrivK^{PRF}_{\mathsf{A},\mathcal{F}}(n) = 1] \leq \frac{1}{2} + negl(n)$.

Definition 2 (Commitment). *The protocol ($\mathcal{S}$,$\mathcal{R}$) is a commitment scheme with the following algorithms:*

- Com *(m): The sender $\mathcal{S}$ has an input message $m \in \{0,1\}^*$ and security parameter 1^n. The algorithm Commit outputs a pair (c, r) where c is the public commitment and r is the private decommitment randomness.*
- Decom *(c,m,r) $\mathcal{S}$ sends (c,m, r) to the receiver $\mathcal{R}$. $\mathcal{R}$ either accepts and outputs m, or rejects.*

A computationally secure commitment scheme satisfies the following properties:

- ***Completeness:** If (c,r) = Commit(m), then Reveal(c,r) = m.*
- ***Hiding:** For any two message pairs m and m', their respective commitment c and c' are computationally indistinguishable*
- ***Binding:** A p.p.t. adversary wins if it outputs c, r, r' such that Reveal(c, r) = m, Reveal(c, r') = m', and $m \neq m'$ with non-negligible probability*

Honest Verifier Zero-knowledge Argument of Knowledge. Next, we formally define the notion of ZK arguments over an NP-relation $R(x, w)$ as a two-party protocol involving two p.p.t. algorithms, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. Both parties have an NP statement x, and only the prover receives its corresponding witness w. The parties interact to determine whether $R(x, w) = 1$ without revealing the witness. We restrict our attention to the honest verifier setting in which $\mathcal{V}$ never deviates from the protocol.

Definition 3. *The protocol ($\mathcal{P}$, $\mathcal{V}$) is an honest verifier ZK argument for the relation $R(x, w)$ if it satisfies the following properties:*

- ***Completeness:*** *If $R(x, w) = 1$ and both players are honest, $\mathcal{V}$ always accepts.*
- ***Soundness:*** *For any malicious and computationally bounded prover $\mathcal{P}^*$, there is a negligible function negl(·) such that a statement x is not in the language (i.e., $R(x, w) = 0$ for all w), then $\mathcal{V}$ rejects on x with probability $\geq 1 - negl(|x|)$ when interacting with $\mathcal{P}^*$.*
- ***Honest verifier computational zero knowledge:*** *Let $View_{\mathcal{V}(x,w)}$ be a random variable describing the distribution of messages received by $\mathcal{V}(x)$ from $\mathcal{P}(x, w)$. Then, there exists a p.p.t. simulator $\mathcal{SIM}$ such that for all x in the language, $\mathcal{SIM}(x) \approx_c View_{\mathcal{V}(x,w)}$.*

We restrict our attention to honest verifier ZK in this work because our protocol TurboIKOS is also public coins and constant round, so it can be transformed into a non-interactive argument using the Fiat-Shamir transform.

Multi Party Computation (MPC). An MPC protocol allows N players to jointly compute a function of their respective inputs while maintaining the privacy of their individual inputs and the correctness of the output. In addition, the protocol should prevent an adversary who may corrupt a subset of players, from learning additional information or harming the protocol execution. A party's *view* in MPC contains that party's input, randomness, and any messages received by that party.

3 Construction

We present our protocol $\Pi_{\text{TurboIKOS}}$ in this section and in Figure 1. We start by describing the Baum-Nof [4] SPDZ-like protocol and the Turbospeedz MPC protocol. Then, we show how to incorporate Turbospeedz [6] into the MPC-in-the-head paradigm to reduce the amount of communication per MUL gate.

3.1 Starting point: SPDZ and Baum-Nof

We use the MPC-in-the-head paradigm introduced by Ishai et al. (IKOS) [45] combined with the SPDZ MPC protocol [27] as a starting point for our zero-knowledge proof using MPC-in-the-head protocol. In IKOS, a prover simulates

a MPC protocol for all parties and commits to a view for each party containing the party's randomness, input, and messages received. To save proof space, an additional "broadcast channel" is committed to for messages that are sent to all parties, rather than writing the same value in all party views. Then the verifier chooses a subset of the parties and challenges the prover to open the committed views of these parties. The verifier then confirms that the views of the opened parties are *consistent*, that is, the message party i sent to party j is the same in both views. For N-party MPC protocols that *only* send broadcast messages and do not contain any private messages between parties, the verifier opening T parties will have a $\frac{T}{N}$ chance of catching a prover who cheats by creating inconsistent views: the "receiving" half of the message is always revealed in the broadcast channel, and these are checked for consistency with the revealed parties' "sent" messages. By repeating this process R times, the verifier can amplify this probability by a power of R.

We start with the variant of semi-honest SPDZ [27] used by Baum-Nof [4]. Let N denote the set of parties and M denote the set of multiplication gates in the circuit C. The parties hold sharings of the inputs $[x_m]$ and $[y_m]$ for each MUL gate $m \in M$; since this MPC protocol is being emulated by a prover who knows the value on the wire, the parties additionally have a sharing of the gate's output $[z_m]$. The prover generates a random multiplication triple, $\langle a_m, b_m, c_m \rangle$, which will be "sacrificed" to check a multiplication constraint in a MUL gate. The verifier will send a random challenge $\epsilon_m \leftarrow \mathbb{F}$. Each party performs the following:

1. The parties broadcast $[f_m] = \epsilon_m[x_m] + [a_m]$ and $[g_m] = [y_m] + [b_m]$
2. The parties use the recombined f and g to compute

$$[\zeta_m] = \epsilon_m[z_m] - f_m g_m + f_m[b_m] + g_m[a_m] - [c_m]. \tag{1}$$

As Baum-Nof demonstrate, if either $\langle a_m, b_m, c_m \rangle$ or $\langle x_m, y_m, z_m \rangle$ is not a valid multiplication triple, this value ζ_m will be nonzero with probability at least $1 - 1/|\mathbb{F}|$.

To save proof space, rather than broadcasting the ζ_m values for each MUL gate m, an additional challenge variable $\hat{\epsilon}_m$ is sent by the verifier $\mathcal{V}$ and the prover $\mathcal{P}$ responds by sending a linear combination $[Z] = \sum_{m \in M} \hat{\epsilon}_m[\zeta_m]$ of the secret values and public coefficients. If the prover is honest then $Z = 0$, and if the prover is dishonest then at least one of the ζ_m terms is non-zero and thus Z is also nonzero with overwhelming probability.

