TurboIKOS: Improved Non-interactive Zero Knowledge and Post-Quantum Signatures

Yaron Gvili[1], Julie Ha[2], Sarah Scheffler[2], Mayank Varia[2], Ziling Yang[2], and Xinyuan Zhang[3]

[1] Cryptomnium LLC
yaron.gvili@cs.tau.ac.il
[2] Boston University
{hajulie,sscheff,varia,zilyang}@bu.edu
[3] George Mason University
xzhang44@gmu.edu

Abstract. In this work, we present a zero knowledge argument for general arithmetic circuits that is public-coin 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, in which the prover jointly emulates all MPC protocol participants and can provide advice in the form of Beaver triples whose accuracy must be checked by the verifier. Our construction follows the Beaver triple sacrificing approach used by Baum and Nof [PKC 2020]. Our improvements reduce the communication per multiplication gate from 4 to 2 field elements, matching the performance of the cut-and-choose approach taken by Katz, Kolesnikov, and Wang [CCS 2018] and with lower additive overhead for some parameter settings. We implement our protocol and analyze its cost on Picnic-style post-quantum digital signatures based on the AES family of circuits.

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 [10,64] and identification schemes [27]. They are also used as a component within other cryptographic protocols like digital signature schemes [8,56] and malicious-secure multiparty computation protocols [44,59]. Both the interactive [45,46] and non-interactive [19,36] 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 [58]. Initial constructions required superlinear prover time and per-circuit structured setup [11,14,18,21,32,34,42,47,48,60,67], and subsequent work achieved linear prover time and permitted universal structured setup [22,30,39,40,49,62,74]. The newest ZK-SNARKs and ZK scalable transparent arguments of knowledge (STARKs) leverage ideas from interactive oracle proofs [13,68] or the sumcheck protocol [29,61] to remove structured setup altogether but have slightly higher proof size [3,12,17,25,26,69,70,72,76]. 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 [38,41,51,54,75] 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 [5,73] uses vector oblivious

linear evaluation (VOLE) [23, 24, 65] 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. [52]. 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 [37]. 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 [1, 28, 56, 63].

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, 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 [6] that reduces the number of field elements sent per gate from 4 to 3, and is intended for circuits with large field size (Section 3.3). The second version further reduces the number of field elements sent per mult gate from 3 to 2, and uses a modified batched consistency check that allows the technique to be used in smaller fields (Section 3.4). We analyze our security in Section 4. We describe our implementation of our first variant and evaluate the proof size of our second variant in Section 5.

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. [52] (also referred to as "IKOS") demonstrated that this transformation works for any MPC protocol. Subsequently, a line of works designed specific protocols with increasingly smaller proof size: ZKBoo [43], ZKB++ [28], Katz et al. [56], and Baum-Nof [6].

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 [33], 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 [43] and ZKB++ [28] are based on the $N = 3$ party replicated secret sharing MPC protocol of Araki et al. [4]; 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.

Table 1: Proof size (in # of field elements) and soundness error (for large fields) for several MPC-in-the-head protocols. Some lower-order terms are omitted for legibility. N is the number of parties, M is the circuit size (number of multiplication gates), I and O are the number of input and output wires for the circuit, respectively, and R is the number of times the protocol is repeated. Note that ZKBoo and ZKB++ are only constructed for $N = 3$. P is a parameter specific to [56] indicating how many Beaver triples are committed to in advance.

Protocol	**Proof size**	**Soundness error**
IKOS+SPDZ [33,53]	$R \cdot (6MN + (I+O)N)$	$(1/N)^R$
ZKBoo [43]	$R \cdot (2M + 2I + 2O)$	$(2/3)^R$
ZKB++ [28]	$R \cdot (M + I)$	$(2/3)^R$
Katz et al. [56]	$R \cdot (2M + I + \log N + \log_R(P))$	$\max_{0 \le i \le R} \frac{\binom{P-R+i}{P-R}}{\binom{P}{P-R} N^i}$
Baum-Nof [6]	$R \cdot (4M + I + \log N)$	$(1/N)^R$
$\Pi_{TurboIKOS}$ *(this work)*	$R \cdot (3M + I + \log N)$	$(1/N)^R$
$\tilde{\Pi}_{TurboIKOS}$ *(this work)*	$R \cdot (2M + I + \log N + NU)$	See Theorem 3

- ZKB++ and all subsequent works sample 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. [56] 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 [6] also uses pseudorandom shares and Beaver triples in a variant of the SPDZ MPC protocol, but avoids cut-and-choose in favor of *sacrificing* one Beaver triple to check the validity of each multiplication gate.

For each multiplication gate: ZKB++ requires 1 field element to represent the offset e_w for the output value (but requires more repetitions than the rest), 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. In this work, we introduce two new sacrificing-based MPC-in-the-head constructions that require 1 and then 0 field elements to perform this consistency test; the latter introduces an additive overhead that can be smaller than that of Katz et al. for some parameter settings. See Table 1 for more details about the proof size for each protocol.

1.3 Overview of our construction

The simplest way to describe our first protocol variant is that we combine the techniques used in the Baum-Nof ZK proof with the Turbospeedz MPC protocol [9] so that sacrificing a Beaver triple costs only one field element instead of two, while preserving the soundness error. Our second variant replaces the remaining field element per multiplication gate with some prover advice about the overall circuit, reducing the proof size so that it is competitive with Katz et al. [56] but with a different set of parameter tradeoffs. In this section, we briefly describe the Turbospeedz construction and explain the challenge when integrating it into MPC-in-the-head.

SPDZ and Turbospeedz. The SPDZ line of works [15, 33, 66] 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 [9] 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 [57]. 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.

Implementing Picnic digital signatures. We provide an open source implementation of our protocol [50] and evaluate our proof size when using a variant of the Picnic post-quantum digital signature scheme [63] that uses AES as its block cipher, following the techniques introduced by BBQ [35]. Picnic signatures are based on an MPC-in-the-head proof of knowledge of a secret key k such that $\text{AES}_k(x) = y$, where the corresponding public key is (x, y). As we show in §5.1, our protocol returns the smallest proof size among streaming- and memory-friendly systems using less than 32 emulated MPC parties. Our signature sizes are also competitive with those of Banquet [7], an independent recent work that involves a memory-intensive polynomial interpolation over the entire circuit.

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 subsets I of inputs to the circuit and outputs of MUL gates. $O \subseteq W$ denotes the output wires for the circuit. 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.

We say a party is p.p.t. to denote that it is probabilistic polynomial time.

