Limbo: Efficient Zero-knowledge MPCitH-based Arguments

Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy

imec-COSIC, KU Leuven, Leuven, Belgium.
cyprien.delpechdesaintguilhem,emmanuela.orsini,titouan.tanguy@kuleuven.be

Abstract. This work introduces a new interactive oracle proof system based on the MPC-in-the-Head paradigm. To improve concrete efficiency and offer flexibility between computation time and communication size, a generic proof construction based on multi-round MPC protocols is proposed, instantiated with a specific protocol and implemented and compared to similar proof systems.
Performance gains over previous work derive from a multi-party multiplication check optimized for the multi-round and MPC-in-the-Head settings. Of most interest among implementation optimizations is the use of identical randomness across repeated MPC protocol executions in order to accelerate computation without excessive cost to the soundness error.
The new system creates proofs of SHA-256 pre-images of 43KB in 53ms with 16 MPC parties, or 23KB in 188ms for 128 parties. As a signature scheme, the non-interactive variant produces signatures, based on the AES-128 circuit, of 18KB in about 4ms; this is 20% faster and 32 % larger than the Picnic3 scheme (13kB in 5.3ms for 16 parties) which is based on the 90% smaller LowMC circuit.

1 Introduction

A zero-knowledge (ZK) proof is a cryptographic tool that allows a prover to convince a verifier that a statement is true without leaking any information to the verifier other than the validity of the assertion. Since their introduction by Goldwasser, Micali and Rackoff [GMR85] in the 1980s, ZK proofs have become a fundamental tool for both cryptography theory and, more recently, practical systems thanks to real-world applications such as distributed ledger technology and cryptocurrencies.

There have been many developments in the construction of highly efficient zero-knowledge systems in recent years, each of which offers different trade-offs between several efficiency measures such as the number of interactions between prover and verifier (in particular distinguishing interactive and non-interactive systems), communication complexity, proof length, and prover and verifier computation complexity.

A common and useful way to simplify protocol construction in such a large design space is to proceed in a modular way: first construct an information-theoretic protocol (also called a *probabilistically checkable proof* or PCP) which makes use of idealised assumptions, and then compile it to a 'real' world protocol, or more formally an *argument* system [BCC88], using cryptographic tools. This approach is used for example to construct *succint non-interactive arguments* [Kil92,Mic94,Kil95,BFM88,BBC$^+$17,BBHR19,BCGT13]. Here, the term succinct usually refers to systems with sub-linear proof size, but can additionally refer to efficient verification. The extension of PCPs to interactive PCPs (IPCPs) [KR08] allows more interaction between prover and verifier after the proof generation; the recent further extension to interactive oracle proofs (IOPs) [BCS16,RRR16], which are effectively "multi-round PCPs", achieves even better efficiency than standard PCPs. Other related variants include linear PCPs [BCI$^+$13] and their generalization to fully linear PCPs and IOPs [BBC$^+$19]. In particular, linear PCPs have been used to build sub-linear arguments with preprocessing, with very efficient instantiations [GGPR13]. The main drawbacks of this approach usually include prover complexity, heavy use of public-key machinery and requirement for trusted setup.

More generally, due to the modular approach, it is possible to combine different information-theoretic proof systems with different cryptographic tools to obtain systems with very different characteristics, especially in term of efficiency.

In 2007, Ishai, Kushilevitz, Ostrovsky and Sahai [IKOS07] introduced a very powerful paradigm to build (honest-verifier) ZKPCPs using secure multi-party computation (MPC), known as MPC-in-the-Head (MPCitH). Recent efficient solutions for circuit satisfiability based on this approach include ZKBoo [GMO16], KKW [KKW18], BN [BN20] and Ligero [AHIV17,BFH+20]. A common feature of these schemes is that the prover's complexity is linear in the circuit size and their overall concrete efficiency which makes these schemes very competitive, even for relatively large statements. In particular, among MPCitH-based systems, KKW offers the best concrete computational performance, while Ligero notably achieves sub-linear communication complexity and hence shortest proof lengths for large enough circuits.

Interestingly, the MPCitH approach has been successfully used to construct very efficient digital signature schemes with post-quantum security, such as Picnic [CDG+17,KKW18,ZCD+20].

1.1 Our Contributions and Techniques

Motivated by the simplicity and flexibility of the MPCitH paradigm, in addition to the good concrete performance of systems based on it, we construct Limbo, a new zero-knowledge MPCitH-based argument for circuit satisfiability which works for both Boolean and arithmetic circuits.

Our construction offers linear communication and prover complexity, however our focus is on concrete performance rather than asymptotic complexity, and Limbo achieves extremely good efficiency, both in terms of prover complexity and proof length. As common to all MPCitH-based systems, it also achieves transparency (no need for trusted setup) and post-quantum security.

Concretely, our scheme offers computational performance comparable with KKW, but with significantly shorter proofs, achieving the best overall performance among MPCitH-based schemes for medium size circuits (i.e. with less than 500000 multiplication gates). For larger circuits, Ligero has shorter proofs but is computationally more expensive than our protocol.

We also use the non-interactive variant of our construction (NILimbo) to design a post-quantum signature scheme, in line with previous works such as Picnic [CDG+17], BBQ [dDOS19], LegRoast [Bd20] and Banquet [BdSGK+21].

We furthermore provide an implementation of our protocols and compare its performance with other MPCitH-based systems. We now detail our contributions and techniques.

MPCitH zk-IOP. We extend the general MPCitH zk-IOP constructions defined in Ligero and Ligero++ [AHIV17,BFH+20], which in turn can be seen as an optimized version of the black-box transformation introduced in IKOS [IKOS07], to work with MPC protocols with arbitrary number of rounds. This allows for more freedom in the choice of the MPC components and hence for zero-knowledge systems with different efficiency features.

After this, we instantiate this general system with a very simple MPC protocol with low communication complexity in order to minimize both the proof length and prover complexity.

A common way to design concretely efficient MPCitH protocols is to instantiate the MPC component with efficient MPC building blocks, as done in KKW [KKW18] and BN [BN20]. While this approach can be seen as the most natural, it may not hold that efficient MPC protocols lead to the most efficient MPCitH counterpart. Instead, we use a protocol that is specifically designed to fit in the MPCitH framework, i.e. with a single party knowing all inputs and with minimal communication complexity. At a high level, we define a protocol with only one computing party, where the role of the remaining parties is only to check that the circuit evaluation was done correctly. Note that the underlying MPC component used in the Ligero family [AHIV17,BFH+20] respects this same model, but with very different security guarantees and "checking method". While the goal of [AHIV17,BFH+20] was to achieve succinctness (still with competitive running times), we aim to have a better concrete balance between proof size and prover complexity. When we compile our zero-knowledge proof system to interactive and non-interactive zero-knowledge arguments, we obtain better overall performance than previous related work for small and medium size circuits.

Post-quantum Signature Schemes. We use our zero-knowledge argument protocol to describe a Picnic-like post-quantum signature scheme. Picnic is one of the alternate candidates in Round Three of the NIST Post-quantum Standardization process and, as proved in a recent work by Cremers et al. [CDF+20], is the one (together with CRYSTALS-Dilithium [LDK+20]) offering the strongest security guarantees among the six finalists. Picnic uses an MPCitH zero-knowledge protocol to prove knowledge of a secret key $\mathtt{k}$ such that $F_{\mathtt{k}}(x) = y$, where $F_{\mathtt{k}}$ is a one-way function. In practice, Picnic uses LowMC [ARS+15] as the underlying OWF, hence basing its security on non-standard assumptions. Replacing LowMC with a more standard cipher, such as AES, increases the proof size significantly. The BBQ protocol [dDOS19] shows how to reduce this overhead when AES is used instead of LowMC, using the same underlying MPC protocol as in Picnic [KKW18], but basing the computation on $\mathbb{F}_{2^8}$ rather than $\mathbb{F}_2$, i.e. focusing on S-boxes rather than individual AND gates. BBQ signatures are however still at least two times larger than Picnic ones. The more recent proposal of Banquet further reduces this gap using an underlying MPC protocol similar to the one used in this work [BdSGK+21].

Assuming an additive secret-sharing scheme, our protocol first has an input and evaluation phase where a single sender party performs the actual computation of the circuit, "injecting" the values needed to evaluate non-linear gates to the remaining server parties, after distributing the shares of the inputs. Given those values, all the server parties can then perform a local evaluation of the circuit on their own shares to compute their shares of the circuit output. After this phase, the server parties check that the injected values are correct, i.e. that the circuit has been correctly evaluated. This phase also requires injected values from the sender party and does not require any communication between the server parties, but only access to a random coin functionality. The check protocol that we use is an adaptation from [BBC+19,BGIN19,GS20] and concretely allows to test whether multiplication gates were correctly evaluated by checking the correctness of the corresponding multiplicative triples. Roughly speaking, the main difference between our MPC protocol and the one used in Banquet is in the way the correctness of the multiplication gates is tested.

Overall, we achieve better running times compared to Banquet [BdSGK+21] and comparable signature size. More importantly, our generalized approach offers a framework for MPCitH signature schemes that could hopefully lead to new improvements to Picnic-like signatures with different instantiations of the main building blocks.

Optimizations. It is common practice to reduce the soundness error of a zero-knowledge proof by repeating, either in parallel or sequentially, the protocol a certain number of times. However,

Our SHA 256	n	**Reps.**	**Prover (ms)** 1 thread	4 threads	**Verifier (ms)** 1 thread	4 threads	**Communication (bytes)**
	16	11	53	25	47	21.1	42229
	32	9	77	39	71	35	34604
	64	7	113	50	104	44	26971
	128	6	188	92	178	82	23157

Table 1: Performance of our system for proving knowledge of a SHA-256 pre-image with soundness 2^{-40} for various number of parties n. Reps stands for the number of required repetitions.

this approach significantly increases the complexity of the system both computationally and in communication. Instead, we improve the soundness of our general interactive construction by running the underlying MPC evaluation protocol multiple times in parallel and then checking these evaluations using the same public coin functionality shared across all of them. We then apply this general technique to our protocol. This approach allows for implementation optimizations and better concrete performance. In particular, this improves prover time by roughly 7–10% compared to naïvely repeating the protocol.

We can use this technique to also improve the performance in the multi-instance case. In Appendix A, we sketch different options to deal with this case efficiently.

Going beyond the gate-by-gate approach. We explore elements which enable our protocol to move beyond the gate-by-gate paradigm. Already in the application to AES-based signatures, similarly to BBQ and Banquet, our protocol considers S-box operations as the unit of computation (1 inverse over $\mathbb{F}_{2^8}$; rather than 32 AND gates over $\mathbb{F}_2$). Taking this approach allows for greater improvements than only improving binary circuits at the AND-gate level.

In Section 7 we further continue in this direction by adapting our protocol to the verification of inner products and matrix multiplications. Considering these larger operations again allows for specific optimizations to be made which provide significant improvements over their gate-by-gate implementation. Using this approach we can prove multiplication of two 256×256 matrices in 20sec (resp. 11 sec) with one thread (resp. 4 threads); this requires only 340KB of total communication: a 38x improvement compared to the naïve approach which would require 256^3 AND gates.

Concrete Efficiency. We present a detailed concrete analysis of both the communication and the computational cost of our protocols, and measure the concrete efficiency of our construction and compare it with other MPCitH-based systems.

Both our interactive and non-interactive variants work for arithmetic and Boolean circuits, however, since the checking phase of our protocol requires a large field, our construction is inherently more efficient when used for arithmetic circuits over such large finite fields. Nevertheless, to better compare our protocols with systems such as KKW, and use it for post-quantum signatures, we run most of our tests over very small fields, namely $\mathbb{F}_2$ and $\mathbb{F}_{2^8}$.

Our system depends on many parameters and is very flexible; we can trade communication for computation in a significant way by changing the number of parties in the MPC protocol, the extension field or other settings in the checking phase.