Naively, this protocol broadcasts $(2M+1)N$ elements since each party broadcasts their f shares and g shares for each multiplication gate, plus $[Z]$. However, the dependency on N can be removed because the verifier $\mathcal{V}$ will corrupt $N-1$ of the parties, so the prover only has to transmit the remaining share of f, g, and Z. We emphasize that all parties' shares must still be committed to before $\mathcal{P}$ knows which party will remain uncorrupted, but only one need be revealed. This can be accomplished using a Merkle tree or by hashing all the broadcast shares and hashing all hashes together as in [48].

Equivalently, we can generate the shares of all values pseudorandomly, with only one public "offset" value per wire. Then, for each multiplication gate, the prover only needs to broadcast the offset values for f and g, along with the true output wire z and the Beaver triple product c. Hence, the proof contains 4 field elements per multiplication gate, as shown in Table 1.

3.2 Introducing Turbospeedz

Turbospeedz [6] generally shows how to have only one broadcast per multiplication gate instead of two in normal SPDZ by adding a function-dependent preprocessing step where the circuit to be computed is known, but the input to the circuit need not yet be known. The idea is to add a sharing of a "mask" on each wire, propagated additively (but not multiplicatively) during preprocessing. Then, the masks of the input wires can serve as the first two elements of a Beaver triple, which is also generated during the preprocessing. Let x and y denote the input wire and z denote the output wire of any gate. Let v_x, v_y, v_z denote the real values on the wires. In the preprocessing phase, the prover performs the following:

1. For each party, the prover generates random "masking shares" $[\lambda_w]$ for each input wire and the output wire of each MUL, w.
2. The prover homomorphically computes the mask shares for each ADD gate internal output wire, $[\lambda_z] = [\lambda_x] + [\lambda_y]$.
3. For each wire w, the prover computes external value, $e_w = \lambda_w + v_w$. In MPC, these external values are public to all parties. In MPC-in-the-head, we will give them to $\mathcal{V}$ in the clear.

In Turbospeedz, given e_x, e_y, and a Beaver triple $\langle a_m, b_m, c_m \rangle$, each party computes their share of MUL gate m's output wire by locally computing

$$\begin{aligned}[v_z] &= e_x e_y - e_y[a_m] - e_x[b_m] + [c_m] \\ &= (v_x + a_m)(v_x + b_m) - (v_y + b_m)[a_m] - (v_y + a_m)[b_m] + [c_m] \\ &= [v_x v_y]\end{aligned}$$

The parties then proceed to compute and open $[e_z] = [v_z] + [\lambda_z]$. Note that this relies on the parties already possessing a sharing of a *valid* Beaver triple $\langle a_m, b_m, c_m \rangle$ in advance.

The upshot of this method is that multiplication gates can be computed using only one opening (e_z) instead of two (d and e in the previous section).

3.3 Adapting Turbospeedz

We incorporate the Turbospeedz method, with some modifications, into the SPDZ-based MPC-in-the-head framework described above. Instead of opening shares of e_z, $\mathcal{P}$ will provide e_z in the clear. However, unlike Turbospeedz, we do not wish to have a costly preprocessing process in which the verifier becomes

Fig. 1: Interactive zero knowledge protocol $\Pi_{\text{TurboIKOS}}$ between prover $\mathcal{P}$ and honest verifier $\mathcal{V}$, given a relation represented as an arithmetic circuit with ADD and MUL gates over a field $\mathbb{F}$. All verifier messages are public coins, so the protocol can be made non-interactive using the Fiat-Shamir transform.

Input: The prover $\mathcal{P}$ and verifier $\mathcal{V}$ receive an input circuit C comprising a set of gates G of which a subset M are MUL or AND gates, along with a set of wires W with subsets of input and output wires I and O, respectively. $\mathcal{P}$ is the sole recipient of a witness. Both parties also receive constants R and N (the latter of which we equate with the set $\{1, \ldots, N\}$ by abuse of notation). The prover $\mathcal{P}$ and verifier $\mathcal{V}$ run R independent executions of the following protocol in parallel.

Function-dependent preprocessing: $\mathcal{P}$ pseudorandomly derives shares $[\lambda_w]$ for each wire $w \in W$ and a Beaver triple for each multiplication gate as follows.

1. Generate a random master seed σ^*. Use a PRG to derive a binary tree with root σ^* until there are as many leaves as parties. Assign the p^{th} leaf as party key σ_p.
2. For all input wires $w \in I$, pseudorandomly derive the share $[\lambda_w]$ for each party $p \in N$ from its key σ_p
3. Go through C layer by layer, starting at the input layer. For every $g \in G$, do the following on gate C_g with input wires x, y and output wire z.
   - If C_g is an ADD gate: assign $[\lambda_z] \coloneqq [\lambda_x] + [\lambda_y]$.
   - If C_g is a MUL gate:
     - Derive $[\lambda_z]$, $[\hat{\lambda}_{y,g}]$, and $[\hat{\lambda}_z]$ for every $p \in N$ from σ_p. (Note that y can be an input to many MUL gates, hence the two indices in $[\hat{\lambda}_{y,g}]$.)
     - Set $\hat{e}_z \coloneqq \lambda_x \cdot \hat{\lambda}_{y,g} + \hat{\lambda}_z$, creating a Beaver triple $\langle \lambda_x, \hat{\lambda}_{y,g}, \hat{e}_z - \hat{\lambda}_z \rangle$.

Interactive phase: Once $\mathcal{P}$ receives the witness, the parties interact as follows.

1. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ executes the circuit to determine the value v_w of each wire $w \in W$, assigns the offset $e_w \coloneqq v_w + \lambda_w$ for each wire $w \in W$, and sends to $\mathcal{V}$:
   - Offsets e_w for input wires $w \in I$, and offsets e_z and $\hat{e}_z$ for the outputs of MUL gates. (The remaining offsets can be computed from these.)
   - A Merkle tree commitment to all shares $[\lambda_w]$ for all output wires $w \in O$.
   - A Merkle tree commitment to the seeds σ_p for all parties $p \in N$.
2. $(\mathcal{V} \to \mathcal{P})$ $\mathcal{V}$ sends $\mathcal{P}$ two random elements $\epsilon_m, \hat{\epsilon}_m \leftarrow \mathbb{F}$ for every MUL gate $m \in M$.
3. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ sends $\mathcal{V}$ all α_m values and a Merkle root of the $[Z]$ shares, which are computed as follows.
   - For every MUL gate $m \in M$, assign $[\alpha_m] \coloneqq \epsilon_m[\lambda_y] + \hat{\epsilon}_m[\hat{\lambda}_{y,m}]$ and $[\zeta_m] \coloneqq \epsilon_m e_z - \epsilon_m e_x e_y + \hat{\epsilon}_m \hat{e}_z + (\epsilon_m e_y - \alpha_m)[\lambda_x] + \epsilon_m e_x[\lambda_y] - \epsilon_m[\lambda_z] - \hat{\epsilon}_m[\hat{\lambda}_z]$.
   - For every party $p \in N$, assign $[Z] \coloneqq \sum_{m \in M}[\zeta_m]$.
4. $(\mathcal{V} \to \mathcal{P})$ $\mathcal{V}$ sends $\mathcal{P}$ random field elements $\gamma_m \leftarrow \mathbb{F}$ for all $m \in M$.
5. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ sends $\mathcal{V}$ a Merkle root of the shares $[A] \coloneqq \sum \gamma_m[\alpha_m] - \sum \gamma_m \alpha_m$ $\forall$ parties $p \in N$.
6. $(\mathcal{V} \to \mathcal{P})$ $\mathcal{V}$ sends $\mathcal{P}$ a randomly-chosen set T of $N - 1$ parties to corrupt.
7. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ opens commitments to $\mathcal{V}$ in the following way:
   - Send a Merkle path excluding party p of the seed tree generated by the PRG in preprocessing step 1, allowing $\mathcal{V}$ to recompute σ_p for corrupted parties $p \in T$.
   - Send p's share of $[Z]$ and $[A]$ along with the Merkle paths in their respective Merkle trees

Verification. $\mathcal{V}$ accepts only if all of the following are true for all executions:

- The output values (v_w for $w \in O$) provided by $\mathcal{P}$ correspond to logical true.
- The shares $[A]$ and $[Z]$ provided by $\mathcal{P}$ satisfy $\sum_{p \in N}[A] = \sum_{p \in N}[Z] = 0$.
- All keys σ_p revealed in round 7 (for the corrupted parties $p \in T$) are valid openings of the commitments made by the prover $\mathcal{P}$ in round 1.
- For all corrupted parties $p \in T$, the prover $\mathcal{P}$ correctly computed the shares of $[A]$, $[Z]$, and each output wire $[\lambda_w]$ from the key σ_p.

convinced of the validity of all Beaver triples in the circuit; instead we wish to use ζ values as in Eq. 1. In order to do this without reusing masks on multiple values, we must make some substantial changes to the original Turbospeedz protocol.

First, to save space, we set the parties' shares $[\lambda_w]$ on each wire pseudorandomly, taking advantage of the external value e as an offset: the value on wire w is defined as simply $v_w = e_w - \lambda_w$.

$\mathcal{P}$ will generate all λ_w values for all wires in the circuit the same as in original Turbospeedz, but by generating each party's share pseudorandomly using a party key. Additionally, for each MUL gate m, the prover will generate additional pseudorandom shares $[\hat{\lambda}_{y,g}]$ and $[\hat{\lambda}_z]$. $\mathcal{P}$ computes $\hat{e}_z = \lambda_x\hat{\lambda}_{y,g} + \hat{\lambda}_z$, forming a new Beaver triple $\langle \lambda_x, \hat{\lambda}_{y,g}, \hat{e}_z - \hat{\lambda}_z\rangle$ that will be sacrificed. The double index on $[\hat{\lambda}_{y,g}]$ is due to the fact that wire y may be reused in several different mult gates, each of which must define their own Beaver triple for the prover's privacy.

The largest change is in how ζ_m is calculated for each MUL m. Similar to the Baum-Nof challenge, $\mathcal{V}$ will send random challenges $\epsilon_m, \hat{\epsilon}_m \leftarrow \mathbb{F}^*$. The prover will send $\alpha_m = \epsilon_m\lambda_y + \hat{\epsilon}_m\hat{\lambda}_y$. Assume for now that this value is honestly computed from the parties' shares. Using the opened α_m, the parties compute

$$[\zeta_m] = \epsilon_m e_z - \epsilon_m e_x e_y + \hat{\epsilon}_m\hat{e}_z + (\epsilon_m e_y - \alpha_m)[\lambda_x] + \epsilon_m e_x[\lambda_y] - \epsilon_m[\lambda_z] - \hat{\epsilon}_m[\hat{\lambda}_z].$$

First, we wish to show that this ζ_m serves a similar purpose to Baum-Nof's. Define $\Delta_z = (e_z - \lambda_z) - (e_x - \lambda_x)(e_y - \lambda_y)$. Define $\hat{\Delta}_z = (\hat{e}_z - \hat{\lambda}_z) - \lambda_x\hat{\lambda}_y$. Observe that if $\mathcal{P}$ is honest, then $\langle e_x - \lambda_x, e_y - \lambda_y, e_z - \lambda_z\rangle$ and $\langle \lambda_x, \hat{\lambda}_y, \hat{e}_z - \hat{\lambda}_z\rangle$ are both valid Beaver triples and therefore $\Delta_z = \hat{\Delta}_z = 0$.

Lemma 1. *If ϵ_m and $\hat{\epsilon}_m$ are chosen uniformly randomly from $\mathbb{F}^*$, and if $\Delta_z \neq 0$ or $\hat{\Delta}_z \neq 0$, then $\zeta_m \neq 0$ with probability at least $1 - 1/|\mathbb{F}^*|$.*

Proof. Observe that

$$\begin{aligned}
\zeta_m &= \epsilon_m e_z - \epsilon_m e_x e_y + \hat{\epsilon}_m\hat{e}_z + (\epsilon_m e_y - \alpha_m)\lambda_x + \epsilon_m e_x\lambda_y - \epsilon_m\lambda_z - \hat{\epsilon}_m\hat{\lambda}_z \\
&= \epsilon_m e_z - \epsilon_m e_x e_y + \hat{\epsilon}_m(\lambda_x\hat{\lambda}_y + \hat{\Delta}_z) + (\epsilon_m e_y - \alpha_m)\lambda_x + \epsilon_m e_x\lambda_y - \epsilon_m\lambda_z \\
&= \epsilon_m e_z - \epsilon_m e_x e_y + \hat{\epsilon}_m\hat{\Delta}_z + (\epsilon_m e_y - \epsilon_m\lambda_y)\lambda_x + \epsilon_m e_x\lambda_y - \epsilon_m\lambda_z \\
&= \epsilon_m e_z - \epsilon_m\lambda_z - \epsilon_m e_x e_y + \epsilon_m e_x\lambda_y + \epsilon_m e_y\lambda_x - \epsilon_m\lambda_y\lambda_x + \hat{\epsilon}_m\hat{\Delta}_z \\
&= \epsilon_m((e_z - \lambda_z) - (e_x - \lambda_x)(e_y - \lambda_y)) + \hat{\epsilon}_m\hat{\Delta}_z \\
&= \epsilon_m\Delta_z + \hat{\epsilon}_m\hat{\Delta}_z
\end{aligned}$$