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 4. 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 $\mathcal{F}$: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, length-preserving, and keyed function. $\mathcal{F}$ is a pseudorandom function with soundness κ if for all adversaries A that run in at most q time steps, A's advantage $Adv_{\mathsf{PRF}}(\mathsf{A}) = |\Pr_k[\mathsf{A}^{\mathcal{F}(k,-)} = 1] - \Pr_H[\mathsf{A}^H = 1]|$ at distinguishing the pseudorandom function from a random oracle H is at most $q/2^\kappa$.*

Definition 2 (Commitment). *A commitment scheme is a protocol between two parties $\mathcal{S}$ and $\mathcal{R}$ with the following algorithms:*

- $\mathsf{Com}(m)$*: 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.*
- $\mathsf{Decom}(c, m, r)$*: the sender $\mathcal{S}$ sends (c, m, r) to the receiver $\mathcal{R}$, who then either accepts and outputs m or rejects.*

A computationally secure commitment scheme satisfies the following properties:

- ***Completeness:*** *If $(c, r) = \mathsf{Com}(m)$, then in $\mathsf{Decom}(c, m, r)$ the receiver $\mathcal{R}$ accepts and outputs m.*
- *(Computational)* ***Hiding:*** *For any two message pairs $m, m' \in \{0,1\}^*$, any receiver $\mathcal{R}^*$ running in q time cannot distinguish their respective commitments $Adv_{\mathsf{Com}}(\mathcal{R}^*) = |\Pr[\mathcal{R}^*(\mathsf{Com}(m, r)) = 1] - \Pr[\mathcal{R}^*(\mathsf{Com}(m', r')) = 1]|$ except with probability at most $q/2^\kappa$.*
- *(Computational)* ***Binding:*** *No adversarial sender $\mathcal{S}^*$ running in at most q time has more than probability $q/2^\kappa$ of outputting c, m, m', r, r' such that $m \neq m'$, and $\mathsf{Decom}(c, m, r)$ and $\mathsf{Decom}(c, m', r')$ both accept.*

While our main construction can support arbitrary commitment schemes, in this work we focus on the hash-based commitment scheme in the random oracle model, in which $\mathsf{Com}(m; r) = H(m, r)$ feeds the input message and randommess into the random oracle and $\mathsf{Decom}(c, m, r) = (m, r)$ provides the preimage to the hash. The binding of this scheme follows from a birthday bound analysis: if a random oracle has 2κ bit output length and an adversary makes at most q queries to this oracle, then the probability that the adversary finds a collision in the oracle is at most $q^2/2^{2\kappa}$, and a collision is necessary to break the binding property of the commitment scheme. The hiding property can be proved similarly.

There are a few optimizations that prior works have used here to save space. First, when committing to a list of messages $\langle m_1, m_2, \ldots, m_\ell \rangle$, the sender can provide a succinct commitment $H(\mathsf{Com}(m_1, r_1), \ldots, \mathsf{Com}(m_\ell, r_\ell))$ to the entire list, again thanks to collision resistance. Second, if m is already known to the receiver, then it suffices to send only r during decommitment. Third and most ambitiously, because we will only commit to strings that already have min-entropy κ, when generating a signature scheme we can go further and remove the randomness r from the Com and Decom algorithms to create a deterministic scheme in which decommitments are free. This strategy breaks the hiding property of the commitment and thus the zero knowledge property of the schemes we will construct, but it will suffice for our signature construction; we refer readers to Katz et al. [56, §3.1] for details.

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 the same 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 $\mathcal{P}$ and $\mathcal{V}$ are honest and $R(x, w) = 1$, $\mathcal{V}$ always accepts.*
- ***Soundness:*** *For any malicious and computationally bounded prover $\mathcal{P}^*$, there is a negligible function $\mathsf{negl}(\cdot)$ 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 $\geq 1 - \mathsf{negl}(|x|)$ 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 Sim such that for all x in the language, $\mathsf{Sim}(x) \approx_c \mathsf{View}_{\mathcal{V}(x,w)}$.*

In this work, we will construct a ZK argument of knowledge, which provides a stronger *knowledge soundness* guarantee that if a bounded-time malicious prover $\mathcal{P}^*$ can make the verifier accept a statement x with non-negligible probability, then there exists an extractor $E^{\mathcal{P}^*}(x)$ that can output a witness w such that the relation holds $R(x,w) = 1$.

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

Secure 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. For use in MPC-in-the-head, secure computation protocols must satisfy t-privacy, meaning that the view of any subset $t < N$ of the parties can be simulated (see [52] for a formal definition).

3 Construction

We present our protocol $\Pi_{\text{TurboIKOS}}$ in this section and in Figure 1. We start by describing the Baum-Nof [6] SPDZ-like protocol and the Turbospeedz MPC protocol. Then, we show how to incorporate Turbospeedz [9] 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) [52] combined with a semi-honest version of the $(N-1)$-private SPDZ MPC protocol [33] as a starting point for our zero-knowledge proof using MPC-in-the-head protocol. In IKOS, a prover simulates an 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 views of both those parties. 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 with fresh randomness, the verifier can shrink the probability of error by a power of R.

We start with the variant of semi-honest SPDZ [33] used by Baum-Nof [6]. 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 $\varepsilon_m \leftarrow \mathbb{F}$. Each party does the following:

1. Broadcast $[f_m] = \varepsilon_m[x_m] + [a_m]$ and $[g_m] = [y_m] + [b_m]$
2. Use the recombined f and g to compute

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

Baum and Nof show that 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 $1 - 1/|\mathbb{F}|$ over the choice of ε_m. We will prove similar claims in Lemmas 1-2.

To save proof space, rather than broadcasting the ζ_m values for each MUL gate m, an additional challenge variable $\hat{\varepsilon}_m \in \mathbb{F}$ is sent by the verifier $\mathcal{V}$ and the prover $\mathcal{P}$ responds by sending a linear combination $[Z] = \sum_{m \in M} \hat{\varepsilon}_m [\zeta_m]$ of the secret values and public coefficients. If the prover is honest then $Z = 0$. Baum and Nof show (Proposition 1 of [6]) that Z will be nonzero if at least one mult gate constraint is violated with probability at least $1 - 2/|\mathbb{F}|$. Later in Lemma 2 of this pape we will improve this bound for a batched set of ζ_m values to a $1/|\mathbb{F}|$ error using a very-slightly different batching technique.

If $1/|\mathbb{F}|$ does not yield sufficient soundness error, we can reduce this error by doing multiple batched checks. To do so, we reveal linear combinations $[Z_1], \ldots, [Z_U]$, all over the same $[\zeta_m]$ shares, but using different random $\hat{\varepsilon}_m$ choices provided by the verifier. Let U be the number of these checks.

Naively, this protocol broadcasts $(2M + U)N$ elements since each party broadcasts their f shares and g shares for each multiplication gate, plus $[Z]$. Later in this work we will show multiple ways to reduce this size with different tradeoffs, by taking advantage of the fact that $\mathcal{V}$ has corrupted $N - 1$ parties. We emphasize that all parties' shares must still be committed to before $\mathcal{P}$ knows which party will remain uncorrupted.

To compress the parties' views, 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 offsets of 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 [9] 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, $\mathcal{P}$ 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).

Fig. 1: Beginning of an interactive zero knowledge protocol 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. There are two different endings to this protocol, given in Figures 2 and 4.

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 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^*. Pseudorandomly 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 each input wire $w \in I$ and party $p \in N$, pseudorandomly derive share $[\lambda_w]$ from 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 $\hat{e}_z \coloneqq \lambda_x \cdot \hat{\lambda}_{y,g} + \hat{\lambda}_z$, which creates 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 $w \in W$, and assigns the offset $e_w \coloneqq v_w + \lambda_w$ for each wire $w \in W$. It 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 then be computed.)
   - A commitment to all shares $[\lambda_w]$ for all output wires $w \in O$.
   - A commitment to the seeds σ_p for all parties $p \in N$.
2. $(\mathcal{V} \to \mathcal{P})$ $\mathcal{V}$ randomly selects two elements $\varepsilon_m, \hat{\varepsilon}_m \leftarrow \mathbb{F}$ for every MUL gate $m \in M$.

[There are two ways to complete this protocol, shown in Figures 2 and 4.]

3.3 Adapting Turbospeedz into sacrificing-based MPC-in-the-head

In this section, we incorporate a modified version of the Turbospeedz method from Section 3.2 into the SPDZ-based MPC-in-the-head framework described above from Section 3.1. For large field sizes, the resulting MPC-in-the-head protocol $\Pi_{\text{TurboIKOS}}$ will require only 3 field elements per multiplication, rather than the 4 elements used in Baum-Nof [6].