Concretely, to verify one instance of SHA-256 preimage, with 40 bits of security, our system requires 53ms for the prover and 43KB of communication when the number of parties is $n = 16$, and 188ms and 23 KB, respectively, when $n = 128$ (see Table 1). This represents a 3x improvement in computation time (with comparable communication) over the Ligero system (44KB of communication and 140ms of running time for the same circuit). Using 4 threads we further reduce prover computation time to 25ms.

We also compare the performance of our protocol with KKW and its recent highly optimized implementation, Reverie [MHA]. Although our current implementation is incomparable with Reverie, we show that our performance are already very close to those of Reverie. We plan to apply some of the techniques used in Reverie to improve our implementation in future works. Overall, we improve KKW in both proof size and run times.

Our implementation also shows that our signature scheme has signing/verification run times comparable with those of Picnic and signature size only 30% larger (for a 10x bigger circuit), assuming the same number of parties. We can reduce the signature size by running the protocol with larger number of parties at the cost of slower signing and verification. More details about our concrete measures can be found in Section 7.

Other related works. A recent line of work [WYKW20,BMRS20,DIO20,YSWW21], based on subfield vector oblivious linear evaluation (sVOLE), provides ZK proofs with very small memory footprint and extremely good efficiency. Our system, like other MPCitH protocols, allows streaming and can potentially achieve small memory overhead. Although we cannot accomplish the same performance of sVOLE-based protocols, our approach does not require an interactive preprocessing. Moreover, LPZK [DIO20] and Mac'n'Cheese [BMRS20] are currently designed only for large fields, whereas our protocol naturally works for both arithmetic and Boolean circuits.

2 Preliminaries

We denote by κ (resp. λ) the computational (resp. statistical) security parameter. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if, for every positive polynomial $p(\cdot)$ and all sufficiently large integer k, it holds that $\mu(k) < \frac{1}{p(k)}$. We also use the abbreviation PPT to denote probabilistic poly-nomial-time algorithms. We use bold letters to denote vectors, e.g. $\mathbf{a}$, and use brackets to denote entries, e.g. $(\mathbf{a})_i$; the operator $*$ denotes the inner product of two vectors. We denote by $[d]$ the set of integers $\{1, \ldots, d\}$, and by $[e, d]$ the set of integers $\{e, \ldots, d\}$ with $1 < e < d$.

MPC notation. The notation $\langle\cdot\rangle$ stands for additively secret-shared values with full threshold, and $\langle\cdot\rangle_i$ for the share held by party P_i.

Languages and relations. We denote by $\mathcal{R}$ a relation consisting of pairs (x, w), where x is the instance and w is the witness. We denote by $\mathcal{L}(\mathcal{R})$ the language corresponding to $\mathcal{R}$.

2.1 Zero-knowledge Arguments of Knowledge

An argument of knowledge for an NP relation $\mathcal{R}$ is a protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. We let $\mathsf{view}(\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle)$ denote the transcript generated by $\mathcal{P}$ and $\mathcal{V}$ when interacting on inputs (x, w) and x, respectively. Also, we say that $\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle = b \in \{0, 1\}$ depending on whether $\mathcal{V}$ accepts, $b = 1$, or rejects $b = 0$.

Definition 1. *The pair $(\mathcal{P}, \mathcal{V})$ is called an* argument of knowledge *for the relation $\mathcal{R}$ if the following properties are satisfied.*
Completeness: $\forall(x, w) \in \mathcal{R}$, $\langle\mathcal{P}(x, w), \mathcal{V}(x)\rangle = 1$.

SOUNDNESS: *For any PPT prover $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ such that, for any x, the probability*

$$\Pr[\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle = 1 \wedge (x,w) \notin \mathcal{R} \mid w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x)]$$

is negligible, where the extractor $\mathcal{E}^{\mathcal{P}^*}$ has access to the entire execution, including the randomness of $\mathcal{P}^*$.

Definition 2. *An argument of knowledge $(\mathcal{P}, \mathcal{V})$ is* public coin *if the verifier samples its messages uniformly at random and independently of the messages sent by $\mathcal{P}$. This is equivalent to say that $\mathcal{V}$'s messages correspond to $\mathcal{V}$'s randomness.*

Definition 3. *A public coin argument of knowledge is* (honest verifier) zero-knowledge *for a relation $\mathcal{R}$ if there exists a simulator $\mathcal{S}$ such that, for any $(x,w) \in \mathcal{R}$, the view of an honest verifier in the interaction $\langle \mathcal{P}, \mathcal{V}\rangle$ and the output of $\mathcal{S}$ are indistinguishable, i.e.*

$$\mathsf{view}(\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle) \approx \mathcal{S}^{\mathcal{V}}(x),$$

where $\mathcal{S}^{\mathcal{V}}$ denotes access to the public coin randomness used by the verifier.

2.2 Interactive Oracle Proofs

Interactive oracle proofs (IOPs) simultaneously extend probabilistic checkable proofs (PCPs) and interactive proofs (IPs) by allowing more rounds of interaction and using point-wise queries from the verifier to the oracles, instead of linear queries. IOPs also differ from IPCPs which can be viewed as special IOPs where the verifier has oracle access to the first prover messages, but must read in full subsequent prover's messages.

Definition 4. *A ρ-round public-coin IOP for the relation $\mathcal{R}$ consists of a ρ-round interactive protocol between $\mathcal{P}$ and $\mathcal{V}$, with $\rho \geq 2$, such that in each round $i \geq 2$, after an initial π_1 created by $\mathcal{P}$, the verifier $\mathcal{V}$ sends a uniformly random message v_{i-1} to $\mathcal{P}$ and the prover replies with π_i. The verifier has oracle access to $\pi = \{\pi_1, \ldots, \pi_\rho\}$ and $\mathcal{P}$'s last message in response to v_ρ and, based on the responses from the oracles, either accepts or rejects. It satisfies the following two properties:*
COMPLETENESS: *As in Definition 1.*
SOUNDNESS: *For all $x \notin \mathcal{L}$, and for all (computationally unbounded) $\mathcal{P}^*$*

$$\Pr[\langle \mathcal{P}(x,w), \mathcal{V}^{\pi}(x)\rangle = 1] \text{ is negligible.}$$

This definition can be extended with the knowledge and honest verifier zero-knowledge properties [BCS16]. Beyond soundness we can consider other complexity measures, in particular the *query complexity*, i.e. the number of queries asked by $\mathcal{V}$ to any of the oracles during the ρ rounds, and the *proof complexity*, i.e. the number of bits communicated during the interactions.

2.3 MPC-in-the-Head

In [IKOS07], Ishai, Kushilevitz, Ostrovsky and Sahai introduced the MPC-in-the-Head (MPCitH) paradigm that uses any MPC protocol with honest majority to construct a zero-knowledge proof for an arbitrary NP relation $\mathcal{R}$. The high level idea of this powerful technique is as follows. A zero-knowledge protocol can be viewed as an instance of secure function evaluation, and hence as two-party computation between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, with common input the statement x, and $\mathcal{P}$'s private input w, which is a witness to the assertion that x belongs to a given NP language $\mathcal{L}$. The function they want to compute is then $f_x(w) = \mathcal{R}(x, w)$, which checks if w is a valid witness or not. The verifier $\mathcal{V}$ will accept the proof if $f_x(w) = \mathcal{R}(x, w) = 1$.

In the MPCitH paradigm the zk-PCP prover $\mathcal{P}$ emulates an n-party MPC protocol Π in "its head": $\mathcal{P}$ generates a sharing $\langle w \rangle$ of the witness and distributes the corresponding shares as private inputs to the parties, and then simulates the evaluation of $f_x(\langle w \rangle) = \mathcal{R}(x, \langle w \rangle)$ by choosing uniformly random coins r_i for each party P_i, $i \in [n]$. Once the inputs and random coins are fixed, for each round j of communication of the protocol Π and for each party P_i, the messages sent by P_i at round j are deterministically specified as a function of the internal state of P_i, i.e. P_i's private inputs and randomness, and the messages that P_i received in previous rounds. The set with the state and all messages received by party P_i during the execution of the protocol constitutes the view of P_i, denoted as view_i.

After the evaluation, the prover sets $\pi = (\mathsf{view}_1, \ldots, \mathsf{view}_n)$ and sends it to an oracle $\mathcal{O}$. At this point, the verifier queries π on some points and finally verifies that the computation was done correctly by checking that the opened views are all consistent with each other and that the protocol outputs a positive result.

3 MPC-in-the-Head based IOP—General Construction

In this section we describe a general interactive proof system based on the MPC-in-the-Head paradigm which can be instantiated with different MPC protocols that respect a specific network topology. While IKOS [IKOS07] presents a general transformation of information-theoretic MPC protocols to a ZK proof in a "black-box" way, we follow Ligero's blueprint and precisely define the MPC model we use to build our system. We extend the general proof system defined in [AHIV17] by allowing arbitrary number of rounds. Then, we instantiate the MPC component with a different and yet very simple protocol which will allow the verifier to open a bigger number of views (or, equivalently, to query the oracles at a larger number of points).

3.1 MPC Model

Here we describe the MPC model that can be used to implement our general interactive proof system. This model can in turn be instantiated with MPC protocols with different security properties, leading to systems with different soundness, communication and computational complexity.

First we recall the following basic definition.

Definition 5 (Correctness, privacy and robustness [IKOS07]). *Let Π_f be an MPC protocol for a functionality f.*

- *We say that the protocol Π_f realizes f with perfect (resp. statistical)* **correctness** *if for all inputs x, the probability that the output of some party is different from $f(x)$ is 0 (resp. negligible in λ), where the probability is over the random inputs of each party.*
- *Let $1 \leq t < n$, the protocol Π_f has* **t-privacy** *if it is correct and for all $I \subseteq [n]$ such that $|I| \leq t$, there exists a PPT algorithm $\mathcal{S}$ such that the joints views $(\mathsf{view}_I(x))$ of parties in I has the same distributions of $\mathcal{S}(I, x_I, f_I(x))$. We will talk of perfect, statistical or computational t-privacy accordingly.*

- *Let $0 \leq r < n$, the protocol $\mathit{\Pi}_f$ has perfect (resp. statistical) r-**robustness** if it is correct and for all $I \subseteq [n]$ such that $|I| \leq r$, even assuming that all the parties in I have been* adaptively *corrupted, if there does not exists any random input such that $f(x) = 1$, the probability that $\mathit{\Pi}_f$ outputs 1 and the views of honest parties are consistent is zero (resp. negligible in λ).*

We now describe our MPC model.

Definition 6 (Client-server ρ-phase protocol). *Let $\mathit{\Pi}_f$ be an MPC protocol for a functionality f. We say that $\mathit{\Pi}_f$ is in the* client-server *model if its parties can be divided into a distinguished "input (or sender) client" P_S, n "computation servers" $P_1, \ldots, P_n$, and (optionally) a distinguished "output (or receiver) client" P_R. Additionally, P_S receives the entire input x and only sends at most one message to each of the computation servers at the beginning of each phase,*[1] *and P_R only receives a single message from each of the servers at the end of the protocol. The servers can only communicate with each other via a broadcast.*

We then say that $\mathit{\Pi}_f$ is a client-server ρ-phase *protocol if the computation of the n servers can be divided into ρ consecutive phases each separated by the sampling of a public random string via a call to* RandomCoin *from all the servers.*

The following three stages summarise the execution of a client-server ρ-phase protocol.

1. In the first phase, the servers receive the input message $\mathbf{m}_1$ from P_S and start their local computation of the circuit. More precisely, $\mathbf{m}_1$ is a vector of messages, where each server gets one entry of the vector.
2. For each phase $j \in [2, \rho - 1]$:
   (a) The servers call RandomCoin and obtain a public random string R_{j-1}, along with at most a single message $\mathbf{m}_j$ from P_S. Again, each party P_i only receives $(\langle \mathbf{m}_j \rangle)_i$, for $i \in [n]$.
   (b) The servers use the random string (and $\mathbf{m}_j$) to continue their local computation.
3. In phase ρ, the servers obtain a public random string R_ρ and each sends a single message to the receiver client P_R.

In our model we consider a *threshold (P_c, t_s)-adversary* which corrupts at most one client P_c, up to t_s servers, or both. In particular, we extend Definition 5 as follows.