Now, consider the probability that $\zeta_m = 0$ over the choice of ϵ_m and $\hat{\epsilon}_m$ (both chosen uniformly from $\mathbb{F}^*$). The only way for this to occur is if $\epsilon_m\Delta_z = -\hat{\epsilon}_m\hat{\Delta}_z$. If $\Delta_z = 0$, this is impossible. If $\Delta_z \neq 0$, this occurs with the same probability that $\epsilon_m = -\hat{\epsilon}_m\hat{\Delta}_z\Delta_z^{-1}$, which is $1/|\mathbb{F}^*|$.

Similar to Baum-Nof, we can combine these in a linear combination Z (where the $\hat{\epsilon}_m$ values are already sufficient to serve as challenge coefficients) and, as long

as at least one ζ_m is nonzero then the linear combination Z will be nonzero with probability at least $1/|\mathbb{F}|$.

We must additionally show that the α_m values sent by the prover were honestly computed from the parties' shares $[\alpha_m] = \epsilon_m[\lambda_y] + \hat{\epsilon}_m[\hat{\lambda}_y]$. To do this, we have $\mathcal{V}$ provide another set of challenge coefficients $\gamma_m \leftarrow \mathbb{F}$ for each mult gate m in the set of mult gates M. These form a similar linear combination of values $[A] = \sum_{m \in M} \gamma_m([\alpha_m] - \alpha_m)$ that ought to be zero, namely the sharing of the value minus the revealed value, $([\alpha_m] - \alpha_m)$. All shares $[Z]$ and $[A]$ are broadcast, and $\mathcal{V}$ ensures that they sum to 0 and are properly calculated for the corrupted parties.

Our protocol is described in detail in Figure 1. We prove the privacy and soundness of our ZK proof in Appendix B; completeness follows by inspection.

Improvement over Baum-Nof [4]. For each multiplication gate, Baum-Nof must send the d and e value described in §3.1. Their protocol must also send a Beaver triple offset (analogous to $\hat{e}_z$) as well as the offset for the output wire of the mult gate (similar to e_z). The overall improvement we achieve over Baum-Nof is that whereas Baum-Nof sends these four values per multiplication gate m, our protocol sends only three: e_z, $\hat{e}_z$, and α_m.

4 Implementation and Experimental Results

We created a prototype implementation for our protocol in Python. We did not make any effort to optimize the runtime or memory usage of our code, thus, we will likely not compete on prover runtime with other MPC-in-the-head approaches written in C or C++, e.g. [4, 48]. In this section we briefly describe our implementation and then we describe concrete results.

4.1 Implementation details

Our Python implementation[1] supports both Bristol circuit formats and PWS formats as input. The circuit is parsed into a list of Gate objects, found in gate.py, and initializes a Wire data structure, found in wire.py, that takes in a list of dictionaries containing the values on each wire.

Dictionaries were used extensively to manage information. Circuit information such as the number of ADD gates and number of MUL gates were stored in a dictionary. Values on each wire such as e and λ were also stored in a dictionary. Each wire had a dictionary of values, resulting in a list of dictionaries with length of the number of wires. This list of dictionaries is later used as the input to the Wire object, found in wire.py, which defines functions to access values on the wire. Dictionaries were also used by the Prover to build the open broadcasts, later sent to the verifier in Round 7.

[1] Our implementation (anonymized for submission) can be found at https://anonymous.4open.science/r/6d5d149b-d8e3-437a-b726-9e60ebcab5b7/

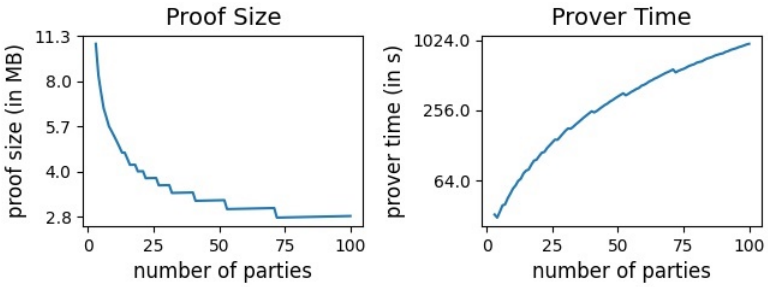

Fig. 2: Metrics progression as number of parties increases for 16×16 matrix multiplication

Party Management. On the Prover side, $\mathcal{P}$ calculates all the parties as she parses through the circuit, so $\mathcal{P}$ uses the Wire objects to access a party's value. On the Verifier side, $\mathcal{V}$ picks one party to leave unopened and reconstructs the other $N-1$ views from the seeds given by $\mathcal{P}$.

Pseudorandom inputs and lambdas (Preprocessing). We generated the values λ, $\hat{\lambda}_y$, and $\hat{\lambda}_z$ pseudorandomly based on a party's random key. The lambdas were determined by using AES as a pseudorandom function using the party's key, and the concatenation of the (fixed-length) wire index and type of value as the message.

Commitments. The prover is required to send a commitment in Round 1, Round 3, and Round 5. Our commitment scheme uses $\mathsf{Com}(m) := H(r, m)$ as the commit algorithm, and $\mathsf{Decom}(d, r, x)$ is true if and only if $d = H(r, x)$. This is a computationally hiding and binding commitment scheme if H is a random oracle. We use SHA2 from `hashlib` for our instantiation of H.

Fiat-Shamir Transform [30] . To achieve non-interactivity, the random messages sent by the Verifier in Round 2, Round 4, and Round 6, are replaced by a hash (random oracle) call on the info sent by $\mathcal{P}$ in all previous rounds.