Committing to all wire values. The first step of converting Turbospeedz into an MPC-in-the-head protocol is to replace the step of opening shares of e_z by having the $\mathcal{P}$ simply provide e_z in the clear. However, unlike Turbospeedz, we do not wish to have a costly preprocessing process in which the verifier becomes convinced of the validity of all Beaver triples in the circuit; instead we wish to use ζ_m values as in Eq. 1. In order to do this without reusing masks on multiple values, we must make some subtle changes to the original Turbospeedz protocol.

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 $[\hat{\lambda}_{y,m}]$ and $[\hat{\lambda}_z]$. $\mathcal{P}$ computes $\hat{e}_z = \lambda_x \hat{\lambda}_{y,m} + \hat{\lambda}_z$, forming

Fig. 2: End of the 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 $\mathbb{F}$. See Figure 1 for the beginning of this protocol.

Interactive phase, continued from Fig. 1:

3. ($\mathcal{P} \to \mathcal{V}$) $\mathcal{P}$ sends all α_m values and commits to $\mathcal{P}$ commits to all $[\alpha_m]$ and $[Z]$ shares. These variables are computed as follows.
   - For every MUL gate $m \in M$, assign $[\alpha_m] \coloneqq \varepsilon_m[\lambda_y] + \hat{\varepsilon}_m[\hat{\lambda}_{y,m}]$.
   - For every party $p \in N$, assign $[Z] \coloneqq \sum_{m \in M}[\zeta_m]$, which is the sum of all $[\zeta_m] \coloneqq \varepsilon_m e_z - \varepsilon_m e_x e_y + \hat{\varepsilon}_m \hat{e}_z + (\varepsilon_m e_y - \alpha_m)[\lambda_x] + \varepsilon_m e_x[\lambda_y] - \varepsilon_m[\lambda_z] - \hat{\varepsilon}_m[\hat{\lambda}_z]$.
4. ($\mathcal{V} \to \mathcal{P}$) $\mathcal{V}$ randomly selects a set $T = N \setminus \{i^*\}$ of $N-1$ parties to corrupt.
5. ($\mathcal{P} \to \mathcal{V}$) $\mathcal{P}$ reveals the $\log(N)$ seeds from preprocessing step 1 that suffice for $\mathcal{V}$ to recompute σ_p for all corrupted parties $p \in T$, but not the remaining seed σ_{i^*}.

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

- The output values (v_w for $w \in O$) provided by $\mathcal{P}$ correspond to logical true.
- The commitments in rounds 1 and 3 are consistent with the opened keys σ_p for corrupted parties, and with the shares of $[Z]$, $[\alpha_m]$, and $[\lambda_w]$ for output wires for *all* parties. $\mathcal{V}$ can compute $N-1$ shares of these from its seeds, and the remaining shares from the revealed α_m values and the known values for Z and output wires.

a correlated Beaver triple $\langle \lambda_x, \hat{\lambda}_{y,m}, \hat{e}_z - \hat{\lambda}_z \rangle$ that will be sacrificed. The double index on $[\hat{\lambda}_{y,m}]$ is due to the fact that wire y may be reused in several different mult gates m, each of which must define their own Beaver triple for the prover's privacy. (For legibility, we sometimes omit this double-subscript when the gate under consideration is clear from context.)

Creating a test for consistency of all gates. The largest change is in how ζ_m is calculated for each MUL gate m. To save space, we check the consistency of all of these values with one random linear combination of Eq. (1) for all MUL gates. We begin similarly to the Baum-Nof challenge: $\mathcal{V}$ will send random challenges $\varepsilon_m, \hat{\varepsilon}_m \leftarrow \mathbb{F}$. Our α_m values are defined slightly differently, for a reason we will explain shortly. The prover will send $\alpha_m = \varepsilon_m \lambda_y + \hat{\varepsilon}_m \hat{\lambda}_y$. Then, the parties compute:

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

First, we wish to show that this ζ_m serves a similar purpose to Baum-Nof's, assuming (for the moment) that the prover $\mathcal{P}$ honestly computes all α_m values from the parties' shares. For each MUL gate $m \in M$, define:

$$\Delta_{z,m} = (e_z - \lambda_z) - (e_x - \lambda_x)(e_y - \lambda_y) \text{ and } \hat{\Delta}_{z,m} = (\hat{e}_z - \hat{\lambda}_z) - \lambda_x \hat{\lambda}_y.$$

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 $\Delta_{z,m} = \hat{\Delta}_{z,m} = 0$.

Lemma 1. *Fix a MUL gate $m \in M$. If ε_m and $\hat{\varepsilon}_m$ are chosen uniformly randomly from $\mathbb{F}$, and if either $\Delta_{z,m} \neq 0$ or $\hat{\Delta}_{z,m} \neq 0$ (or both), then $\zeta_m \neq 0$ with probability at least $1 - 1/|\mathbb{F}|$.*

Proof. Observe that

$$\begin{aligned}
\zeta_m &= \varepsilon_m e_z - \varepsilon_m e_x e_y + \hat{\varepsilon}_m \hat{e}_z + (\varepsilon_m e_y - \alpha_m)\lambda_x + \varepsilon_m e_x \lambda_y - \varepsilon_m \lambda_z - \hat{\varepsilon}_m \hat{\lambda}_z \\
&= \varepsilon_m e_z - \varepsilon_m e_x e_y + \hat{\varepsilon}_m (\lambda_x \hat{\lambda}_y + \hat{\Delta}_{z,m}) + (\varepsilon_m e_y - \alpha_m)\lambda_x + \varepsilon_m e_x \lambda_y - \varepsilon_m \lambda_z \\
&= \varepsilon_m e_z - \varepsilon_m e_x e_y + \hat{\varepsilon}_m \hat{\Delta}_{z,m} + (\varepsilon_m e_y - \varepsilon_m \lambda_y)\lambda_x + \varepsilon_m e_x \lambda_y - \varepsilon_m \lambda_z \\
&= \varepsilon_m e_z - \varepsilon_m \lambda_z - \varepsilon_m e_x e_y + \varepsilon_m e_x \lambda_y + \varepsilon_m e_y \lambda_x - \varepsilon_m \lambda_y \lambda_x + \hat{\varepsilon}_m \hat{\Delta}_{z,m} \\
&= \varepsilon_m ((e_z - \lambda_z) - (e_x - \lambda_x)(e_y - \lambda_y)) + \hat{\varepsilon}_m \hat{\Delta}_{z,m} \\
&= \varepsilon_m \Delta_{z,m} + \hat{\varepsilon}_m \hat{\Delta}_{z,m}.
\end{aligned}$$

Now, consider the probability that $\zeta_m = 0$ over the uniform choice of ε_m and $\hat{\varepsilon}_m$ from $\mathbb{F}$. The only way for this to occur is if $\varepsilon_m \Delta_{z,m} = -\hat{\varepsilon}_m \hat{\Delta}_{z,m}$. If $\Delta_{z,m} = 0$, this happens if and only if $\hat{\varepsilon}_m = 0$, which occurs with probability $1/|\mathbb{F}|$. If $\Delta_{z,m} \neq 0$, then for any choice of $\hat{\varepsilon}_m$ there exists a single option for $\varepsilon_m = -\hat{\varepsilon}_m \hat{\Delta}_{z,m} \Delta_{z,m}^{-1}$ that makes $\zeta_m = 0$, so again we arrive at a probability of $1/|\mathbb{F}|$. These two cases are mutually exclusive, which yields the desired bound.