Definition 7. *We say that a protocol $\mathit{\Pi}_f$ realizes f with $(P_c, t_{s,p})$-privacy (resp. $(P_c, t_{s,r})$-robustness) if the properties in Definition 5 hold with respect to a semi-honest (resp. adaptive malicious) adversary $\mathcal{A}$ that corrupts all the parties in $I = \{P_c\} \times I_s \subset \{P_S, P_R\} \times \{P_i\}_{i \in [n]}$, such that $|\{P_c\}| \leq 1$ and $|I_s| \leq t_{s,p}$ (resp. $t_{s,r}$).*

Note that this definition allows $(\emptyset, t_s)$-adversaries that only corrupt server parties.

3.2 Interactive Proof System - General Description

Given an MPC protocol $\mathit{\Pi}_f$ as described in Definition 6, we show a ρ-round interactive protocol, $\mathit{\Pi}_{\rho-\mathsf{ZKIOP}}$ (Figure 1), verifying the properties in Definition 4.

Let $\mathcal{L}(\mathcal{R})$ be an NP-language with relation $\mathcal{R}$, and let $f_x(w) = \mathcal{R}(x, w)$. Our ρ-round systems starts with the prover $\mathcal{P}$ emulating a ρ-round MPC protocol $\mathit{\Pi}_f$ (meeting Definition 6) that realizes

[1] Ligero allows only one message from P_S to the servers in the entire computation, i.e. it cannot send another message even after the public coin sampling that takes place between the two phases.

Protocol $\mathit{\Pi}_{\rho-\mathsf{ZKIOP}}$

Let $\mathit{\Pi}_f$ a ρ-phase MPC protocol in the client/server model. COMMON INPUT: A statement x and a circuit description C_f that realizes the relation $\mathcal{R}$.
PRIVATE INPUT: $\mathcal{P}$ holds the witness w such that $\mathcal{R}(x, w) = 1$
First Oracle π_1. $\mathcal{P}$ runs the MPC protocol $\mathit{\Pi}_f$ in its head: it samples a random $r_S, \{r_i\}_{i\in[n]} \in \{0,1\}^* \cup \emptyset$ and invoke the sender client P_S on input $(x, w; r_S)$. and the servers on random input r_i. The prover computes the views $(\mathsf{view}^1_1, \ldots, \mathsf{view}^1_n)$ of the servers in phase 1.
It sets the oracle $\pi_1 = (\mathsf{view}^1_1, \ldots, \mathsf{view}^1_n)$
Interactive protocol.

- For $j \in [2, \rho]$:
  - $\mathcal{V}$ picks a random challenge R_{j-1} and sends to $\mathcal{P}$
  - $\mathcal{P}$ continues to run the protocol in its head.
    It invokes the sender client and the servers on input R_{j-1} obtained in the previous step and produces views $(\mathsf{view}^j_1, \ldots, \mathsf{view}^j_n)$. It sets the oracle $\pi_j(R_{j-1}) = (\mathsf{view}^j_1, \ldots, \mathsf{view}^j_n)$.
- $\mathcal{V}$ picks a random challenge R_ρ and sends to $\mathcal{P}$.
- $\mathcal{P}$ computes and sends view_R of the receiver client
- $\mathcal{V}$ rejects if P_R outputs $C_f = 0$; if not, $\mathcal{V}$ asks to open a subset of server views. More precisely $\mathcal{V}$ picks random subsets $V_j \subset [n], j \in [\rho]$, such that $| \cup_j V| \leq t_{s,p}$.
- $\mathcal{P}$ open the views in $\cup_j V_j$
- Final verification: $\mathcal{V}$ aborts if the views in V_j are inconsistent with each other and/or with view_R, otherwise it accepts.

Figure 1: General description of the ρ-round IOP

the functionality f. As done in Ligero, we further restrict the MPC model and assume that the servers P_i never communicate with each other. The first round of $\mathit{\Pi}_{\rho-\mathsf{ZKIOP}}$ provides to an oracle $\mathcal{O}$ the string $\pi_1 = (\mathsf{view}^1_1, \ldots, \mathsf{view}^1_n)$, corresponding to the views of the n servers at the end of the first round. After this, we have the interactive steps, which exactly correspond to the rounds $[2, \rho]$ of the underlying MPC protocol, with the randomness obtained by the $\mathsf{RandomCoin}$ functionality being replaced by the verifier's challenges $R_1, \ldots, R_\rho$. We can pictorially represent the oracles $\pi_1, \ldots, \pi_\rho$ as a $\rho \times n$ matrix Q, where the rows are $Q_j = \pi_j, j \in [\rho]$, and the columns, $Q^i, i \in [n]$, correspond to the "global" view of the parties, i.e. $Q^i = \{\mathsf{view}^1_i, \ldots, \mathsf{view}^\rho_i\}$, for $i \in [n]$.

Note that if we instantiate $\mathit{\Pi}_{\rho-\mathsf{ZKIOP}}$ with $\rho = 1$, we obtain the system described in [AHIV17], which only allows one single message from P_S to the servers P_i, $i \in [n]$.

Restricting the model. We now specialize the MPC model $\mathit{\Pi}_f$ used in $\mathit{\Pi}_{\rho-\mathsf{ZKIOP}}$ with a protocol achieving $(P_R, n-1)$-privacy in the semi-honest model and $(P_S, 0)$-robustness in the malicious model. In particular, this latter property means that the MPC protocol does not allow any collusion between a malicious client sender and servers.

Moreover, we restrict $\mathcal{V}$'s queries (and hence also our IOP system) to the columns of the matrix Q, assuming that the verifier only opens up to $n-1$ of these "global" views.

If we now consider the security of our construction, it is very important to distinguish between the randomness used to ensure privacy and that used for robustness. The former is generated by $\mathcal{P}$ when it samples the randomness for the MPC parties. The latter is given by $\mathcal{V}$ and the crucial point is that each string generated in the middle of the protocol must be unpredictable for $\mathcal{P}$ during previous phases. Intuitively, the prover $\mathcal{P}$ can cheat either by "corrupting P_S" and computing the $\mathbf{m}_j$ messages wrongly, or by "corrupting" one or more of the servers $P_i, i \in [n]$, and computing their message to P_R wrongly.

More formally, we obtain the following result.

Theorem 1. *Let x be a public statement and w an additional input, let f be the functionality for $P_S, P_1, \ldots, P_n$ and P_R that outputs $\mathcal{R}(x, w)$ to P_R. Let Π_f be a ρ-phase MPC protocol in the client/server model that correctly realizes f with $(P_R, (n-1))$-privacy in the semi-honest model and $(P_S, 0)$-robustness (in the malicious model) with robustness error δ. The protocol $\Pi_{\rho-\mathsf{ZKIOP}}$ described in Fig. 1 is a ZKIOP for NP relation $\mathcal{R}$, with soundness error*

$$\epsilon = \frac{1}{n} + \delta\left(1 - \frac{1}{n}\right).$$

Proof: *(Completeness)* As Π_f correctly realizes f which outputs $\mathcal{R}(x, w)$ to P_R, every honestly computed proof for valid pair (x, w) will cause $\mathcal{V}$ to output accept.

(Honest verifier zero-knowledge) An honest verifier $\mathcal{V}$ will always choose the random challenges R_ρ according to the correct distribution thus emulating the calls to the functionality $\mathsf{RandomCoin}$ perfectly. Zero-knowledge of the proof then follows from the $(P_R, n-1)$-privacy of Π_f.

(Soundness) Let $\mathcal{P}^*$ be a malicious prover attempting to convince $\mathcal{V}$ of a false statement x^*. Considering that the output of $\mathcal{P}^*$ is exactly the views of the server parties $\{P_i\}$ and of the receiver client P_R, we study the cheating strategies of $\mathcal{P}^*$ in terms of corruption strategies against the MPC protocol. As P_R never communicates with the other parties, it cannot be corrupted against robustness. Furthermore, as we assume that $n-1$ server views are seen by the verifier, in addition to the view of P_R, the malicious prover can corrupt at most 1 server, i.e. cheat on the messages it sends to P_R, while retaining a non-zero chance of convincing $\mathcal{V}$. Therefore the possible strategies for P_S are to corrupt P_S, at most one server P_i, or both.

If $\mathcal{P}^*$ does not corrupt P_S, then the correctness of Π_f implies that the outputs of an honest execution will cause P_R, and therefore $\mathcal{V}$, to reject. In this case, if $\mathcal{P}^*$ corrupts one of the servers, then it breaks the $(\emptyset, 0)$-robustness of Π_f (implied by its $(P_S, 0)$-robustness), and can cause P_R to accept with certainty. The success probability of this strategy is then exactly $1/n$.

If $\mathcal{P}^*$ does corrupt P_S, then the $(P_S, 0)$-robustness of Π_f implies that P_R will accept with probability δ over the random outputs of $\mathsf{RandomCoin}$; if this happens, then no inconsistency between the servers and the receiver is visible, and $\mathcal{V}$ accepts the proof. With probability $1-\delta$, P_R rejects an honest execution and $\mathcal{P}^*$ must corrupt one of the servers to break the $(P_S, 0)$-robustness of Π_f which causes $\mathcal{V}$ to accept with probability $1/n$ as in the previous strategy. The final success probability here is therefore

$$\delta + (1-\delta) \cdot \frac{1}{n}, \tag{1}$$

which is the best of the two and therefore the maximum cheating probability for $\mathcal{P}^*$. Rearranging (1) yields ϵ. □

A very common solution to achieve the desired soundness in zero-knowledge systems, is to run the base protocol a certain number of times τ. Obviously, this approach increases the complexity of the system both computationally and in communication by a multiplicative factor τ. In the next section we describe a better strategy that allows to reach better soundness with less overhead.

3.3 Improving Soundness—More MPC Evaluations

We improve the soundness of the IOP construction of Figure 1 by having multiple sets of server parties execute the underlying MPC protocol in parallel. This improvement comes from the ability to open multiple sets of $n-1$ views to the verifier, each picked independently at random thus reducing the limiting $1/n$ term of Theorem 1.

By having the public randomness of RandomCoin shared across the executions, we limit the corruption strategies that are available against robustness. While independent challenges would possibly reduce the robustness error further, using identical ones also allows for implementation optimizations and we therefore establish a theoretical basis for this practice.

Definition 8 (τ-parallel execution). *Let Π_f be a client-server ρ-phase MPC protocol for a functionality f with n server parties. For an integer τ, Π_f^τ is the τ-fold parallel execution of Π_f as a client-server ρ-phase protocol where there is only one sender P_S, one receiver P_R, but τ independent sets of n server parties.*

The client parties P_S and P_R independently run an execution of Π_f with each set of servers who also do not communicate across sets, excepted for the calls to RandomCoin *which are shared across the τ executions; i.e. the $\tau \cdot n$ servers receive the same output from* RandomCoin*. If the τ executions output the same result, then P_R outputs the same; if any one of the executions dissents, P_R aborts the protocol.*

We first argue that privacy and robustness properties of the underlying protocol are maintained by the one run in parallel.

Proposition 1. *If Π_f is $(P_R, n-1)$-private in the semi-honest model, then Π_f^τ is $(P_R, \tau(n-1))$-private in the semi-honest model with the restriction that at most $n-1$ servers are corrupted for each of the τ executions.*

Proof: The $(P_R, n-1)$-privacy of Π_f implies the existence of a simulator $\mathcal{S}$ which, on input $(I, x_I, f_I(x))$, produces simulated views which are indistinguishable from the joint views $(\mathsf{view}_I(x))$ of an honest execution.