4.2 Concrete results

We run our experiments on an Amazon EC2 instance of type r6g.4xlarge with 16 vCPUs and 128GB of RAM. The machine runs Ubuntu 20.04 LTS and we run our Python3 implementation with command line arguments. All experiments target at a 2^{-80} soundness error.

We test our implementation on a circuit where $\mathcal{P}$ proves that two private matrices over $\mathbb{F}_q$ (where q is the Mersenne prime 2^127-1) multiply to a public matrix. This has become a de-facto benchmark for arithmetic circuits in ZK proofs (e.g. [3, 64]).

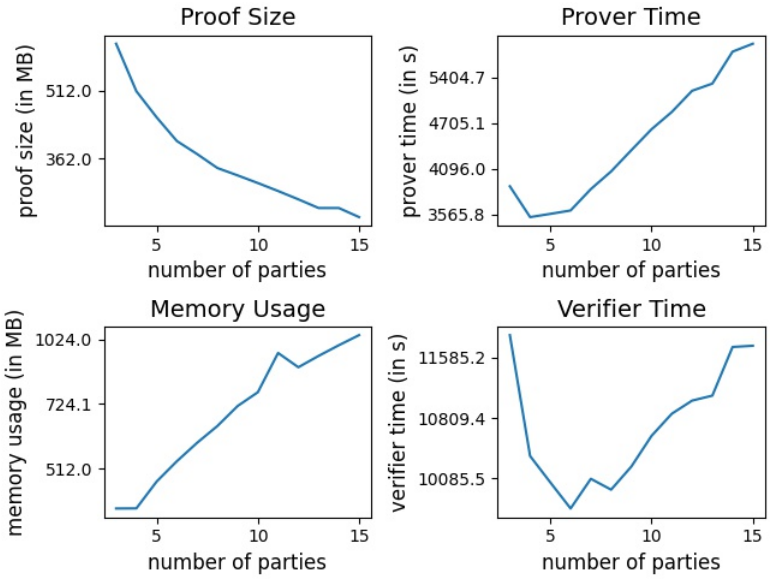

Fig. 3: Metrics progression as number of parties increases for 64×64 matrix multiplication

In Figure 2, we show how proof size and prover time change as the number of parties increases for a 16×16 matrix multiplication. We ran experiments for all points from 3 parties to 100 parties for a detailed visualization. The graph clearly shows a "trade-off" between proof size and prover time for our protocol. As the number of parties increases, the proof size decreases and the prover time increases. The proof size includes all messages sent by $\mathcal{P}$; we ignore $\mathcal{V}$'s messages because in the non-interactive version they can be computed from $\mathcal{P}$'s messages. The "staircase" shape in the proof size graph illustrates how proof size depends strongly on the number of repetitions required by the specified soundness error: each time we increase the number of parties past a threshold that lowers the number of repetitions, the proof size drops dramatically, then slowly climbs back up as we add more parties within the same number of repetitions. Theoretically, this process could be increased all the way to an exponential number of parties at only one repetition, however this would lead to a prohibitively large runtime.

In Figure 3, the graph shows our protocol execution for a 64×64 matrix multiplication for our metrics: proof size, prover memory, prover time, and verifier time. The proof size shrinks by about 2.5× from 3 parties to 15 parties, while the prover time and verification time almost remains the same.

In Table 2, we show extreme values for circuits of different sizes. The overall running time is slower than expected due to the nature of Python and we had to skip some of the expensive operations. But we can still see the huge drop for proof size as the number of parties grows.

circuit	# parties	prover time(in s)	verifier time(in s)	proof size	prover memory(in MB)
matmult_16	5	33	47	10.7 MB	6
	100	949	523	2.8 MB	90
matmult_64	5	3575	10040	651.8 MB	477
	100	61907	36616	166.3 MB	6827

Table 2: Proof metrics for circuits of different sizes. Matmult_n indicates a an $n \times n$ matrix.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134104
2. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 805–817. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978331
3. Baum, C., Malozemoff, A.J., Rosen, M., Scholl, P.: Mac'n'cheese: Zero-knowledge proofs for arithmetic circuits with nested disjunctions. Cryptology ePrint Archive, Report 2020/1410 (2020), https://eprint.iacr.org/2020/1410
4. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 495–526. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45374-9_17
5. Bellare, M., Goldwasser, S.: New paradigms for digital signatures and message authentication based on non-interative zero knowledge proofs. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 194–211. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_19
6. Ben-Efraim, A., Nielsen, M., Omri, E.: Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 19. LNCS, vol. 11464, pp. 530–549. Springer, Heidelberg (Jun 2019). https://doi.org/10.1007/978-3-030-21568-2_26
7. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
8. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1_6
9. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17653-2_4

10. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 31–60. Springer, Heidelberg (Oct / Nov 2016). https://doi.org/10.1007/978-3-662-53644-5_2
11. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 781–796. USENIX Association (Aug 2014)
12. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20465-4_11
13. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A new optimized sublinear IOP. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 20. pp. 2025–2038. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417893
14. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (Mar 2013). https://doi.org/10.1007/978-3-642-36594-2_18
15. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC. pp. 103–112. ACM Press (May 1988). https://doi.org/10.1145/62212.62222
16. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 327–357. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_12
17. Bootle, J., Cerulli, A., Ghadafi, E., Groth, J., Hajiabadi, M., Jakobsen, S.K.: Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 336–365. Springer, Heidelberg (Dec 2017). https://doi.org/10.1007/978-3-319-70700-6_12
18. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 896–912. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243868
19. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_16
20. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00020
21. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 677–706. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45721-1_24
22. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7
23. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures

from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133997
24. Chiesa, A., Forbes, M.A., Spooner, N.: A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305 (2017), http://eprint.iacr.org/2017/305
25. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part I. LNCS, vol. 12105, pp. 738–768. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45721-1_26
26. Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy. pp. 253–270. IEEE Computer Society Press (May 2015). https://doi.org/10.1109/SP.2015.23
27. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_38
28. Danezis, G., Fournet, C., Groth, J., Kohlweiss, M.: Square span programs with applications to succinct NIZK arguments. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 532–550. Springer, Heidelberg (Dec 2014). https://doi.org/10.1007/978-3-662-45611-8_28
29. De Santis, A., Persiano, G.: Zero-knowledge proofs of knowledge without interaction (extended abstract). In: 33rd FOCS. pp. 427–436. IEEE Computer Society Press (Oct 1992). https://doi.org/10.1109/SFCS.1992.267809
30. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
31. Frederiksen, T.K., Nielsen, J.B., Orlandi, C.: Privacy-free garbled circuits with applications to efficient zero-knowledge. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 191–219. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_7
32. Gabizon, A.: AuroraLight: Improved prover efficiency and SRS size in a sonic-like system. Cryptology ePrint Archive, Report 2019/601 (2019), https://eprint.iacr.org/2019/601
33. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019), https://eprint.iacr.org/2019/953
34. Ganesh, C., Kondi, Y., Patra, A., Sarkar, P.: Efficient adaptively secure zero-knowledge from garbled circuits. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part II. LNCS, vol. 10770, pp. 499–529. Springer, Heidelberg (Mar 2018). https://doi.org/10.1007/978-3-319-76581-5_17
35. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_37
36. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 1069–1083. USENIX Association (Aug 2016)

37. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
38. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM **38**(3), 691–729 (1991)
39. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989)
40. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone: Status report on the second round of the nist post-quantum cryptography standardization process. https://csrc.nist.gov/publications/detail/nistir/8309/final (2020)
41. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Heidelberg (Dec 2010). https://doi.org/10.1007/978-3-642-17373-8_19
42. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (May 2016). https://doi.org/10.1007/978-3-662-49896-5_11
43. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 698–728. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_24
44. Heath, D., Kolesnikov, V.: Stacked garbling for disjunctive zero-knowledge proofs. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 569–598. Springer, Heidelberg (May 2020). https://doi.org/10.1007/978-3-030-45727-3_19
45. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007). https://doi.org/10.1145/1250790.1250794
46. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 433–442. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374438
47. Jawurek, M., Kerschbaum, F., Orlandi, C.: Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 955–966. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516662
48. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243805
49. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_6
50. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992). https://doi.org/10.1145/129712.129782
51. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EURO-

CRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (May 2007). https://doi.org/10.1007/978-3-540-72540-4_4
52. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 41–60. Springer, Heidelberg (Dec 2013). https://doi.org/10.1007/978-3-642-42033-7_3
53. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. In: 31st FOCS. pp. 2–10. IEEE Computer Society Press (Oct 1990). https://doi.org/10.1109/FSCS.1990.89518
54. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339817
55. Microsoft Corporation: Picnic. https://microsoft.github.io/Picnic/
56. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed E-cash from Bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE Computer Society Press (May 2013). https://doi.org/10.1109/SP.2013.34
57. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: 31st ACM STOC. pp. 245–254. ACM Press (May 1999). https://doi.org/10.1145/301250.301312
58. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (Aug 2012). https://doi.org/10.1007/978-3-642-32009-5_40
59. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society Press (May 2013). https://doi.org/10.1109/SP.2013.47
60. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 49–62. ACM Press (Jun 2016). https://doi.org/10.1145/2897518.2897652
61. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 704–737. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56877-1_25
62. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020), https://eprint.iacr.org/2020/1275
63. Wahby, R.S., Tzialla, I., shelat, a., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: 2018 IEEE Symposium on Security and Privacy. pp. 926–943. IEEE Computer Society Press (May 2018). https://doi.org/10.1109/SP.2018.00060
64. Weng, C., Yang, K., Katz, J., Wang, X.: Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925 (2020), https://eprint.iacr.org/2020/925
65. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 733–764. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_24

66. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole - reducing data transfer in garbled circuits using half gates. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_8
67. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy. pp. 859–876. IEEE Computer Society Press (May 2020). https://doi.org/10.1109/SP40000.2020.00052

A Memory-optimized version

With some changes to the protocol, we can obtain a variant of our protocol that is highly memory-optimized. It outperforms the next best alternative of Mac'n'Cheese [3] by a factor of 7. However, this memory optimization comes at a modest cost to prover runtime – the circuit is re-evaluated three times. Furthermore, it comes at an extreme cost to round complexity, which increases to be proportional to the circuit size. This version of the protocol therefore loses compatibility with the Fiat-Shamir transform and we do not believe it can be made non-interactive by straightforward means.

A.1 Protocol changes

We proceed to present the interactive, designated malicious verifier version of our protocol that is memory-optimized. For simplicity, this version of the protocol allows a term of N rather than $\log N$ in the complexity; this should be removable.

1. Before Round 1 of the original protocol, $\mathcal{V}$ rolls all random challenges in advance: $\epsilon_m, \hat{\epsilon}_m$ and γ_m for all gates m, and the party $i^* \in \{1, \ldots, N\}$ chosen to leave un-revealed. She places these in three commitment structures and sends them to $\mathcal{P}$:
   - She generates a normal commitment c_{party} to commit to the unrevealed party index i^*, and sends c_{party} to the prover. ($\mathcal{P}$ allocates ℓ_{Com} memory to store it.).
   - She places all γ_m values, in topological order, in a Merkle Tree with M items. She sends the Merkle root c_{gamma} to the prover. ($\mathcal{P}$ allocates an additional ℓ_σ to store it.)
   - She places all $(\epsilon_m, \hat{\epsilon}_m)$ values, again in topological order, in another Merkle Tree, for a total of M items. She sends the Merkle root c_{eps} to the prover. ($\mathcal{P}$ allocates an additional ℓ_σ to store it.)
2. The prover executes **Preprocessing and Round 1** in the following way:
   (a) The prover chooses N keys, each of length ℓ_σ, allocating an additional $N\ell_\sigma$. In turn, she commits to each of these and stores the randomness of length ℓ_r (allocating an additional $N\ell_r$).
   (b) The prover must now generate and send e_w for the input wires, e_z for the outputs of mult gates, $\hat{e}_z$ for all mult gates, and $[\lambda_w]$ for the output wires. $\mathcal{P}$ allocates additional memory able to store one field element times the

circuit width, as measured by the maximum number of wires needed as input to a layer of gates. $\mathcal{P}$ goes through the circuit in topological order and does the following:

i. For each input wire w, it computes λ_w by keeping a running sum of the re-computed $[\lambda_w]$ shares. Once it has λ_w, it adds its input value v_w, sends the result e_w to the $\mathcal{V}$ and also stores it in memory.
ii. To go through an ADD gate, $\mathcal{P}$ adds the two input e values and stores the result. At any point, if a wire is no longer needed, $\mathcal{P}$ overwrites it with the next e. There should be enough space within the circuit-width to store all wires needed at a time.
iii. To go through a MUL gate, $\mathcal{P}$ first computes $\hat{e}_z$ by summing all parties' recomputed shares ($\hat{e}_z = \lambda_x \hat{\lambda}_y + \hat{\lambda}_z$) similar to how it computed λ_w for the input wires. It sends this to $\mathcal{V}$. $\mathcal{P}$ then computes e_z pseudorandomly and continues through the circuit.
iv. At the end of the circuit, $\mathcal{P}$ recomputes each party's $[\lambda_w]$ for each output wires and sends them in turn.