Similar to Baum-Nof, we can combine these in a linear combination Z to test all multiplication gates at once. However, because we defined ζ_m to already include two different random coefficients on the different Δ values, these coefficients already suffice to serve as challenge coefficients for this linear combination. As we show in Lemma 2, the upshot is that we can test all gates in the circuit with a soundness error of only $1/|\mathbb{F}|$ by merely revealing $[Z] = \sum_{m \in M} [\zeta_m]$.

Lemma 2. *If ε_m and $\hat{\varepsilon}_m$ are chosen uniformly randomly from $\mathbb{F}$ for all multiplication gates in the circuit, and if there exists at least one* MUL *gate $\bar{m} \in M$ such that $\Delta_{z,m} \neq 0$ or $\hat{\Delta}_{z,m} \neq 0$, then $Z \neq 0$ with probability at least $1 - 1/|\mathbb{F}|$.*

Proof. Consider $Z = \sum_{m \in M} \zeta_m = (\varepsilon_{\bar{m}} \Delta_{z,\bar{m}} + \hat{\varepsilon}_{\bar{m}} \hat{\Delta}_{z,\bar{m}}) + Z'$, where Z' is the sum of all other terms in the formula and $\bar{m} \in M$ is the gate where the sum is guaranteed to be nonzero; without loss of generality, suppose that $\Delta_{z,\bar{m}} \neq 0$. Then, $Z = 0$ if and only if $\varepsilon_{\bar{m}} = \Delta_{z,\bar{m}}^{-1} \cdot (-Z' - \hat{\varepsilon}_{\bar{m}} \hat{\Delta}_{z,\bar{m}})$, which occurs with probability $1/|\mathbb{F}|$.

Completing the consistency test. Rounds 3-5 of the protocol provide a method for the verifier to check whether $Z = 0$, up to $1/N$ soundness error. We will describe two ways to perform this task: a base protocol $\Pi_{\text{TurboIKOS}}$ described in this section (shown also in Figure 2) and an improved protocol $\tilde{\Pi}_{\text{TurboIKOS}}$ in Section 3.4. Both techniques involve providing some 'advice' in the form of the non-privacy-sensitive α_m value for each MUL gate that assists the verifier in its computation of Z.

In round 3 of the base protocol $\Pi_{\text{TurboIKOS}}$ in this section, the prover provides for each MUL gate $m \in M$. Importantly, the prover also commits to all shares $[\alpha_m] = \varepsilon_m [\lambda_y] + \hat{\varepsilon}_m [\hat{\lambda}_y]$, and analogously for all $[Z]$ shares. There are three claims that the verifier must check:

- The committed $[\alpha_m]$ and $[Z]$ are consistent with the parties' individual views, at least for the $N - 1$ emulated parties that the verifier can open.
- The committed $[\alpha_m]$ shares in round 3 collectively sum to the provided α_m value. That is, the prover provided the public α_m 'advice' value correctly.
- Assuming the advice is correct, then the $[Z]$ shares committed in round 3 sum to $Z = 0$. That is, the prover passes the test posed in Lemma 2.

After the prover reveals seeds for $N - 1$ parties in round 5, the verifier can check these claims as follows. First, $\mathcal{V}$ can compute the remaining party's $[\alpha_m]$ by subtracting the known shares from the public α_m value, and then check whether these shares together constitute a valid opening of the commitment in round 3. This checks (most of) the first two claims simultaneously. The final claim is verified similarly; the key observation here is that if the prover is honest, then the value $Z = 0$ is publicly known. So, the $\mathcal{V}$ computes $N - 1$ shares of Z, calculates what the remaining party's share must be in order for the overall value $Z = 0$ as required, and then checks whether these shares together constitute a valid opening of the commitment.

Putting it all together. Our protocol is described in detail in Figure 1. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ interact in a 5-round protocol, and if all consistency checks pass then the verifier believes that the output wire labels derived from the circuit evaluation is correct.

Completeness is a straightforward consequence of the fact that the honest prover computes the desired circuit (many times, in fact). We prove the privacy and knowledge soundness of our ZK argument of knowledge in Section 4.

Compared to Baum-Nof [6], we reduce communication per multiplication gate from 4 to 3 field elements. Concretely, for each multiplication gate, Baum-Nof must send the f and g values 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 MUL gate (similar to e_z). By using the Turbospeedz approach, we reduce communication to only 3 field elements: e_z, $\hat{e}_z$, and α_m.

Algorithmic optimizations. There are a few optimizations that we can apply to the base protocol $\Pi_{\text{TurboIKOS}}$ to save space even further. Some of these optimizations are deliberately omitted from Figures 1 and 2 for brevity; they are simple to add, and they are built into our implementation in §5.

Our first optimization saves on the cost of commitments. Recall that we need to commit to values in each of the prover steps (rounds 1, 3, and 5), and also that the entire procedure from Figs. 1-2 is repeated R times. It suffices to build a single commitment per round across all repetitions: that is, just 2 commitments in total for the entire proof.

Second, we described the SPDZ-style MPC protocol by considering pseudorandom values for each party plus a public offset. Following prior works, we save space by integrating the offset into a single party's value (say, party 1). While this party no longer has pseudorandom value, the upshot is that we only need to reveal the $\hat{e}_z$ values within party 1's view, or in other words we don't need to reveal these values for the $1/N$ fraction of repetitions in which party 1 is the unopened party. (Note that we still need to publish the e_z and α_m values on all repetitions because $\mathcal{V}$ needs this information to perform its consistency check.)

Third, if the circuit has a single known value that represents 'logical true' (say, the value 0), then we can save on the cost of opening the output wire shares $[\lambda_w]$ for all parties (i.e., including the unopened party). Instead, we can follow a similar trick as we described above for $[\alpha_m]$ and $[Z]$. In round 1 of the protocol, the prover $\mathcal{P}$ commits to all output wire shares. Once the verifier $\mathcal{V}$ learns the seeds to reconstruct $N-1$ of these shares for itself, it assumes that the output wires collectively reconstruct to logical true and calculates the remaining share accordingly. Finally, $\mathcal{V}$ checks that all shares match $\mathcal{P}$'s commitment.

3.4 Constructing smaller consistency tests

In this section, we describe an improved protocol $\tilde{\Pi}_{\text{TurboIKOS}}$ that reduces the cost per multiplication gate from 3 field elements down to 2. Specifically, we show a new method to check the consistency of Z in rounds 3-5 without revealing an α_m element for each MUL gate. The motivation for this change is twofold. The first reason is obvious: reducing the number of field gates required per MUL gate shrinks the proof size. The second and more subtle reason is to improve the performance of TurboIKOS on smaller fields, such as the field GF(256) used in AES, without blowing up the size of the protocol with additional zero-checks.

We will explain this second motivation at a high level here; for more detail see the soundness analysis (Theorems 2 and 3) in §4. A cheating prover must have a sufficiently low chance of getting a set of coefficients $\varepsilon_m, \hat{\varepsilon}_m$ where $Z = 0$ even though at least one MUL constraint is violated. For small fields, a malicious prover has a decently-high probability of passing a single zero-test Z by pure chance. While one could overcome this issue by increasing the number of repetitions R, there is an alternative solution: run multiple Z values with fresh random ε_m and $\hat{\varepsilon}_m$ coefficients for each, but the same wire shares λ_z and offsets e_z and $\hat{e}_z$.

This alternative method doesn't fare well in the original protocol $\Pi_{\text{TurboIKOS}}$ because our method requires revealing an α_m value per MUL gate for each test, so each additional Z value used to improve the soundness error would reveal an additional field element per MUL gate, making the proof size much larger. Thus, our goal in this section is to show how to consistency-check multiple zero-tests $Z_1, \ldots, Z_U$ without transmitting additional information proportional to the number of MUL gates beyond the two field elements e_z and $\hat{e}_z$ we already transmitted in round 1.