When receiving I^τ from the $(P_R, \tau(n-1))$-adversary against privacy, the simulator $\mathcal{S}^\tau$ invokes τ parallel copies of $\mathcal{S}$, using identical randomness for calls to RandomCoin but independent randomness otherwise. For the simulated view of P_R, $\mathcal{S}^\tau$ outputs the concatenation of the views of P_R produced by the parallel executions of $\mathcal{S}$. As the output of RandomCoin is public, this identical randomness does not leak private information and $\mathcal{S}^\tau$ simulates the τ independent executions with the same privacy error as $\mathcal{S}$. □

Proposition 2. *If Π_f is $(P_S, 0)$-robust in the malicious model with error δ, then Π_f^τ is $(P_S, 0)$-robust in the malicious model with error at most δ.*

Proof: As there does not exist any random input such that $f(x) = 1$ (Definition 5), and as the calls to RandomCoin are shared between the τ executions, the best strategy for a corrupt sender P_S^* is to deviate from the protocol in the same way across all executions. If not, the probability that a call to RandomCoin can satisfy two or more constraints created by a cheating sender can only be less than or equal to that of satisfying a single one. Therefore Π_f^τ is also $(P_S, 0)$-robust with error at most δ. □

We then argue that the IOP construction equivalent to that of Figure 1 using Π_f^τ instead of Π_f is also a ZKIOP with improved soundness error.

Theorem 2. *Let x be a public statement, and w an additional input, let f be the functionality for $P_S, P_1, \ldots, P_n, P_R$ that outputs $\mathcal{R}(x, w)$ to P_R. Let Π_f be a ρ-phase MPC protocol in the client-server model that correctly realizes f with $(P_R, (n-1))$-privacy in the semi-honest model and $(P_S, 0)$-robustness in the malicious model with robustness error δ.*

With Π_f^τ constructed from Π_f as in Definition 8, the protocol $\Pi_{\rho-\mathsf{ZKIOP}}$ as described in Figure 1 using Π_f^τ is a ZKIOP for $\mathcal{R}$ with soundness error

$$\epsilon = \frac{1}{n^\tau} + \delta\left(1 - \frac{1}{n^\tau}\right).$$

Proof: *(Completeness)* This follows from the completeness of Π_f and the construction of Π_f^τ.

(Honest verifier zero-knowledge) This follows from the $(P_R, \tau(n-1))$-privacy of Π_f^τ given by Proposition 1.

(Soundness) The same strategy for a malicious prover $\mathcal{P}^*$ applies as for the first protocol: by first corrupting only P_S, it has a probability of at most δ of causing Π_f^τ to output accept; if this fails, it can then corrupt at most one server for each of the τ independent executions to make P_R accept, this is not detected by $\mathcal{V}$ with probability $1/n^\tau$. □

4 Multiplications Check

In this section we describe an efficient MPC protocol in the client-server model for checking multiplication triples. This protocol is an adaptation of previous protocols described in recent works [BBC+19,BGIN19,GS20], and constitutes one of the main building block of our MPC component.

More concretely, the goal is for the server parties to verify the correctness of m multiplication tuples $\{x_\ell, y_\ell, z_\ell\}_{\ell\in[m]}$ given by the sender client; i.e. that $x_\ell \cdot y_\ell = z_\ell$, for each $\ell \in [m]$. We describe two different MPC checking protocols; the first, $\Pi_{\mathsf{MultCheck}}$, presents how to check multiplications using inner-products, the second, $\Pi_{\mathsf{CompressedMC}}$ extends this idea by repeating several compression rounds to reduce the communication between the servers and the recipient. While we do not prove the MPC security of these protocols, we present several properties which we will use in the next section.

4.1 First Multiplication Check Protocol

The first protocol, presented in Figure 2, checks the correctness of m secret-shared multuplication tuples by testing the correctness of a single secret-shared inner product tuple of size m.

It proceeds in two steps: first, given $\{\langle \mathbf{x}_i \rangle, \langle \mathbf{y}_i \rangle, \langle \mathbf{z}_i \rangle\}_{i\in[m]}$, the parties call a random coin functionality, $\mathsf{RandomCoin}$, to obtain a random value R in an extension field $\mathbb{G}$ of $\mathbb{F}$. Using R, the parties construct the inner-product tuple $\langle \mathbf{x} \rangle \in \mathbb{G}^m$, $\langle \mathbf{y} \rangle \in \mathbb{G}^m$, and $\langle z \rangle \in \mathbb{G}$, such that $\mathbf{x} * \mathbf{y} = z$. In the second step, parties test the correctness of this tuple using an auxiliary random inner-product tuple $(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle c \rangle)$ and a random field element $s \in \mathbb{G}$.

The idea here is that both steps will maintain the "incorrectness", if any, of the input tuples with high probability.

We note that the parties make use of a broadcast channel in the second phase, which does not respect our restriction to servers which communicate only with P_R in Phase ρ of the protocol. This broadcast channel will not be required by the next protocol.

Protocol $\Pi_{\mathsf{MultCheck}}$

We consider an extension field $\mathbb{G} \supseteq \mathbb{F}$. We assume access to a random coin functionality, $\mathsf{RandomCoin}$.

Phase 1 P_S sends the shares $\langle x_\ell \rangle_i, \langle y_\ell \rangle_i, \langle z_\ell \rangle_i$ for $\ell \in [m]$, and the shares $\langle \mathbf{a} \rangle_i, \langle \mathbf{b} \rangle_i, \langle c \rangle_i$ of a random inner-product tuple to each server P_i .

Sampling The parties call $\mathsf{RandomCoin}$ to obtain $R \in \mathbb{G}$ and $s \in \mathbb{G}$.

Phase 2 The servers parties proceed as follows:

1. Lift $\{\langle x_\ell \rangle, \langle y_\ell \rangle, \langle z_\ell \rangle\}_{\ell \in [m]}$ to $\mathbb{G}$.
2. Set
   $\langle \mathbf{x} \rangle = (\langle x_1 \rangle, R \cdot \langle x_2 \rangle, \dots, R^{m-1} \cdot \langle x_m \rangle)$,
   $\langle \mathbf{y} \rangle = (\langle y_1 \rangle, \langle y_2 \rangle, \dots, \langle y_m \rangle)$
   $\langle z \rangle = \sum_{\ell \in [m]} R^{\ell-1} \cdot \langle z_\ell \rangle$
3. Compute $\langle \boldsymbol{\sigma} \rangle = s \cdot \langle \mathbf{x} \rangle - \langle \mathbf{a} \rangle$ and $\langle \boldsymbol{\rho} \rangle = \langle \mathbf{y} \rangle - \langle \mathbf{b} \rangle$.
4. Open $\langle \boldsymbol{\sigma} \rangle$ and $\langle \boldsymbol{\rho} \rangle$ using a broadcast channel.
5. Compute $\langle v \rangle = s \cdot \langle z \rangle - \langle c \rangle - \langle \mathbf{b} \rangle * \boldsymbol{\sigma} - \langle \mathbf{a} \rangle * \boldsymbol{\rho} - \boldsymbol{\rho} * \boldsymbol{\sigma}$.
6. Send $\langle v \rangle$ to P_R.

The receiver party P_R accepts if $v = 0$ and rejects if not.

Figure 2: Protocol $\Pi_{\mathsf{MultCheck}}$

Lemma 1. *If at least one multiplication triple is incorrect, the resulting inner-product tuple obtained in Step 2. of protocol $\Pi_{\mathsf{MultCheck}}$ is correct with probability at most $\frac{m-1}{|\mathbb{G}|}$.*

Proof: We show that if at least one triple is incorrect then $\langle \mathbf{x} \rangle, \langle \mathbf{y} \rangle, \langle z \rangle$ is an incorrect inner-product tuple except with probability at most $\frac{m-1}{|\mathbb{G}|}$. We construct three polynomials $F(t) = x_1 \cdot y_1 + t \cdot x_2 \cdot y_2 + \cdots + t^{m-1} \cdot x_m \cdot y_m$, $G(t) = z_1 + t \cdot z_2 + \cdots + t^{m-1} \cdot z_m$ and $H(t) = F(t) - G(t)$. If there was at least one incorrect tuple, then $F(\cdot) \neq G(\cdot)$ and, since $H(t)$ is a non-zero polynomial of degree $m-1$, there are at most $m-1$ values R such that $H(R) = 0$. Therefore, since R is sampled uniformly at random in $\mathbb{G}$, the probability that incorrect triples lead to a correct inner-product tuple is at most $\frac{m-1}{|\mathbb{G}|}$. □

Lemma 2. *If at least one of the two inner-product tuples $(\mathbf{x}, \mathbf{y}, z)$ and $(\mathbf{a}, \mathbf{b}, c)$ is incorrect, the probability that the check passes is $2/|\mathbb{G}|$.*

Proof: First we show that if the two inner product tuples $\langle \mathbf{x} \rangle, \langle \mathbf{y} \rangle, \langle z \rangle$ and $\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \langle c \rangle$ are correct, then the sacrifice will pass:

$$\begin{aligned}
v &= s \cdot z - c - \mathbf{b} * \boldsymbol{\sigma} - \mathbf{a} * \boldsymbol{\rho} - \boldsymbol{\rho} * \boldsymbol{\sigma} \\
&= s \cdot z - c - \sum_{i \in [m]} b_i \cdot (R^i x_i - a_i) - \sum_{i \in [m]} a_i \cdot (y_i - b_i) \\
&\quad - \sum_{i \in [m]} (R^i x_i - a_i) \cdot (y_i - b_i) \\
&= s \cdot z - c - s \cdot \sum_{i \in [m]} R^{i-1} x_i \cdot y_i + \sum_{i \in [m]} a_i \cdot b_i = 0
\end{aligned}$$

We compute now the probability that incorrect tuples pass the sacrifice. Let $\varDelta_z = z - \mathbf{x} * \mathbf{y}$ and $\varDelta_c = c - \mathbf{a} * \mathbf{b}$ we have :

$$\begin{aligned} v &= s \cdot z - c - \mathbf{b} * \boldsymbol{\sigma} - \mathbf{a} * \boldsymbol{\rho} - \boldsymbol{\rho} * \boldsymbol{\sigma} \\ &= s \cdot (\mathbf{x} * \mathbf{y} + \varDelta_z) - (\mathbf{a} * \mathbf{b} + \varDelta_c) - \mathbf{b} * \boldsymbol{\sigma} - \mathbf{a} * \boldsymbol{\rho} - \boldsymbol{\rho} * \boldsymbol{\sigma} \\ &= s \cdot \varDelta_z - \varDelta_c \end{aligned}$$

The check will pass if the condition $s \cdot \varDelta_z - \varDelta_c = 0$ holds. If $\varDelta_z = 0$, then the condition is verified if $\varDelta_c = 0$ which contradicts the assumption; if $\varDelta_z \neq 0$ and $\varDelta_c = 0$, the condition is verified if and only is $s = 0$ which happens with probability $1/|\mathbb{G}|$; finally, if both $\varDelta_z \neq 0$ and $\varDelta_c \neq 0$ the condition holds iff $s = \varDelta_c/\varDelta_z$ that happens with probability $1/|\mathbb{G}|$. Combining the cases we conclude proof. □

Combining the two previous lemma we obtain.

Proposition 3. *We have that if at least one of the m triples $\{(\langle x_i\rangle, \langle y_i\rangle, \langle z_i\rangle)\}_{i\in[m]}$ is incorrect, the probability that the protocol $\Pi_{\mathsf{MultCheck}}$ outputs* accept *is at most $\frac{m-1}{|\mathbb{G}|} + (1 - \frac{m-1}{|\mathbb{G}|}) \cdot \frac{2}{|\mathbb{G}|}$.*

4.2 Second Multiplication Check Protocol

Here we describe a more efficient protocol which allows to compress the size of the inner-product to be tested in order to reduce the communication complexity at the expense of (potentially) more interactions. The protocol $\Pi_{\mathsf{CompressedMC}}$, described in Figure 3, uses two core subroutines, Π_{Compress} and $\Pi_{\mathsf{CompressRand}}$ given in Figure 4, which compress a set of k inner-product tuples down to only one (of the same dimension) in such a way that, with high probability, the output tuple is incorrect if one of the inputs is. The difference between the two subroutines is that the second introduces randomness in such a way that the compressed tuple can be opened without leaking information about the input tuples. This also enables the protocol to dispense with the broadcast channel used in $\Pi_{\mathsf{MultCheck}}$.