3. **Rounds 2 and 3** are replaced by the following process. The goal is to send the α values and $[Z]$ without going beyond memory proportional to the circuit width. Essentially, $\mathcal{P}$ will slowly recompute the circuit gate by gate, sending α values, and building accumulators for $[Z]$. $\mathcal{P}$ allocates N additional field elements' worth of memory and initializes them to 0. These slots will eventually hold $[Z]$. For $m \in M$:
   (a) $\mathcal{V}$ sends $(\epsilon_m, \hat{\epsilon}_m)$ along with MP_1 (allocating another ℓ_σ), the other leaf of the Merkle path leading to $(\epsilon_m, \hat{\epsilon}_m)$ in the
   (b) $\mathcal{P}$ begins the process of evaluating the Merkle tree by hashing the received $(\epsilon_m, \hat{\epsilon}_m)$ and hashing the result concatenated with MP_1.
   (c) $\mathcal{V}$ sends $\mathcal{P}$ the merkle path values MP_i one at a time, and $\mathcal{P}$ hashes them in with the previous value, until the root is reached.
   (d) $\mathcal{P}$ recomputes the circuit up to mult gate m (using the real values, but remaining with the allocated memory proportional to circuit width.
   (e) For each party, $\mathcal{P}$ recomputes $[\zeta]$ and adds it to the appropriate running tally of $[Z]$.
   (f) Share by share, $\mathcal{P}$ computes and sends the α_m value.

   After the circuit has been fully rerun through, $\mathcal{P}$ sends the accumulated $[Z]$ for all parties to $\mathcal{V}$.
4. **Rounds 4 and 5** operate similarly to the replacement of round 2-3, but for generating $[A]$ rather than $[Z]$. $\mathcal{P}$ starts a zeroed accumulator for each $[A]$. $\mathcal{V}$ sends each γ_m in turn. $\mathcal{V}$ also re-sends the ϵ and $\hat{\epsilon}$ values for the gates. At each gate, $\mathcal{P}$ updates each party's $[A]$ appropriately. This step does not require allocating any memory beyond what has already been allocated.
5. **Rounds 6 and 7** are unchanged from the original protocol. $\mathcal{V}$ requests $N-1$ parties to open; $\mathcal{P}$ decommits to the appropriate keys it stored at the beginning.

   The repetitions are performed sequentially, so no additional memory is used.

A.2 Discussion

Observe that the memory used by the Prover in the above protocol is only $|\mathbb{F}|\mathsf{Width}(C) + N(\ell_\sigma + \ell_r + |\mathbb{F}|) + 3\ell_\sigma + \ell_{\mathsf{Com}}$ plus whatever space is needed to evaluate the PRF, compute the commitment, or perform the hashing operation in the Merkle tree.
As a benchmark, we compute the memory needed to show the existence of two 512×512 matrices that multiply to a public matrix of 128-bit field elements. The circuit width of this matrix multiplication is $2 \cdot 512^2$ wires. We suppose party keys, commitment randomness, hash and commitment length are all 256 bits. This can be computed with only 8.4 MB of memory, and much less than the 60.2 MB of Mac'n'Cheese [3] and the $\approx 400MB$ of Wolverine [64] for the same calculation on a field half as large.

B Security analysis

In this section we prove the honest-verifier zero-knowledge and soundness properties of our protocol.

Theorem 1. *When instantiated with a pseudorandom function PRF and a computationally hiding commitment scheme Com, Protocol $\Pi_{TurboIKOS}$ run with N parties and R repetitions is honest-verifier computational zero knowledge with distinguishing bound at most $R \cdot (Adv_{\mathsf{Com}} + Adv_{\mathsf{PRF}})$.*

Proof. We prove the statement for a single repetition, from which the theorem follows by a union bound over the independent repetitions. Let I and M' represent the set of input wires and MUL-gate-output wires respectively. Consider a simulator that follows the following steps during the interactive protocol.

1. The simulator uniformly randomly picks all verifier challenges: all $\epsilon_m, \hat{\epsilon}_m$ in round 2, and γ_m for all mult gates m in round 4, and a party $i^* \in N$ in round 6 to be the "uncorrupted party" whose key will not be revealed to $\mathcal{V}$.
2. Choose keys σ_p for all parties $p \in N$ uniformly at random, honestly following step 1 of preprocessing.
3. For all corrupted parties, derive all shares $[\lambda_w]$ for all wires w from the keys honestly, as in step 3 of the preprocessing.
4. For the output wires $w \in O$, choose the e_w values uniformly at random, and set party i^*'s share of λ_w such that $v_w = e_w - \lambda_w$ represents logical true.
5. Now, work backward through the circuit in reverse topological order.
   (a) For ADD gates, choose a random setting of the e_x and e_y values on the input wires to the ADD gate, conditioned on meeting the linear constraints induced by all ADD gates at this layer.
   (b) For a MUL gate with input wires x and y and output wire z, the values of e_x, e_y, and the corrupted shares of $\hat{\lambda}_y$, $\hat{\lambda}_z$, and $\hat{e}_z$ must all be simulated. Note that e_x, e_y, or both may already be set by an existing constraint (e.g. if the wire was reused in a later layer or used in multiple gates).

i. Generate $\hat{\lambda}_y$ and initialize the $\hat{\lambda}_z$ shares honestly for the corrupted parties from the party keys (leaving it unspecified for party i^*).
ii. Generate $\hat{e}_z$ uniformly (and also e_x or e_y, if they are unspecified).
iii. Let Δ_z be the difference between the value on the output wire z and the product of the value on the input wires xy. That is, $\Delta_z = v_z - v_x v_y = e_z - e_x e_y - \lambda_z + e_x\lambda_y + e_y\lambda_x - \lambda_x\lambda_y$. Let $\hat{\Delta}_z$ be the difference between $(\hat{e}_z - \hat{\lambda}_z)$ and $\lambda_x\hat{\lambda}_y$; that is, $\hat{\Delta}_z = \hat{e}_z - \hat{\lambda}_z - \lambda_x\hat{\lambda}_y$. (For an honest prover, $\Delta_z = \hat{\Delta}_z = 0$, but the simulator is not honest so these values are non-zero.) Using the foreknowledge of ϵ_m and $\hat{\epsilon}_m$, alter party i^*'s share of $\hat{\lambda}_z$ so that $\zeta = \epsilon_m\Delta_z + \hat{\epsilon}_m\hat{\Delta}_z$ equals 0.