Recall that for a single MUL gate m,

$$[\zeta_m] = \varepsilon_m e_z - \varepsilon_m e_y e_y + \hat{\varepsilon}_m \hat{e}_z + \varepsilon_m e_y [\lambda_x] - \alpha_m [\lambda_x] + \varepsilon_m e_x [\lambda_y] - \varepsilon_m [\lambda_z] - \hat{\varepsilon}_m [\hat{\lambda}_z]$$

is a sharing of $\zeta_m = \varepsilon_m \Delta_{z,m} + \hat{\varepsilon}_m \hat{\Delta}_{z,m} = 0$ for an honest prover. Observe that each party can calculate most of the terms in this sum even without receiving α_m. Specifically, for each MUL gate, define $\phi_m \coloneqq \zeta_m + \alpha_m \lambda_x$. Each party has the information to compute its own share $[\phi_m]$ using information already available: the public ε_m and $\hat{\varepsilon}_m$ values, the known offsets e, and the corrupted shares $[\lambda]$.

Also, recall from the original protocol that the parties never test each $[\zeta_m]$ directly, but rather they only test that the sum $Z = \sum_{m \in M} \zeta_m$ equals zero. We can rewrite the shares of this test in the following way. (We add a subscript m to the wire values to make it unambiguous which gate each wire belongs to.)

$$[Z] = \sum_{m \in M} [\zeta_m] = \sum_{m \in M} [\phi_m] - \sum_{m \in M} \alpha_m [\lambda_{m,x}].$$

The corrupted parties can compute their shares for the left sum. However, the sharing of the remaining term, which we will name

$$\beta = \sum_{m \in M} \alpha_m \lambda_{m,x} \tag{2}$$

is problematic because it seems to require each α_m to be known in the clear to calculate the sum. This is where the original protocol $\Pi_{\text{TurboIKOS}}$ revealed all α_m, so that each party could compute their share of β as $\sum_{m \in M} \alpha_m [\lambda_{m,x}]$.

We proceed to show a different way that the prover can commit to and provide the shares for $[\beta]$, which we also describe pictorially in Fig. 3. Let $[x]_i$ denote the ith share of x. The crucial observation is that all shares $[\alpha_m]$ can be revealed without a loss in privacy; our only objective here is a performance improvement to avoid sending these shares, even though we could safely do so. Furthermore, observe that:

$$\begin{aligned}
\beta &= \sum_{m=1}^{M} \alpha_m \lambda_{m,x} = \sum_{m=1}^{M} \left(\sum_{i=1}^{N} [\alpha_m]_i \right) \left(\sum_{j=1}^{N} [\lambda_{m,x}]_j \right) = \sum_{m=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} [\alpha_m]_i [\lambda_{m,x}]_j \\
&= \sum_{j=1}^{N} \sum_{i=1}^{N} \beta_{i,j}, \text{ where } \beta_{i,j} = \sum_{m=1}^{M} [\alpha_m]_i [\lambda_{m,x}]_j
\end{aligned} \tag{3}$$

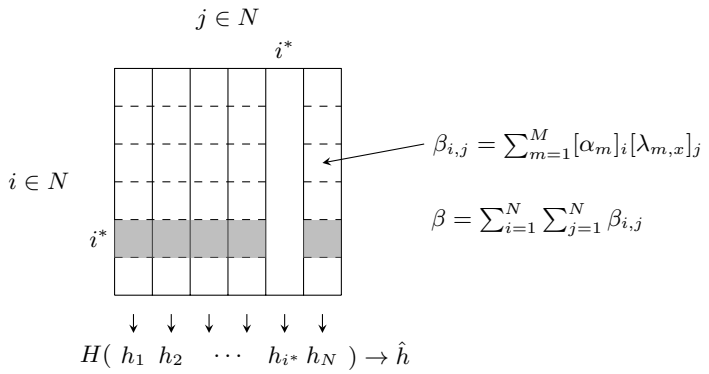
We will take advantage of the MPC-in-the-head structure of our proof to create a sharing where each party essentially holds a "column" of these values: that is, party j's share of β is $\sum_{i=1}^{N} \beta_{i,j}$. In a normal MPC protocol, each pair of parties i and j could collaborate to compute $\beta_{i,j}$. In MPC-in-the-head this is mostly unnecessary, because $\mathcal{V}$ has corrupted $N-1$ of the parties, it has $N-1$ of these $[\alpha_m]_i$ shares and therefore can compute $\beta_{i,j}$ for all corrupted parties $i, j \in T$. It is missing only $\beta_{i^*,j}$, where i^* is the remaining, uncorrupted party.

In Fig. 3, the parties' shares of β are the sum of the elements in each column. $\mathcal{V}$ has $N-1$ elements (the unshaded regions in each column) out of the N elements needed to compute the full share (column sum), but needs to be sent the remaining (shaded) element to sum the column and compute the share.

For soundness, we require that all $\beta_{i,j}$ be committed to in advance. To do this, we concatenate all elements in each column and commit to them; call these commitments $h_1, \ldots, h_N$. Each commitment h_j must be randomized since the field elements might be small enough not to provide the required min-entropy on their own, but this can be done "for free" by using party j's seed to generate this random value. We then commit to the concatenation of those commitments and give the result $\hat{h}$ to $\mathcal{V}$ in round 3, before the corrupted parties are chosen.

To check the commitments, $\mathcal{V}$ checks two properties: First, the commitment to the $\beta_{i,j}$ values must be consistent with the party views, and second, the shares of Z must sum to 0. To show the first property, $\mathcal{V}$ first recreates all $(N-1)^2$ elements it can from the parties it corrupted. It then receives $(N-1)$ missing elements

Fig. 3: Creating a more efficient sharing of β (Eq. 2). Each cell holds a single $\beta_{i,j}$ value (Eq. 3). A single commitment $\hat{h}$ binds the entire matrix. To open $N-1$ of the columns, $\mathcal{P}$ gives $\mathcal{V}$ the field elements in the shaded region along with h_{i^*}.



(the shaded region in Fig. 3) from $\mathcal{P}$, which lets it compute the shares $[\beta]_j$ for the parties it corrupted and also the commitment h_j to the concatenation of the elements in each column. $\mathcal{V}$ does not get the missing share $[\beta]_{i^*}$; instead, it is given h_{i^*}, the missing commitment. Using all these, it can check $\hat{h}$ to ensure the corrupted party views are consistent with the commitment to the $\beta_{i,j}$.

Second, as in the other version of this protocol, the shares to Z are also committed to, but where the β component of Z is shared using this method. $\mathcal{V}$ checks that $Z = 0$ by recomputing the $N-1$ shares it corrupted, and then computes the last share by subtracting those shares from 0. The recomputed shares are checked against the commitment to the $[Z]$ shares from round 3.

This portion of the proof sends $(N-1)$ field elements (the shaded elements) and one commitment (h_{i^*}) per repetition; the additional commitments to the shares of Z and the $\hat{h}$ need only use one commitment for the entire proof, across all repetitions. It bears repeating that this method was able to check the consistency of all multiplication gates without revealing an additional α_m value per MUL gate. This allows us to add additional fresh zero-tests independently of the number of MUL gates as well. If there are U of these tests, our communication per repetition becomes about $(2M + UN)$ field elements, which can outperform Katz et al. [56] when the number of parties N is small. The description of this version of the protocol is given in Fig. 4.