The protocol assumes access to a $\mathsf{RandomCoin}$ functionality and to two untrusted subroutines $\Pi_{\mathsf{InnerProd}}$ and Π_{Rand}, which we don't instantiate. On input of two vectors $\langle\mathbf{a}\rangle$ and $\langle\mathbf{b}\rangle$, $\Pi_{\mathsf{InnerProd}}$ outputs a possibly incorrect $\langle c\rangle$, with $\mathbf{a} * \mathbf{b} = c$. When queried by the servers, Π_{Rand} outputs a possibly biased random value. At a high level, $\Pi_{\mathsf{CompressedMC}}$ proceeds as follows. The first step is similar to the first step in $\Pi_{\mathsf{MultCheck}}$, where parties produce the inner-product tuple $(\langle\mathbf{x}\rangle, \langle\mathbf{y}\rangle, \langle z\rangle)$ of dimension m. To reduce the dimension of this tuple, parties divide the vectors $\langle\mathbf{x}\rangle$ and $\langle\mathbf{y}\rangle$ into k smaller vectors of dimension ℓ and perform Π_{Compress}. In this way parties obtain a single inner-product tuple, but this time of dimension $\ell = m/k$, for any divisor k of m. This step can then be repeated with identical or different values of k until a final inner-product tuple (potentially of dimension 1) needs to be checked. (For identical values of k, these steps need to be repeated $\log_k m$ times to check a single multiplication triple at the end).

Lemma 3. *If one of the k input inner-product tuples is incorrect, or if any of the the $h(u)$ values is incorrectly computed by $\Pi_{\mathsf{InnerProd}}$, then the inner-product tuple output by Π_{Compress}, resp. $\Pi_{\mathsf{CompressRand}}$, is also incorrect, except with probability at most*

$$\frac{2(k-1)}{|\mathbb{G}|-k}, \quad \textit{resp.}\ \frac{2k}{|\mathbb{G}|-k}.$$

Compressed multiplication check, $\Pi_{\mathsf{CompressedMC}}$

We consider an extension field $\mathbb{G} \supseteq \mathbb{F}$. We assume access to a random coin functionality, $\mathsf{RandomCoin}$.

Phase 1 P_S sends the shares $\langle x_\ell \rangle_i, \langle y_\ell \rangle_i, \langle z_\ell \rangle_i$ in $\mathbb{F}$ to each server P_i for $\ell \in [m]$.
Sampling The parties call $\mathsf{RandomCoin}$ to obtain $R \in \mathbb{G}$.
Phase 2 The server parties proceed as follows:

1. Lift $\{\langle x_\ell \rangle, \langle y_\ell \rangle, \langle z_\ell \rangle\}_{\ell \in [m]}$ to $\mathbb{G}$.
2. Set
$\langle \mathbf{x}^0 \rangle = (\langle x_1 \rangle, R \cdot \langle x_2 \rangle, \ldots, R^{m-1} \cdot \langle x_m \rangle)$,
$\langle \mathbf{y}^0 \rangle = (\langle y_1 \rangle, \langle y_2 \rangle, \ldots, \langle y_m \rangle)$
$\langle z^0 \rangle = \sum_{\ell \in [m]} R^{\ell-1} \cdot \langle z_\ell \rangle$

For each compression round $j \in [\lfloor \log_k(m) \rfloor]$:

Phase $3 + (j-1)$ The server parties proceed as follows:

1. Parse $\langle \mathbf{x}^{j-1} \rangle$ and $\langle \mathbf{y}^{j-1} \rangle$ as
$$\langle \mathbf{x}^{j-1} \rangle = (\langle \mathbf{a}_1^j \rangle, \ldots, \langle \mathbf{a}_k^j \rangle),$$
$$\langle \mathbf{y}^{j-1} \rangle = (\langle \mathbf{b}_1^j \rangle, \ldots, \langle \mathbf{b}_k^j \rangle), \quad \mathbf{a}_u^j, \mathbf{b}_u^j \in \mathbb{G}^{m/k^j}.$$
2. Call $\Pi_{\mathsf{InnerProd}}(\langle \mathbf{a}_u^j \rangle, \langle \mathbf{b}_u^j \rangle)$ to obtain $\langle c_u^j \rangle$, for $u \in [k-1]$.
3. Set $\langle c_k^j \rangle = \langle z^{j-1} \rangle - \sum_{u \in [k-1]} \langle c_u^j s \rangle$.
4. If $j \neq \lfloor \log_k m \rfloor$, begin Π_{Compress} on $(\langle \mathbf{a}_i^j \rangle, \langle \mathbf{b}_i^j \rangle, \langle c_i^j \rangle)_{i \in [k]}$; if $j = \lfloor \log_k m \rfloor$, begin $\Pi_{\mathsf{CompressRand}}$ instead.

Sampling Within Π_{Compress} or $\Pi_{\mathsf{CompressRand}}$.
Phase $3 + j$ Complete Π_{Compress} or $\Pi_{\mathsf{CompressRand}}$ to obtain $(\langle \mathbf{x}^j \rangle, \langle \mathbf{y}^j \rangle, \langle z^j \rangle)$ of dimension m/k^j.

After the last compression round:

Phase $3 + \lfloor \log_k(m) \rfloor$ cont. Servers open the last tuple to P_R which outputs either accept if it is correct, or abort if not.

Figure 3: Compressed multiplication check

Proof: Suppose that one of the k inner-product tuples $(\langle \mathbf{x}_i \rangle, \langle \mathbf{y}_i \rangle, \langle z_i \rangle)_{i \in [k]}$ is incorrect, i.e. there exists $i \in [k]$ such that $z_i \neq \mathbf{x}_i * \mathbf{y}_i$. For this i, it then holds that $h(i) \neq \mathbf{f}(i) * \mathbf{g}(i)$ and hence that $h(\cdot) \neq \mathbf{f}(\cdot) * \mathbf{g}(\cdot)$. (This holds even if one or more of the $k-1$, resp. $k+1$, computed evaluation points of $h(\cdot)$ is maliciously altered in an effort to correct for the incorrect tuple.) By the Schwartz–Zippel lemma, the probability that the output tuple is correct, i.e. that $h(s) = \mathbf{f}(s) * \mathbf{g}(s)$ for a randomly sampled $s \in \mathbb{G} \setminus [k]$, is bounded above by $\frac{\deg h(\cdot)}{|\mathbb{G} \setminus [k]|} = \frac{2(k-1)}{|\mathbb{G}|-k}$, resp. $\frac{2k}{|\mathbb{G}|-k}$. □

Proposition 4. *If at least one of the m multiplication triples $\{(\langle x_i \rangle, \langle y_i \rangle, \langle z_i \rangle)\}_{i \in [m]}$ is incorrect, the probability that protocol $\Pi_{\mathsf{CompressedMC}}$ outputs* accept *is at most*

$$\frac{m-1}{|\mathbb{G}|} + \left(1 - \frac{m-1}{|\mathbb{G}|}\right) \cdot \left(\frac{2k}{|\mathbb{G}|-k} \cdot (1-B)^{\lfloor \log_k(m) \rfloor - 1}\right) + \left(1 - \frac{m-1}{|\mathbb{G}|}\right) \cdot \left(B \cdot \sum_{i=0}^{\lfloor \log_k(m) \rfloor - 2} (1-B)^i\right)$$

where k is the compression parameter and $B = \frac{2(k-1)}{|\mathbb{G}|-k}$

Sub-protocols $\mathit{\Pi}_{\mathsf{Compress}}$ and $\mathit{\Pi}_{\mathsf{CompressRand}}$

$\mathit{\Pi}_{\mathsf{CompressRand}}$ is identical to $\mathit{\Pi}_{\mathsf{Compress}}$ except where highlighted below.
INPUT: k inner-product tuples $(\langle \mathbf{x}_i \rangle, \langle \mathbf{y}_i \rangle, \langle z_i \rangle)_{i \in [k]}$, of dimension ℓ.

1. Define two dimension-ℓ vectors of degree-$(k-1)$ polynomials $\langle \mathbf{f}(\cdot) \rangle, \langle \mathbf{g}(\cdot) \rangle$ such that:

$$\mathbf{f}(u) = \begin{pmatrix} f_1(u) \\ \vdots \\ f_\ell(u) \end{pmatrix} = \mathbf{x}_u, \quad \mathbf{g}(u) = \begin{pmatrix} g_1(u) \\ \vdots \\ g_\ell(u) \end{pmatrix} = \mathbf{y}_u, \quad \forall u \in [k].$$

   In $\mathit{\Pi}_{\mathsf{CompressRand}}$, for $j \in [\ell]$, the f_j and g_j polynomials are of degree k and are defined by the additional points $\langle f_j(k+1) \rangle = \langle v_j \rangle$ and $\langle g_j(k+1) \rangle = \langle w_j \rangle$ where the shares of v_j and w_j are given by $\mathit{\Pi}_{\mathsf{Rand}}$.
2. Define the polynomial $h(\cdot)$ of degree $2(k-1)$ such that:

$$\begin{aligned} \langle h(u) \rangle &= \langle z_u \rangle, & \forall u \in [k], \\ \langle h(u) \rangle &= \mathit{\Pi}_{\mathsf{InnerProd}}(\langle \mathbf{f}(u) \rangle, \langle \mathbf{g}(u) \rangle), & \forall i \in [k+1, 2k-1]. \end{aligned}$$

   In $\mathit{\Pi}_{\mathsf{CompressRand}}$, h is of degree $2k$ and is defined by the two additional points $\langle h(2k) \rangle$ and $\langle h(2k+1) \rangle$ defined as the other points $\langle h(i) \rangle$ for $i \in [k+1, 2k-1]$.

Sampling Call $\mathsf{RandomCoin}$ to obtain $s \in \mathbb{G} \setminus [k]$,

3. Compute $\langle \mathbf{f}(s) \rangle$, $\langle \mathbf{g}(s) \rangle$, $\langle h(s) \rangle$.

OUTPUT: One tuple $(\langle \mathbf{f}(s) \rangle, \langle \mathbf{g}(s) \rangle, \langle h(s) \rangle)$ of dimension ℓ.

Figure 4: Compressing inner products

Proof: We express the probability that the inner-product tuples are randomly "corrected" by the public coins at some point in the protocol (after which the protocol always outputs accept). Denote by A the event that "$\mathit{\Pi}_{\mathsf{CompressedMC}}$ outputs accept when at least one of the m triples is incorrect."

First, denote by A_1 the event that the tuple $(\langle \mathbf{x}^0 \rangle, \langle \mathbf{y}^0 \rangle, \langle z^0 \rangle)$ produced at step 2 of $\mathit{\Pi}_{\mathsf{CompressedMC}}$ is correct after randomizing by R; Lemma 1 implies that $\Pr[\mathsf{A}_1] = \frac{m-1}{|\mathbb{G}|}$. We then have that

$$\begin{aligned} \Pr[\mathsf{A}] &= \Pr[\mathsf{A}_1] + \Pr[\neg \mathsf{A}_1] \cdot \Pr[\mathsf{A} \mid \neg \mathsf{A}_1] \\ &= \frac{m-1}{|\mathbb{G}|} + \left(1 - \frac{m-1}{|\mathbb{G}|}\right) \cdot \Pr[\mathsf{A} \mid \neg \mathsf{A}_1]. \end{aligned}$$

Next, denote by A_2^j the event that the tuple output by $\mathit{\Pi}_{\mathsf{Compress}}$ or $\mathit{\Pi}_{\mathsf{CompressRand}}$ in the j-th compression round is correct after the random sampling of s; we note that A_2^0 is the same event as A_1. Lemma 3 implies that, for $j = 1, \ldots, \lfloor \log_k(m) \rfloor - 1$, $\Pr[\mathsf{A}_2^j] = \frac{2(k-1)}{|\mathbb{G}|-k}$, and that, for $j = \lfloor \log_k(m) \rfloor$, $\Pr[\mathsf{A}_2^j] = \frac{2k}{|\mathbb{G}|-k}$. For $j = 0, \ldots, \lfloor \log_k(m) \rfloor - 2$, we then have that

$$\begin{aligned} \Pr[\mathsf{A} \mid \neg \mathsf{A}_2^j] &= \Pr[\mathsf{A}_2^{j+1}] + \Pr[\neg \mathsf{A}_2^{j+1}] \cdot \Pr[\mathsf{A} \mid \neg \mathsf{A}_2^{j+1}] \\ &= \frac{2(k-1)}{|\mathbb{G}|-k} + \left(1 - \frac{2(k-1)}{|\mathbb{G}|-k}\right) \cdot \Pr[\mathsf{A} \mid \neg \mathsf{A}_2^{j+1}], \end{aligned}$$

and for $j = \lfloor \log_k(m) \rfloor - 1$:

$$\Pr[\mathsf{A} \mid \neg \mathsf{A}_2^j] = \frac{2k}{|\mathbb{G}|-k} + \left(1 - \frac{2k}{|\mathbb{G}|-k}\right) \cdot \Pr[\mathsf{A} \mid \neg \mathsf{A}_2^{j+1}].$$

Finally, if the last compression round using $\Pi_{\mathsf{CompressRand}}$ does not correct the tuple, then P_R will cause $\Pi_{\mathsf{CompressedMC}}$ to output reject. This implies that $\Pr[\mathsf{A} \mid \neg \mathsf{A}_2^{\lfloor \log_k(m) \rfloor}] = 0$.