6. Commit honestly (and separately) to each of the party keys σ_p. Send the offsets e_z for all $z \in I \cup M'$. Also send all parties' shares $[\lambda_z]$ for all output wires $z \in O$.
7. In round 3, compute and send the α values honestly for all parties. Also, compute and send $[Z]$ using the dishonestly manipulated ζ shares and the blinding value.
8. In round 5, compute and send $[A]$ honestly for corrupted parties, and choose party i^*'s share such that $A = 0$.
9. In round 7, honestly open the key commitments for all parties except i^*.

It is straightforward to confirm that the simulated proof passes all verification checks. It remains to show that the simulated proof is computationally indistinguishable from a real one. As a stepping stone, we consider a hybrid proof H that is constructed like the real one, except using an ideal commitment scheme Com (where it is impossible to recover an un-opened key) and a truly random function in place of PRF. The distinguishing probability between the real game and H is at most $\mathrm{Adv}_{\mathsf{Com}} + \mathrm{Adv}_{\mathsf{PRF}}$.

In the hybrid world, we claim that all of the information provided by $\mathcal{P}$ to $\mathcal{V}$ throughout the proof is meaningless. The commitment to σ_{i^*} is unbreakable, values like the output wires or Z are publicly known to $\mathcal{V}$ beforehand, and the remaining information (e_w for all wires $w \in W$, $\hat{e}_z$ and α_m values for all MUL gates, and the shares $[Z]$) contains masks that hide the real values from $\mathcal{V}$.

- On each wire w, the revealed $e_w = v_w + \lambda_w$ does not reveal anything about the value on the wire v_w because it is masked by party i^*'s share of λ_w.
- For each MUL gate $m \in M$, information in α_m is masked by i^*th share of $\hat{\lambda}_y$. This makes the linear combination A masked by these as well.
- For each MUL gate, $\hat{e}_z$ hides info about $\lambda_x\hat{\lambda}_y$ by masking it with party i^*'s share of $\hat{\lambda}_z$.
- Party i^*'s share of Z reveals no information because it can be computed as $-\sum_{p\in T}[Z]$ (leveraging the fact that $Z = 0$ is public knowledge), and the corrupted parties' shares of Z are only a function of their own data.

All of these masks are truly random in the hybrid world. Observe that e_w, $\hat{e}_z$ and α_m all have the uniform distribution in the simulated world as well. Therefore, the distance between these games is 0, which completes the proof.

Theorem 2. *When instantiated with a a statistically (resp. computationally) binding commitment scheme* Com *with error* δ*, then Protocol* $\Pi_{TurboIKOS}$ *is a sound proof (resp. argument) with soundness error* $(1/N + \delta + 1/|\mathbb{F}| + 1/|\mathbb{F}^*|)^R$.

Proof. Since ADD gates do not contribute to the view, in order to cheat, a prover must either lie about a MUL gate or lie about the shares of the output wires. There are five possible ways this could happen within a single repetition:

1. $\mathcal{P}$ cheated on the commitment to the party seeds and so the shares were different than those committed to.
2. The alpha used to compute ζ was not correct, that is, $\alpha_m \neq \epsilon\lambda_y + \hat{\epsilon}_m\hat{\lambda}_y$.
3. There exists a multiplication gate over wires x, y, and z where $\Delta_z \neq 0$, that is, $\langle v_x, v_y, v_z\rangle = \langle e_x - \lambda_x, e_y - \lambda_y, e_z - \lambda_z\rangle$ is not a valid multiplication triple.
4. There exists a multiplication gate over wires x, y, and z where $\hat{\Delta}_z \neq 0$, that is, $\langle \lambda_x, \hat{\lambda}_y, \hat{v_z}\rangle = \langle \lambda_x, \hat{\lambda}_y, \hat{e}_z - \hat{\lambda}_z\rangle$ is not a valid multiplication triple.
5. There exists an addition gate over wires x, y, and z for which $v_x + v_y \neq v_z$.

The binding property of the commitment should catch the first case except with negligible probability δ.

If the second case is true, then the honest $[A]$ will not reconstruct to A. $\mathcal{V}$ will reject if the honest shares are used, so the prover must cheat in at least one view (by altering that party's share of A). This will be caught with probability $1 - 1/N$ based on $\mathcal{V}$'s party choice.

For the remainder of this section, we thus assume the α_m values are correct. If either case 3 or case 4 is true, then with probability $1 - 1/|\mathbb{F}^*|$ over $\mathcal{V}$'s choice of ϵ_m and $\hat{\epsilon}_m$, the honest ζ_m value for that gate m will be nonzero. If that is true, then Z will be nonzero with probability $1 - 1/|\mathbb{F}|$. $\mathcal{V}$ will reject if this occurs honestly; if the prover cheats in a view to make the fake Z be zero, this will be caught with probability $1 - 1/N$ based on the party choice. By a union bound, the soundness error cheating prover in case 2 or 3 is at most $1/|\mathbb{F}^*| + 1/|\mathbb{F}| + 1/N$.

For the last case, since no communication between parties occurs during ADD gates and the parties simply sum their own shares, the only way for this to occur is if $\mathcal{P}$ lied about at least one view (i.e. set $[\lambda_z] \neq [\lambda_x] + [\lambda_y]$). This is caught with probability at least $1 - 1/N$.

All view-modifications are caught together with error $1/N$ per repetition. This does not need to be union bounded across the different ways views are modified; if any cheating via view-modification exists, it will be caught with the same $1/N$ chance. Doing a union bound over this plus all sources of error shows that the soundness of each inner repetition is $1/N + \delta + 1/|\mathbb{F}| + 1/|\mathbb{F}^*|$. Since $\mathcal{P}$ must cheat on all repetitions if it is to cheat in the protocol overall, the overall soundness error of this protocol is $(1/N + \delta + 1/|\mathbb{F}| + 1/|\mathbb{F}^*|)^R$.