4 Security analysis

In this section we prove the honest-verifier zero-knowledge and knowledge soundness properties of the two protocols constructed in §3. For each property, we first analyze the base protocol $\Pi_{\text{TurboIKOS}}$, and then we describe how the analysis changes for the improved protocol $\tilde{\Pi}_{\text{TurboIKOS}}$.

Theorem 1. *When instantiated with a pseudorandom function PRF and a computationally hiding commitment scheme Com, both the base protocol $\Pi_{TurboIKOS}$ and improved protocol $\tilde{\Pi}_{TurboIKOS}$ 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 focus here on the privacy argument for the base protocol $\Pi_{\text{TurboIKOS}}$. The privacy of the improved protocol $\tilde{\Pi}_{\text{TurboIKOS}}$ then follows immediately from the fact that it provides strictly less information to the verifier than the base protocol $\Pi_{\text{TurboIKOS}}$ does. Additionally, we prove the statement for a single repetition of the base protocol $\Pi_{\text{TurboIKOS}}$, from which the theorem follows by a union bound over the independent repetitions.

Fig. 4: Ending of the improved zero knowledge protocol $\tilde{\Pi}_{\text{TurboIKOS}}$ 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 $\mathbb{F}$. See Figure 1 for the beginning of this protocol. Compared to Fig. 2, step 3 is new, and this change affects the opening and checking of commitments in step 5 and verification; the remainder of the protocol is unchanged from before. If multiple zero tests are desired, then steps 2, 3, and 5 are repeated with independent ε_m and $\hat{\varepsilon}_m$, Z, and $\beta_{i,j}$ values.

Interactive phase, continued from Fig. 1:

3. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ commits to all $\beta_{i,j}$ by sending $\hat{h}$ and commits to all $[Z]$ shares. These variables are computed as follows.
   - For $i \in N$, for $j \in N$, $\beta_{i,j} = \sum_{m=1}^{M} [\alpha_m]_i [\lambda_{m,x}]_j$.
   - For every party $j \in N$, assign $[Z]_j = \sum_{m \in M} \left(\varepsilon_m e_{m,z} - \varepsilon_m e_{m,y} e_{m,y} + \hat{\varepsilon}_m \hat{e}_z + \varepsilon_m e_y [\lambda_{m,x}] + \varepsilon_m e_{m,x} [\lambda_{m,y}] - \varepsilon_m [\lambda_{m,z}] - \hat{\varepsilon}_m [\hat{\lambda}_{m,z}]\right) - \sum_{i \in N} \beta_{i,j}$.
4. $(\mathcal{V} \to \mathcal{P})$ As before, $\mathcal{V}$ samples a set $T = N \setminus \{i^*\}$ of $N-1$ parties to corrupt.
5. $(\mathcal{P} \to \mathcal{V})$ $\mathcal{P}$ opens commitments to $\mathcal{V}$ in the following way:
   - As before, reveal $\log(N)$ seeds from preprocessing step 1 that suffice for $\mathcal{V}$ to recompute σ_p for all corrupted parties $p \in T$, but not the remaining seed σ_{i^*}.
   - Reveal $\beta_{i^*,j}$ for all $j \neq i^*$ and reveal h_{i^*}.

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 commitments in rounds 1 and 3 are consistent with the opened keys σ_p for corrupted parties, with the $\beta_{i,j}$ opened, and with the shares of $[Z]$ and $[\lambda_w]$ for output wires for *all* parties. $\mathcal{V}$ computes these using the party seeds, the remaining $\beta_{i^*,j}$ values it is given, and the known Z and output values.

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 samples all verifier challenges uniformly at random: all $\varepsilon_m, \hat{\varepsilon}_m$ in round 2, and a party $i^* \in N$ in round 4 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 likely to be non-zero. Using the foreknowledge of ε_m and $\hat{\varepsilon}_m$, alter party i^*'s share of $\hat{\lambda}_z$ so that $\zeta = \varepsilon_m \Delta_z + \hat{\varepsilon}_m \hat{\Delta}_z$ equals 0 (or alter λ_z if $\hat{\Delta}_z = 0$ but $\Delta_z \neq 0$).

6. In round 1, commit honestly to all parties' keys σ_p and all parties' shares $[\lambda_z]$ for all output wires $z \in O$. Also, send the offsets e_z for all $z \in I \cup M'$.
7. In round 3, commit to all shares $[\alpha_m]$ and $[Z]$, where $[Z]$ is computed correctly from the already-manipulated ζ shares from above.
8. In round 5, honestly open the key commitments for all parties except i^*. Also, honestly reveal party i^*'s shares $[Z]$, $[\alpha_m]$, and $[\lambda_w]$ for output wires.

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 now useless, 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^*'s share of $\hat{\lambda}_y$.
- 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 the hybrid and simulated games is 0, which completes the proof for the base protocol $\Pi_{\mathrm{TurboIKOS}}$.

Next, we examine the soundness of the base protocol $\Pi_{\mathrm{TurboIKOS}}$. We focus on its security for the non-interactive version of the MPC-in-the-head construction using the Fiat-Shamir transform using a random oracle H with 2κ bits of output, so that finding a collision has 2^κ cost. (The interactive version of the protocol has even better soundness because the prover cannot rewind the verifier.)

Theorem 2. *Consider the non-interactive version of the base protocol $\Pi_{TurboIKOS}$ over a large field $|\mathbb{F}| = 2^\kappa$ and instantiated with a random oracle H with 2κ output length. Then, Protocol $\Pi_{TurboIKOS}$ with $R = \frac{\kappa}{\log_2(N)} + 1$ repetitions provides knowledge soundness with error at most $1/2^\kappa$.*

Proof. We focus here on proving the traditional soundness property. The stronger knowledge soundness claim immediately follows by applying the analysis from Katz et al. [56, §3.1] in order to build an extractor that recovers a witness by observing the inputs to the random oracle-based commitment scheme on a single execution. The reduction is tight because the extractor never needs to rewind.

To prove soundness, consider a malicious prover $\mathcal{P}^*$ that is attempting to prove a false statement. Since an honest execution of the circuit would return logical false, the prover must deviate from the protocol on each repetition. There are effectively four different places where the malicious prover $\mathcal{P}^*$ can deviate from the protocol in order to gain an advantage:

1. In round 1 of the protocol, $\mathcal{P}^*$ can change the offsets for one or more MUL gates, so that the Δ_z or $\hat{\Delta}_z$ values on these gate(s) are non-zero. (We presume it is simple for the prover to determine a sufficient set of gates to tamper in order to cause the circuit to return logical true.) $\mathcal{P}^*$ will learn in round 2 whether $\mathcal{V}$ catches this deviation or whether $\mathcal{P}^*$ has successfully evaded detection. By Lemma 2, the prover is successful with probability $1/|\mathbb{F}|$.
2. In round 3, $\mathcal{P}^*$ can change one party's $[\alpha_m]$ and $[Z]$ shares so that the verifier reconstructs a Z value of 0. The success of this attack is revealed in round 4, and $\mathcal{P}^*$ evades detection with probability $1/N$.

3. In rounds 1 or 3, $\mathcal{P}^*$ can attempt to break the binding property of the commitment scheme and open it later to different values.
4. In round 1, $\mathcal{P}^*$ can change one party's share of the final output so the result becomes logical true. $\mathcal{P}^*$ will learn in round 4 whether $\mathcal{V}$ catches this deviation or whether $\mathcal{P}^*$ has successfully evaded detection (this occurs with probability $1/N$).

(Observe that the pseudorandom function has no impact on soundness. It only exists to 'compress' each party's share of each wire label $[\lambda_w]$, and any tampering of these wire labels is equivalent to tampering the corresponding offset e_w.)