Putting everything together, the protocol outputs accept with at most

$$\begin{aligned}\Pr[\mathsf{A}] =& \frac{m-1}{|\mathbb{G}|} + \left(1 - \frac{m-1}{|\mathbb{G}|}\right) \cdot \\ & \left(\frac{2k}{|\mathbb{G}|-2} \cdot \left(1 - \frac{2(k-1)}{|\mathbb{G}|-k}\right)^{\lfloor \log_k(m) \rfloor - 1} \right. \\ & \left. + \frac{2(k-1)}{|\mathbb{G}|-k} \sum_{i=0}^{\lfloor \log_k(m) \rfloor - 2} \cdot \left(1 - \frac{2(k-1)}{|\mathbb{G}|-k}\right)^{i} \right)\end{aligned}$$

□

5 Our Zero-knowledge Argument for Arithmetic and Boolean Circuits

We describe now our ZK system for circuit satisfiability based on the MPCitH paradigm. We combine a concrete MPC protocol which verifies all the properties defined in Definition 6 and the general ρ-phase ZK interactive oracle protocol $\Pi_{\rho-\mathsf{ZKIOP}}$ defined in Section 6. Given an NP relation $\mathcal{R}$, we consider a circuit C over a finite field $\mathbb{F}$ such that $C(w) = 1$ if and only if $(x, w) \in \mathcal{R}$. Without loss of generality we assume that C only contains linear and multiplication gates.

Our MPC instantiation Concretely, our MPC protocol Π_f can be divided in two phases. First, we have an input and evaluation phase where the sender client P_S generates and distributes to the servers P_i, $i \in [n]$, an additive sharing of the input and sharings of the output of each multiplication gate in the circuit. Given those, the servers locally evaluate the circuit. In the second phase, parties run the protocol $\Pi_{\mathsf{CompressedMC}}$ described in the previous section where P_S further plays the role of $\Pi_{\mathsf{InnerProd}}$ and Π_{Rand}.

Looking ahead, the protocol Π_f, and therefore the MPCitH protocol based on it, will depend on several parameters: the size of the circuit C, m, i.e. the number of multiplication gates, the number n of servers parties in Π_f, the size of the fields $\mathbb{F}$ and $\mathbb{G}$, with $|\mathbb{G}| > m - 1$, and the compression parameter k used in $\Pi_{\mathsf{CompressedMC}}$.

Proposition 5. *The Π_f protocol derived from $\Pi_{\mathsf{CompressedMC}}$ is correct, $(P_R, n-1)$-private, $(P_S, 0)$-robust with robustness error*

$$\begin{aligned}\delta_k =& \frac{m-1}{|\mathbb{G}|} + \left(1 - \frac{m-1}{|\mathbb{G}|}\right) \cdot \\ & \left(\frac{2k}{|\mathbb{G}|-2} \cdot \left(1 - \frac{2(k-1)}{|\mathbb{G}|-k}\right)^{\lfloor \log_k(m) \rfloor - 1} \right. \\ & \left. + \frac{2(k-1)}{|\mathbb{G}|-k} \sum_{i=0}^{\lfloor \log_k(m) \rfloor - 2} \cdot \left(1 - \frac{2(k-1)}{|\mathbb{G}|-k}\right)^{i} \right),\end{aligned}$$

and a client-server ρ-phase protocol, with $\rho = \lfloor \log_k(m) \rfloor$.

Proof sketch: *(Privacy)* When given the set I of opened servers, $\mathcal{S}$ first simulates the calls to $\mathsf{RandomCoin}$ to obtain the public randomness used by the protocol and then generates random

$\mathit{\Pi}_{\mathsf{Int\_ZKP}}$—**Part 1**

INPUTS: Public circuit C over $\mathbb{F}$ with m MULT gates. Extension field $\mathbb{G}$. Public input x. Private input w for $\mathcal{P}$.
OUTPUTS: Public proof oracles $(\pi_1, \dots, \pi_\rho)$ from $\mathcal{P}$. Private output $b \in \{0,1\}$ from $\mathcal{V}$.
First Oracle π_1

Prover Execute phase 1 of $\mathit{\Pi}_{\mathsf{CompressedMC}}$.

- Client party P_S executes the following for each evaluation $t \in [\tau]$:
  1. Generate a sharing of the witness $\langle w_t \rangle \xleftarrow{\$} \mathbb{F}^n$ and add $\langle w_t \rangle_i$ to $(\mathbf{m}_{t,1})_i$.
  2. For each multiplication gate $\ell \in [m]$:
     (a) Compute the multiplication result: $z_{t,\ell} \leftarrow x_{t,\ell} \cdot y_{t,\ell}$.
     (b) Generate a sharing of the result $\langle z_{t,\ell} \rangle \xleftarrow{\$} \mathbb{F}^n$ and add $\langle z_{t,\ell} \rangle_i$ to $(\mathbf{m}_{t,1})_i$.
  3. Send $(\mathbf{m}_{t,1})_i$ to P_i.
- The server parties execute the following, also for each evaluation $t \in [\tau]$:
  1. Append $(\mathbf{m}_{t,1})_i$ to $\mathsf{view}^i_{t,1}$.
  2. Compute the input shares $\langle x_{t,\ell} \rangle_i$ and $\langle y_{t,\ell} \rangle_i$ for each multiplication gate $\ell \in [m]$ using the shares from $(\mathbf{m}_{t,1})_i$.

Set $(\pi_1)_{t,i} = \mathsf{view}^1_{t,i}$.

Interactive Protocol—First Round.

Verifier Sample a random challenge $R \xleftarrow{\$} \mathbb{G}$ and send it to $\mathcal{P}$ as the output of $\mathsf{RandomCoin}$.
Prover Continue the τ executions of $\mathit{\Pi}_{\mathsf{CompressedMC}}$ by running P_S and the servers $\{P_i\}$ on input R as follows:

1. Each server P_i lifts $\langle x_{t,\ell} \rangle_i, \langle y_{t,\ell} \rangle_i, \langle z_{t,\ell} \rangle_i$ from $\mathbb{F}$ to $\mathbb{G}$.
2. Each server computes their share of $\langle \mathbf{x}^0_t \rangle, \langle \mathbf{y}^0_t \rangle$ and $\langle z^0_t \rangle$ such that:
   $\langle (\mathbf{x}^0_t)_\ell \rangle = R^{\ell-1} \langle x_{t,\ell} \rangle$, $\langle (\mathbf{y}^0_t)_\ell \rangle = \langle y_{t,\ell} \rangle$, and $\langle z^0_t \rangle = \sum_{\ell \in [m]} R^{\ell-1} \langle z_{t,\ell} \rangle$.

Interactive Protocol—Compression Rounds. For each compression round $j \in [\lfloor \log_k(m) \rfloor]$:

Prover Before creating the next oracle, emulate the following computation.

- For the client P_S, for each $t \in [\tau]$:
  1. Parse $\mathbf{x}^{j-1}_t \to (\mathbf{a}^j_{t,1}, \dots, \mathbf{a}^j_{t,k})$ and $\mathbf{y}^{j-1}_t \to (\mathbf{b}^j_{t,1}, \dots, \mathbf{b}^j_{t,k})$.
  2. For each $u \in [k-1]$: Compute inner-products: $c^j_{t,u} \leftarrow \mathbf{a}^j_{t,u} * \mathbf{b}^j_{t,u}$, generate sharing $\langle c^j_{t,u} \rangle \xleftarrow{\$} \mathbb{G}^n$ and add $\langle c^j_{t,u} \rangle_i$ to $(\mathbf{m}_{t,j+1})_i$.
  3. Compute last inner-product: $c^j_{t,u} \leftarrow z^{j-1}_t - \sum_{u \in [k-1]} c^j_{t,u}$.
  4. Construct $\mathbf{f}^j_t, \mathbf{g}^j_t \in (\mathbb{G}[X])^{m/k}$ as in $\mathit{\Pi}_{\mathsf{Compress}}$ if $j \neq \lfloor \log_k(m) \rfloor$, or as in $\mathit{\Pi}_{\mathsf{CompressRand}}$ otherwise.
  5. For each $u \in [k+1, 2k-1]$ if $j \neq \lfloor \log_k(m) \rfloor$, or $u \in [k+1, 2k+1]$ otherwise: Compute inner-product $h^j_t(u) = \mathbf{f}^j_t(u) * \mathbf{g}^j_t(u)$, generate sharing $\langle h^j_t(u) \rangle \xleftarrow{\$} \mathbb{G}^n$ and add $\langle h^j_t(u) \rangle_i$ to $(\mathbf{m}_{t,j+1})_i$.
  6. Send $(\mathbf{m}_{t,j+1})_i$ to P_i.
- For each server party P_i, for each evaluation $t \in [\tau]$:
  1. Append $(\mathbf{m}_{t,j+1})_i$ to $\mathsf{view}^{j+1}_{t,i}$. and compute $\langle \mathbf{f}^j_t \rangle_i, \langle \mathbf{g}^j_t \rangle_i, \langle h^j_t \rangle_i$ using the shares from $(\mathbf{m}_{t,j+1})_i$.

Set $(\pi_{j+1})_{t,i} = \mathsf{view}^{j+1}_{t,i}$.

Verifier Sample a random challenge $s_j \xleftarrow{\$} \mathbb{G} \setminus [k]$ and send it to $\mathcal{P}$ as the output of $\mathsf{RandomCoin}$.
Prover Continue the τ executions of $\mathit{\Pi}_{\mathsf{CompressedMC}}$ by running P_S and the servers $\{P_i\}$ on input s_j as follows:

1. Each server computes their own share of $\langle \mathbf{f}^j_t(s_j) \rangle, \langle \mathbf{g}^j_t(s_j) \rangle$ and $\langle h^j_t(s_j) \rangle$ and labels them as $\langle \mathbf{x}^j_t \rangle, \langle \mathbf{y}^j_t \rangle$ and $\langle z^j_t \rangle$.
2. The sender P_S computes $\mathbf{x}^j_t, \mathbf{y}^j_t$ and z^j_t in the same way.

Figure 5: Interactive (Zero-knowledge) proof (of knowledge) protocol—Part 1

$\Pi_{\mathsf{Int\_ZKP}}$—**Part 2**

Interactive Protocol—Final Rounds.

Prover After computation of the final compressed tuple, for each $t \in [\tau]$ the server parties $\{P_i\}$ each send their shares $\langle \mathbf{f}_t^j(s_j)\rangle_i, \langle \mathbf{g}_t^j(s_j)\rangle_i, \langle h_t^j(s_j)\rangle_i$, with $j = \lfloor \log_k(m) \rfloor$, to P_R, together with their shares $\langle o \rangle_i$ of the values of the output wires of C; all of these form $\mathsf{view}_{t,R}$, which $\mathcal{P}$ sends to $\mathcal{V}$ in full.

Verifier Upon receiving $\mathsf{view}_{t,R}$, for each $t \in [\tau]$, check that the tuple is correct, i.e. that $h_t^j(s_j) = \mathbf{f}_t^j(s_j) * \mathbf{g}_t^j(s_j)$, with $j = \lfloor \log_k(m) \rfloor$, and check that the output of the circuit is valid, i.e. that $\sum_i \langle o \rangle_i = 1$. If one of these fails, reject.

Oracle query The verifier picks a subset $Q_t \subset [n]$ of size $n-1$ uniformly at random for each $t \in [\tau]$ and queries $\{Q_t\}$.

Verifier Upon receiving $\{\mathsf{view}_{t,q}\}_{q \in Q_t}$ for each $t \in [\tau]$ (where $\mathsf{view}_{t,i} = \mathsf{view}_{t,i}^1 \| \dots \| \mathsf{view}_{t,i}^\rho$), recompute the operations of each opened server P_q to check for inconsistencies with $\mathsf{view}_{t,R}$. If an inconsistency is found, reject. If not, accept.

Figure 6: Interactive (Zero-knowledge) proof (of knowledge) protocol—Part 2

shares for all the sharings that P_S provides to the servers. Finally, it edits the communication of the last hidden party with P_R so that the circuit appears to output 1 and the final checking tuple appears to be correct. As the sharings of the wire values are sampled independently, and as $\Pi_{\mathsf{CompressRand}}$ introduces uniform randomness into the last tuple, the above sampling strategy is indistinguishable from an honest execution of Π_f.

(Robustness) The only actions a corrupt sender P_S^* can take are to send incorrect tuples as the multiplication results of the circuit or incorrect results as the output of $\Pi_{\mathsf{InnerProd}}$ during the compression rounds. Should any of these happen, Proposition 4 gives us that the protocol accepts with probability at most δ_k as in the statement.

□

Putting Everything Together. We describe our MPCitH ZK-IOP for arithmetic and Boolean circuit in Figures 5 and 6. The protocol $\Pi_{\mathsf{Int\_ZKP}}$ is derived directly from the parallel execution variant of $\Pi_{\rho-\mathsf{ZKIOP}}$, instantiating Π_f^τ with the MPC protocol described above. Combining results from previous sections, we obtain the following theorem.

Theorem 3. *Let n, m, k be integers and $\mathbb{F} \subseteq \mathbb{G}$ finite fields. Let C be a circuit over $\mathbb{F}$ of multiplicative size m and $|\mathbb{G}| > m$. The protocol $\Pi_{\mathsf{Int\_ZKP}}$ satisfies completeness, soundness and (honest-verifier) zero-knowledge as in Definition 4 with soundness error $\epsilon = 1/n^\tau + (1 - 1/n^\tau) \cdot \delta_k$ and round complexity $\lfloor \log_k(m) \rfloor + 2$.*

From ZK-Interactive MPCitH Proof to ZK Arguments We can compile the interactive ZK proof described in Figures 5 and 6 to an interactive argument, with standard techniques using collision-resistant hash functions. In particular, as described [KKW18], we can achieve better efficiency using collision-resistant hash functions based on Merkle trees [Mer90].

Setting the Parameters. Notice the parameters of our zero-knowledge argument protocol greatly depends on the size of the base field $\mathbb{F}$ and extension field $\mathbb{G}$, other than the compression factor k. In general, for small values of k we have smaller proof size, but larger running times. In Table 2 we show the number of repetitions and estimated proof when the base field $\mathbb{F} = \mathbb{F}_2$ with $k = 8$. Notice that since we choose a big extension filed, $\mathbb{G} = \mathbb{F}_{2^{64}}$, the number of repetitions is the same for different circuit size, but it varies depending on the number of parties.

Circuit size	$(n = 16, \tau = 11)$	$(n = 64, \tau = 7)$	$(n = 128, \tau = 6)$
10^3	4.9	3.5	2.5
10^4	18.5	11.7	10
10^5	143	91	78
10^6	1382	879.5	753.8

Table 2: Number τ of parallel repetitions and proof size (in kB) needed to achieve *interactive* proof soundness of 2^{-40} with compression $k = 8$ and extension field $\mathbb{G} = \mathbb{F}_{2^{64}}$, depending on number n of parties and circuit size.

6 Non-interactive Zero-knowledge Arguments

Using the Fiat-Shamir paradigm [FS87,PS96], we can transform our public coin interactive protocol to a corresponding non-interactive zero-knowledge protocol. Roughly, the prover will compute the first-round message as in the interactive variant and then continue the protocol by setting the verifier's next message to be the output of a hash function H modelled as a random oracle on input the transcript of previous messages.

While the zero-knowledge property directly follows from the corresponding property of the interactive variant, soundness requires more careful analysis. In [BCS16], the authors prove that for IOP systems the soundness of the transformed non-interactive protocol can be derived form the soundness of the IOP verifier against "state restoration attacks". This section presents a better estimation of the soundness of our non-interactive protocol.

6.1 Soundness with independent challenges

This first analysis applies to the non-optimised variant of the protocol where each of the τ parallel executions receives a random challenge from $\mathsf{RandomCoin}$, independently of the other executions. When producing a non-interactive proof, before proceeding to the next round, the prover can re-randomize the commitments they make to the random oracle in order to sample different public coins for the checks. Here the best cheating strategy is to attack different executions at each round of interaction so that, by the end of the protocol, all executions will cause the verifier to accept.

Assuming that the final ZK protocol has r rounds of interaction between prover $\mathcal{P}$ and verifier $\mathcal{V}$, we let X_i, for $i \in [r]$, be the random variable of the maximum number (out of the remaining incorrect executions) of "good" challenges received by the prover during all its queries to the i-th random oracle. (By "good" challenge we mean one which corrects and "hides" any cheating in that execution.)

As demonstrated in previous work on this kind of non-interactive protocol [Bd20,BdSGK+21], the number of "good" challenges received for each call to the random oracle follows a binomial distribution with parameters (τ_i, p_i), where τ_i denotes the number of parallel executions for which this challenge is "good" and p_i denotes the probability that a random challenge is "good" for one execution.

The prover's goal is to receive a "good" challenge in one of the interaction rounds for each of the τ parallel executions. This means that the soundness error is the probability that this strategy succeeds, namely $\Pr\left[\sum_{i=1}^{r} X_i = \tau\right]$.

Specifically to our protocol $\Pi_{\mathsf{Int\_ZKP}}$, we identify the following interactions between the prover and the verifier in the interactive variant:

1. $\mathcal{P}$ commits to the injections of the m values; $\mathcal{V}$ responds with challenge $R \in \mathbb{G}$.
2. For each $j \in [\lfloor \log_k m \rfloor]$: the prover commits to the c_i^j injections (i.e. to the values P_S sends to the server parties P_i), for $i \in [k-1]$, and the $\langle h(i) \rangle$ injections (in Π_{Compress}), for $i \in [k+1, 2k-1]$; $\mathcal{V}$ responds with challenge $s_j \in \mathbb{G}$.
3. At step $j = \lfloor \log_k m \rfloor$, the prover also commits to the additional points required by $\Pi_{\mathsf{CompressRand}}$.

In the non-interactive setting, we therefore have the following probabilities of obtaining a "good" challenge correctly for each of the interaction rounds:

First round. Probability that R makes the tuple correct: $p_R = \frac{m-1}{|\mathbb{G}|}$.

Intermediary rounds. For $j \in [\lfloor \log_k m \rfloor - 1]$ (last round is special as it has polynomials of different degrees), probability that the Schwartz–Zippel test fails to catch a non-zero polynomial, i.e. Π_{Compress} outputs a correct tuple: $p_{\mathsf{int}} = \frac{2(k-1)}{|\mathbb{G}|-k}$

Final round. Probability that the last Schwartz–Zippel test fails, i.e. that $\Pi_{\mathsf{CompressRand}}$ outputs a correct tuple: $p_{\mathsf{fin}} = \frac{2k}{|\mathbb{G}|-k}$.

The soundness of the non-interactive protocol, with the independent challenges variant, is therefore given by

$$\epsilon_{\mathsf{ni}}^{\mathsf{indep}} = \Pr\left[W + \sum_{j=1}^{\lfloor \log_k m \rfloor - 1} X_j + Y + Z = \tau \right],$$

where

$$W = \max_{q_1}\{W_{q_1}\} \qquad W_{q_1} \sim \mathfrak{B}\left(\tau, p_R\right)$$

$$X_j = \max_{q_{j,2}}\{X_{q_{j,2}}\} \qquad X_{j,q_2} \sim \mathfrak{B}\left(\tau - W - \sum_{i=1}^{j-1} X_i, p_{\mathsf{int}}\right)$$

$$Y = \max_{q_3}\{Y_{q_3}\} \qquad Y_{q_3} \sim \mathfrak{B}\left(\tau - W - \sum_{i=1}^{\lfloor \log_k m \rfloor - 1} X_i, p_{\mathsf{fin}}\right)$$

$$Z = \max_{q_4}\{Z_{q_4}\} \qquad Z_{q_4} \sim \mathfrak{B}\left(\tau - W - \sum_{i=1}^{\lfloor \log_k m \rfloor - 1} X_i - Y, \frac{1}{N}\right)$$

with q_i denoting the queries to the i-th random oracle and $\mathfrak{B}$ denoting the binomial mass function.

6.2 Soundness with identical challenges

The optimised protocol presented in Section 5, where the challenges output by $\mathsf{RandomCoin}$ are shared across the τ executions, has a different distribution of "good" challenges.

Considering the first round, a malicious prover can commit to τ cheating strategies each represented by the values of $\{\mathbf{m}_t\}_{t\in[\tau]}$; these are namely the sharings of the witness w_t and of each multiplication output $z_{t,\ell}$, for $\ell \in [m]$. Using the notation of the proof of Lemma 1, each of these strategies defines a polynomial $H^{(t)}$ whose zeroes define a "good" first-round challenge. Indeed, recall from Lemma 1 that a challenge $R \in \mathbb{G}$ corrects a set of incorrect multiplication triples if and only if $H^{(t)}(R) = 0$ when $H^{(t)}$ is not the zero polynomial (due to the error in at least one of the triples). Denote by $\mathcal{H}^{(t)}$ the set $\{r \in \mathbb{G} : H^{(t)}(r) = 0\}$ of "good" challenges.

As the first round challenge R is shared across executions, if the malicious prover wishes to correct τ_1 out of τ executions, then the probability of this happening is highest when at least τ_1 of the zero sets $\mathcal{H}^{(t)}$ are identical. In this case, the probability that R is a "good" challenge for these τ_1 executions is exactly

$$\frac{m-1}{|\mathbb{G}|},$$

independently of τ_1. This implies that, here, the distribution W of $\epsilon_{\mathsf{ni}}^{\mathsf{indep}}$ can take any value between 1 and τ with this probability, depending on the prover's strategy, and is 0 otherwise.

Following the same reasoning, we have that the probability of sampling a "good" challenge for τ' executions in the intermediary rounds or the final rounds can be as high as

$$\frac{2(k-1)}{|\mathbb{G}|-k} \quad \text{or} \quad \frac{2k}{|\mathbb{G}|-k},$$

respectively, when the prover cheats identically across these τ' executions. Indeed, even in the final round when the h polynomial is randomised, since the prover also controls Π_{Rand},the sets of zeros can still be made identical. Similarly, this implies that the X_j and Y distributions can here also take any value between 1 and τ with the above fixed probabilities.