The key observation in this proof is that item 1 is strictly better for $\mathcal{P}^*$ than item 3 and that item 2 is strictly better than item 4. The first part of this claim follows from the observation that items 1 and 3 both require the prover to deviate in round 1 and the first has better probability of success since the commitment scheme has soundness κ. The second part of this claim is true because the attacks in items 4 and 2 both give the same probability of success and are both revealed in round 4, yet the alteration of $[\alpha_m]$ and $[Z]$ occurs later. Delaying the start of the attack is strictly better for $\mathcal{P}^*$ because it can wait to see if the attack on wire offsets in round 1 was successful, and only attempt this attack if necessary.

Concretely, we use the same proof technique as several recent analyses of non-interactive zero knowledge proofs that apply the Fiat-Shamir transform to protocols with more than three rounds [7,16,55]. We consider all *attacker strategies* (r_1, r_2) in which the prover $\mathcal{P}^*$ changes wire offsets in round 1 until r_1 repetitions happen to have $Z = 0$ anyway, and then $\mathcal{P}^*$ attacks the remaining $r_2 = R - r_1$ repetitions in round 3 by altering α_m and Z. In general, the cost of any multi-round attack strategy is given by $C = 1/p_1 + 1/p_2$, where p_1 and p_2 denote the probability that the first and second parts of the attack succeed, respectively. To achieve κ soundness, we must choose a sufficiently large number of repetitions R so that any attacker strategy $(r_1, R - r_1)$ has a total cost $C \geq 2^\kappa$.

In this case, one attacker strategy dominates the rest: $r_1 = 1$ and $r_2 = R-1$. In more detail, the malicious prover $\mathcal{P}^*$ changes the wire offsets for all repetitions in round 1, and rewinds until finding an input that evades detection from the verifier's round 2 challenge on $r_1 = 1$ instance. Because each instance evades detection only with probability $1/|\mathbb{F}| = 1/2^\kappa$, the malicious prover $\mathcal{P}^*$ can only expect to evade detection on $r_1 = 1$ instance in time less than 2^κ. Even succeeding on 2 instances is exceedingly unlikely in the adversary's runtime, and the $\mathcal{P}^*$ gains no benefit by foregoing an attack on round 1 altogether. Thereafter, $\mathcal{P}^*$ must complete the attack on the remaining $r_2 = R - 1$ repetitions by altering $[\alpha_m]$ and $[Z]$. This change is undetected by the verifier with probability $1/N$ independently for each repetition. Hence, the overall probability of success for the second repetition is $(1/N)^{r_2}$, and thus its cost exceeds 2^κ when $R = \frac{\kappa}{\log_2(N)} + 1$.

Theorem 3. *Consider the improved protocol $\tilde{\Pi}_{TurboIKOS}$ that is instantiated with a random oracle H with 2κ output length and executed over the field $\mathbb{F}$ with N parties, R repetitions, and U tests per repetition. Then, the protocol satisfies knowledge soundness with security parameter κ if:*

$$\max_{0 \leq r_1 \leq R} \left\{ \left[p_1 = \sum_{i=r_1}^{R} \binom{R}{i} \cdot \left[\frac{1}{|\mathbb{F}|}\right]^{Ui} \cdot \left[1 - \frac{1}{|\mathbb{F}|}\right]^{U(R-i)} \right]^{-1} + N^{(R-r_1)} \right\} > 2^\kappa. \tag{4}$$

Proof. Once again, we focus on proving traditional soundness, after which we can build an extractor for knowledge soundness using the same technique as before. Additionally, the analysis from Theorem 2 about the options for a malicious prover $\mathcal{P}^*$ to deviate from the protocol holds here too. The prover's dominant strategy remains to change the offsets for one or more MUL gates in a way that causes the remainder of the circuit (computed honestly) to return logical true; note that this will also cause the corresponding Δ_z or $\hat{\Delta}_z$ values to be nonzero. For each repetition independently, there are two ways for the malicious prover $\mathcal{P}^*$ to evade detection by the verifier:

1. Based on the verifier's U independent random choices of ε_m and $\hat{\varepsilon}_m$ in round 2, there is a $(1/|\mathbb{F}|)^U$ probability that all of the Z tests happen to equal 0 (by Lemma 2).

Table 2: Valid parameter settings for $\mathbb{F}_{2^8}$. The body of the table shows the number of repetitions R based on the soundness parameter κ, number of parties N, and number of tests per repetition U.

κ	N	U				
		1	2	4	6	9
128	8	66	53	47	45	44
	16	54	41	36	34	33
	31	47	35	30	28	27
192	8	99	79	70	68	66
	16	81	62	54	52	50
	31	71	53	45	43	41
256	8	132	106	95	91	89
	16	108	83	73	69	67
	31	95	71	61	57	55

2. If even a single Z value is non-zero, then $\mathcal{P}^*$ can commit to erroneous $\beta_{i,j}$ values to 'fix' this error. Note that $\mathcal{P}^*$ can only inject erroneous data in one row of the table in Fig. 3 because the verifier can check the remaining $N-1$ rows directly. This attack evades detection only if the verifier chooses in round 4 to leave this row as the uncorrupted party, which happens with probability $1/N$.

As before, we can analyze the malicious prover $\mathcal{P}^*$'s probability of success by analyzing all attacker strategies (r_1, r_2) that operate as follows. First, the prover $\mathcal{P}^*$ rewinds round 1 until at least r_1 repetitions have the property that the verifier's choice of ε_m and $\hat{\varepsilon}_m$ are such that *all* of the Z tests within these repetitions equal 0. Second, $\mathcal{P}^*$ rewinds round 3 until the remaining $r_2 = R - r_1$ have been tampered in the locations chosen by the verifier in round 4.

To achieve κ soundness, we must select enough repetitions R so that any attacker strategy $(r_1, R - r_1)$ has a total cost $C \geq 2^\kappa$. Here, the *cost* of this multi-round attack strategy is given by $C = 1/p_1 + 1/p_2$, and p_1 and p_2 denote the probability that the first and second parts of the attack succeed, respectively.

Because we consider an arbitrary field size in this theorem, the cost analysis here is more complicated than in Theorem 2. Our argument is very similar to that of Banquet [7]. The first probability boils down to the chance that the attacker has r_1 successes out of R trials, where each trial succeeds with probability $(1/|\mathbb{F}|)^U$. This is the right tail of a binomial distribution:

$$p_1 = \sum_{i=r_1}^{R} \binom{R}{i} \cdot (1/|\mathbb{F}|)^{Ui} \cdot (1 - 1/|\mathbb{F}|)^{U(R-i)}. \tag{5}$$

The second probability is simply $p_2 = (1/N)^{r_2}$ because the malicious prover must succeed on all remaining repetitions of the protocol. Combining the costs of both parts of the attack results in Eq. (4), completing the proof of the theorem.

The goal here is to find the "minimum" choices of N, R, and U that yield a desired soundness parameter κ for a circuit with a given field size $|\mathbb{F}|$. While it is challenging to write a closed-form version of Eq. 4 that connects the five parameters, it is easy to find satisfying tuples empirically. In Table 2, we show several such choices for the field $\mathbb{F}_{2^8}$. We include a computer program that calculates valid parameter settings within our open source repository [50].

5 Performance and Prototype Implementation

In this section, we compare our $\tilde{\Pi}_{\text{TurboIKOS}}$ system's performance against other MPC-in-the-head based systems, and we describe our prototype Python implementation of $\Pi_{\text{TurboIKOS}}$.

Table 3: Signature size comparison for the Picnic signature scheme at different security levels for different systems. All signature sizes are shown in kilobytes (KB). AES-128 has $(M, I) = (200, 128)$, AES-192x2 has $(M, I) = (416, 192)$, and AES-256x2 has $(M, I) = (500, 256)$.

Scheme	**N**	**Protocol** *(all sizes in KB)*				
		BBQ [35]	Katz et al. [56]	Baum-Nof [6]	$\tilde{\Pi}_{\text{TurboIKOS}}$	Banquet [7]
AES-128 (L1)	8	|	26.7	37.3	23.8	–
	16	31.6	22.4	29.4	20.6	19.8
	31	|	19.8	24.8	19.8	17.5
AES-192x2 (L3)	8	|	76.3	110.8	66.7	–
	16	86.9	63.1	87.4	56.3	51.2
	31	|	54.9	73.4	52.2	45.1
AES-256x2 (L5)	8	|	122.6	179.9	109.3	–
	16	133.7	101.9	140.7	90.4	83.5
	31	|	89.1	118.0	82.8	73.9

5.1 Performance Analysis

To evaluate our performance, we measure our signature size when computing a variant of the Picnic signature scheme [63]. Picnic uses MPC-in-the-head (specifically, a variant of the Katz et al. protocol [56]) and LowMC [2], a block cipher with few multiplications that is designed to be efficient in secure computation. Picnic is currently an "alternate candidate" in round 3 of the NIST post-quantum crypto competition [1]. BBQ [35] introduced the idea of using AES in Picnic-like signatures and showed that the signature sizes could be competitive with those using LowMC. To achieve this, rather than evaluating the binary circuit for AES, they used an arithmetic MPC-in-the-head system over $\mathbb{F}_{2^8}$; this facilitates proving constraints about inverses in $\mathbb{F}_{2^8}$, the non-linear component of the AES S-box. They also show how each field inversion can be reduced to a single multiplication gate (without testing for the case in which the input and output are both equal to 0) with very small reduction in the soundness of the resulting system (less than 3 bits of security). We follow their approach in this section.

Table 3 shows the proof sizes of $\tilde{\Pi}_{\text{TurboIKOS}}$ when computing signatures using AES at different security levels. We compare against the following systems:

- BBQ [35]: The figures are taken directly from their paper, except the number of parties is not listed. We make an educated guess that they use approximately $N = 64$ parties; for lower N their proof size will be larger.
- Katz et al. [56]: We calculate the proof size using the formula in [56], assuming that a field inversion constraint can be verified with two field elements per MUL gate like in this work; we believe this should be true but did not check. Also, our calculations use 32 parties rather than 31; this result should be a conservative smaller estimate of the actual proof size when $N = 31$.
- Baum-Nof [6]: We calculate a conservative underestimate of the proof size using the formula in [6], assuming that only a single zero-check is needed per repetition. However, this is unlikely to be the case in $\mathbb{F}_{2^8}$ for the same reason as in our work.
- Banquet [7]: The figures are taken directly from their paper.

Banquet [7] is independent recent work that reduce the size of AES-based Picnic signatures using very different techniques based on polynomial interpolation in extension fields of $\mathbb{F}_{2^8}$ [20, 31]. They achieve a smaller proof size than all competitors, including us. However, this advantage comes with two downsides relative to our scheme and prior ones. First, performing polynomial arithmetic in extension fields would be costly on embedded devices whose CPU architectures typically have a small word size. Second, the polynomials are proportional to the entire circuit size, making their system more memory-costly than traditional MPC-in-the-head based methods like ours where the memory is only proportional to the circuit width (i.e., the memory required to compute the circuit).

5.2 Prototype Implementation

We created a prototype implementation for $\Pi_{\text{TurboIKOS}}$ in Python. We did not optimize the runtime or memory usage of our code, thus, we will likely not win on prover runtime with other MPC-in-the-head approaches written in C or C++, e.g. [6, 56]. That having been said, our protocol is amenable to all of the optimizations made by the recent Reverie software [71] implementing the protocol of Katz et al. Our code currently achieves runtimes about twice as long as Reverie for the same size circuit. In this section we briefly describe our implementation.

Our Python implementation [50] supports both Bristol circuit formats and Prover Worksheet (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 are used extensively to manage information. Circuit information such as the number of various types of gates are stored in a dictionary. Values on each wire such as e and λ are also stored in a dictionary. Each wire has 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.

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 $\mathcal{P}$. We generate the shares λ, $\hat{\lambda}_y$, and $\hat{\lambda}_z$ pseudorandomly using AES as a PRF with the party's seed as the key and the concatenation of the (fixed-length) wire index and type of value as the message.

The prover is required to send a commitment in Round 1 and Round 3. As discussed in §2, when committing to values with sufficient min-entropy, we simply use $\mathsf{Com}(m) := H(m)$, thus decommitments are "free" aside from the cost of m itself. This remains computationally hiding as long as m has sufficient min-entropy and H is a random oracle. We use $\mathsf{SHA2}$ from `hashlib` for our instantiation of H. To achieve non-interactivity via the Fiat-Shamir Transform [37], the random messages sent by the Verifier in Round 2 and Round 4 are replaced by a call to H on the info sent by $\mathcal{P}$ in all previous rounds.

Acknowledgments

This material is supported by a Google PhD Fellowship, the DARPA SIEVE program under Agreement No. HR00112020021, and the National Science Foundation under Grants No. 1414119, 1718135, 1739000, 1801564, 1915763, and 1931714.

References

1. Alagic, G. et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. https://csrc.nist.gov/publications/detail/nistir/8309/final (2020)
2. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (Apr 2015)
3. 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)
4. 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)
5. 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
6. 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)

7. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. In: Public Key Cryptography. Lecture Notes in Computer Science, Springer (2021)
8. 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)
9. 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)
10. 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)
11. 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)
12. 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)
13. 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)
14. 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)
15. 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)
16. Beullens, W., de Saint Guilhem, C.: LegRoast: Efficient post-quantum signatures from the Legendre PRF. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 130–150. Springer, Heidelberg (2020)
17. 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)
18. 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)
19. 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)
20. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 67–97. Springer, Heidelberg (Aug 2019)
21. 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)
22. 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)
23. 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)
24. 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)
25. 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)
26. 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)
27. 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)
28. 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)

29. 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
30. 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)
31. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: NSDI. pp. 259–282. USENIX Association (2017)
32. 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)
33. 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)
34. 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)
35. de Saint Guilhem, C., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: Using AES in picnic signatures. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 669–692. Springer, Heidelberg (Aug 2019)
36. 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)
37. 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)
38. 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)
39. 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
40. 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
41. 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)
42. 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)
43. 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)
44. 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)
45. 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)
46. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing **18**(1), 186–208 (1989)
47. 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)
48. 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)
49. 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)
50. Gvili, Y., Ha, J., Varia, S.S.M., Yang, Z., Zhang, X.: TurboIKOS. https://github.com/sarahscheffler/TurboIKOS (2021)
51. 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)
52. 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)

53. 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)
54. 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)
55. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 3–22. Springer, Heidelberg (Dec 2020)
56. 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)
57. 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)
58. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC. pp. 723–732. ACM Press (May 1992)
59. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (May 2007)
60. 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)
61. 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)
62. 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)
63. Microsoft Corporation: Picnic. https://microsoft.github.io/Picnic/
64. 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)
65. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: 31st ACM STOC. pp. 245–254. ACM Press (May 1999)
66. 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)
67. 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)
68. 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)
69. 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)
70. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020), https://eprint.iacr.org/2020/1275
71. Trail of Bits: Reverie. https://github.com/trailofbits/reverie (2021)
72. 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)
73. 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 (2020)
74. 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)
75. 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)
76. 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)