Only the Z distribution of $\epsilon_{\mathsf{ni}}^{\mathsf{indep}}$ remains the same due to the independent sampling of the τ challenges for the opening of the views of $n-1$ parties in each execution. Putting this all together implies that the soundness of the non-interactive protocol, with identical $\mathsf{RandomCoin}$ challenges for all τ parallel executions is given by

$$\epsilon_{\mathsf{ni}}^{\mathsf{ident}} = \max_{(\tau_1,\ldots,\tau_{r-1})} \Pr\left[W + \sum X_j + Y + Z = \tau \;\middle|\; \sum_{i=1}^{r-1} \tau_i \le \tau \right],$$

where

$$\begin{aligned}
W &= \max_{q_1}\{W_{q_1}\}, & W_{q_1} &\in \{0,\tau_1\} \text{ and } \Pr[W_{q_1}=\tau_1] = p_R;\\
X_j &= \max_{q_{j,2}}\{X_{q_{j,2}}\}, & X_{j,q_2} &\in \{0,\tau_{j+1}\} \text{ and } \Pr[X_{j,q_2}=\tau_{j+1}] = p_{\mathsf{int}};\\
Y &= \max_{q_3}\{Y_{q_3}\}, & Y_{q_3} &\in \{0,\tau_{r-1}\} \text{ and } \Pr[Y_{q_3}=\tau_{r-1}] = p_{\mathsf{fin}};\\
Z &= \max_{q_4}\{Z_{q_4}\}, & Z_{q_4} &\sim \mathfrak{B}\left(\tau - W - \sum_{j=1}^{\lfloor \log_k m \rfloor - 1} X_j - Y, \frac{1}{N}\right).
\end{aligned}$$

7 Parameters and Performance

We describe our implementation and then present the performance of our system and compare them with other related works. Finally, we compare our signature scheme with Picnic and Banquet.

7.1 Parameters

We first describe how we choose parameters for our tests. The soundness of our scheme depends on many parameters, namely the number of parties n, the compression factor k, the extension field ℓ and the number of repetition τ. We already observed that we can trade off computation and communication using different values for n, so that increasing the number of parties will increase prover and verifier running times but it will decrease the proof size. The compression

	$\|C\| = 2^{10}$					$\|C\| = 2^{14}$					$\|C\| = 2^{16}$					$\|C\| = 2^{20}$				
n	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (s)	$t_{\mathcal{V}}$ (s)
16	8	11	6	2.4	2.3	16	11	32	39	32	16	11	102	163	159	32	11	1464	3.08s	2.91s
16	16	11	8	2.6	2.4	32	11	37	43	36	32	11	108	172	167	64	11	1476	3.01s	2.82s
32	8	9	5	3.8	3.6	16	9	26	60	58	16	9	334	83	258	32	9	1198	4.91s	4.71s
32	16	9	7	4.0	3.8	32	9	30	67	64	32	9	333	88	269	64	9	1208	4.76s	4.55s
64	8	7	4	6.7	6.5	16	7	20	92	89	16	7	297	65	394	32	7	932	7.48s	7.20s
64	16	7	5	6.9	6.6	32	7	24	102	99	32	7	294	69	413	64	7	940	7.24s	6.93s
128	8	6	3	11.0	10.6	16	6	17	155	150	16	6	55	707	677	32	6	799	13.4s	12.9s
128	16	6	4	9.7	9.3	32	6	20	162	156	32	6	58	732	701	64	6	805	12.9s	12.2s

Table 3: Performance of our interactive system for different choice of parameters to achieve 40-bit of security. n is the number of parties in the MPC protocol, the extension field is $\mathbb{G} = \mathbb{F}_{2^{64}}$, k is the compression parameter and τ the number of repetitions.

	$\|C\| = 2^{10}$					$\|C\| = 2^{12}$					$\|C\| = 2^{14}$					$\|C\| = 2^{16}$					$\|C\| = 2^{18}$					$\|C\| = 2^{20}$				
n	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (ms)	$t_{\mathcal{V}}$ (ms)	k	τ	size (KB)	$t_{\mathcal{P}}$ (s)	$t_{\mathcal{V}}$ (s)	k	τ	size (KB)	$t_{\mathcal{P}}$ (s)	$t_{\mathcal{V}}$ (s)
16	8	40	24	9	8	8	42	45	35	32	16	40	117	130	130	16	42	389	616	603	16	43	1423	2.5s	2.4s	32	43	5726	11s	11s
16	16	38	29	8	8	16	40	54	31	30	32	38	128	131	130	32	40	392	604	597	32	41	1379	2.4s	2.4s	64	41	5504	10s	9s
32	8	34	20	15	14	8	36	39	65	64	16	34	100	218	218	16	36	334	1015	1015	16	37	1220	4.2s	4.1s	32	37	4927	19s	18s
32	16	32	24	13	13	16	34	46	57	57	32	32	108	210	209	32	34	333	1004	998	32	35	1172	4.1s	4.0s	64	35	4698	17s	17s
64	8	30	18	32	32	8	32	35	112	110	16	30	88	382	381	16	32	297	1798	1796	16	33	1084	7.4s	7.2s	32	33	4394	34s	33s
64	16	28	21	24	24	16	30	40	100	98	32	28	94	360	359	32	30	294	1734	1744	32	31	1034	7.3s	7.2s	64	31	4162	31s	29s
128	8	27	16	48	48	8	29	32	202	201	16	27	79	654	670	16	29	269	3176	3220	16	30	983	14s	14s	32	30	3995	62s	62s
128	16	25	19	42	43	16	27	36	173	171	32	25	84	608	621	32	27	265	3063	3133	32	28	931	13.7s	13.8s	64	28	3759	56s	56s

Table 4: Performance of NILimbo for different choice of parameters to achieve 128-bit of security. n is the number of parties in the MPC protocol, the extension field is $\mathbb{G} = \mathbb{F}_{2^{64}}$, k is the compression parameter and τ the number of repetitions.

factor determines the round complexity of the protocol according to Theorem 3 and its soundness. In general, large values of k will allow better running times and larger proof size. The extension field greatly impact on the proof size, but not that much on the computation. We noticed that computation on $F_{2^{64}}$ were slightly faster, and we prevalently chose this extension field to run the checking step of our protocol.

Finally, in our experiments we only used fixed values of k, but the implementation can be optimized allowing different values of k, for example by considering divisors of m, where m is the number of multiplication gates. Since we chose k independently of m, we need to create random public triple values in order to perform the compression step.

7.2 Implementation

We implemented our protocol in C++ using the same dedicated field arithmetic implementation as Banquet [BdSGK+21], which we extended by adding support for computing in $GF(2^{64})$. We also reduced as much as possible the number of polynomial interpolations that are computed. In particular, when the prover needs to compute the polynomials f, g, h during the first part of the compression rounds, we first reconstruct the plain value from the shares and then do the interpolations.

In addition to the above, when we want to evaluate a binary circuit, we pack the parties' shares by chunks of 64 in a machine word. Therefore, instead of repeating the evaluation of the circuit

(n,k,τ)	$t_{\mathcal{P}}$ (ms) 1 thread	$t_{\mathcal{P}}$ (ms) 4 threads	$t_{\mathcal{V}}$ (ms) 1 thread	$t_{\mathcal{V}}$ (ms) 4 threads	Size (KB)
(8,16,48)	143	83	142	64	180
(16,32,36)	195	100	194	79	150
(32,16,33)	296	144	290	114	121
(64,16,29)	505	229	498	181	110

Table 5: NI SHA-256: Performance of our system for proving knowledge of a SHA-256 pre-image in the non-interactive setting for various number of parties n

for each party independently, a single gate can be computed for 64 parties at once using bitwise operations.

7.3 Performance

All the benchmarks were done on a desktop computer with an Intel i9-9900 (3.1GHz) CPU and 128GB of RAM. All experiments were run locally. For each experiment, we run it either on a single thread or on 4 threads, and we give the average over 100 runs in milliseconds. Although it may slightly vary depending on the parameters used for generating the proof, we try to give some insight on the computational complexity of each of the steps described in Figures 5 and 6. Thanks to the packing technique we use in the binary case, the evaluation of the circuit in MPC, which corresponds to creating the first oracle, is fast and requires less than 10% of the running time. Then, the most computationally heavy task is to lift the shares of each party and to transform them to share of an inner product, this step requires 60% of the total running time. Eventually, about 40% of the prover time is spent doing the compression rounds.

We run all the experiments in the interactive setting using the same challenge across all the τ repetitions, but we used independent challenges for the non-interactive case. This is because in the non-interactive case we need very large extension fields to achieve the desired soundness, as shown in Section 6.2. We plan to further investigate on this direction in future works.

In our experiments we set the computational security parameter $\kappa = 128$ and the statistical security parameter $\lambda = 40$.

SHA-256. Verifying a SHA-256 pre-image in zero knowledge with 2^{-40} soundness error, the size of the proof is about 42KB, with a prover running time of 53ms and a verifier running time of 47ms (Table 1) on a single threads. For Table 1 we used the *Bristol Fashion* circuit available here[2]. The circuit is made of 22573 AND gates and 135073 gates in total. As a comparison, for the same soundness error Ligero's proof size is about 44KB, with a verifier and prover running time of respectively 140ms and 62ms. We also increased the number of threads to increase prover's and verifier's performance. Performance of NILimbo for verifying SHA-256 is given in Table 5.

Binary circuits. We tested Limbo (Table 3) and NILimbo (Table 4) on random binary circuits of different size. In Table 6, we report the performance of our system on 4 threads with $n = 16$ and $n = 8$ and different circuit size. Our protocol can evaluate 2^{20} AND gates in about 8.7 (resp. 4.7 sec) in the non-interactive setting with a proof size of 6.5MB with 8 parties with one thread (resp. 4 threads), and 3 sec (resp. 0.987 sec) in the interactive setting with 6 rounds of communication and total communication of 1.2MB with one thread (resp. 4 threads).

[2] https://homes.esat.kuleuven.be/~nsmart/MPC/sha256.txt

$\lvert C\rvert$	$n=8$ size	$t_{\mathcal{P}}$ (s)	$t_{\mathcal{V}}$(s)	$n=16$ size	$t_{\mathcal{P}}$ (s)	$t_{\mathcal{V}}$ (s)
2^{14}	140(KB)	0.052	0.039	117 (KB)	0.069	0.052
2^{18}	1.6 (MB)	1.06	0.7	1.4 (MB)	1.47	1.04
2^{20}	6.5 (MB)	4.7	2.9	5.5 (MB)	6.4	4.32
2^{21}	13 (MB)	9.49	5.8	10 (MB)	13.9	9.5

Table 6: Performance of NILimbo for $n = 16$ and $n = 8$ to achieve 128-bit of security with 4 threads.

	$t_{\mathcal{P}}$ (s)		$t_{\mathcal{V}}$ (s)		Comm
M	1 thread	4 threads	1 thread	4 threads	(KB)
64	0.26	0.17	0.23	0.14	34
96	0.79	0.53	0.73	0.48	61
128	2.3	1.41	2.1	1.29	97
256	20	11	19	10.7	340
324	34	21	32	19	545
400	62	38	57	32	834

Table 7: Performance for proving matrix multiplication with soundness 2^{-40} with $n = 8$

Matrix multiplication. We also tested Limbo for verifying matrix multiplications. Instead of using the naive $O(n^3)$ multiplication algorithm, we use an inner product based protocol. In particular, given two $M \times M$ matrices, during the MPC evaluations the sender parties P_S directly injects the M^2 values corresponding to the resulting matrix, while in the checking phase parties verify that these M^2 inner products are correctly computed. Note that this approach only requires a minor modification to the our basic protocol and soundness analysis, however it does not consider the special structure of these inner products (e.g., some of them are correlated), so it can be further optimized. In Table 7, we show the performance of Limbo for different values of M. We note that, even with this simple variant of the protocol, there is a big advantage of going beyond the gate-by-gate approach both in term of computation and communication. For example, if $M = 128$ the protocol based on inner products is about 30% faster and uses about 38x less communication than the one based on multiplication gates.

7.4 Comparison with related works

We compare the performance of our scheme with the most efficient MPCitH schemes for circuit satisfiability, namely Ligero, which has the best communication complexity, and KKW, which offers the best run times. Ligero [AHIV17] reports figures for soundness error 2^{-40}, so we compare it with our interactive zero-knowledge argument. Table 3 gives the performance of our interactive system for different choice of parameters to achieve 40-bit of security. Comparing these figures with Ligero, our system gives both better run times and proof size for circuit up to roughly 2^{18} multiplication gates. For a circuit of size 2^{20} AND gates, Ligero requires more than 10 sec, whereas for the same circuit, Limbo only needs 3 sec; Ligero++, which supports R1CS, reported prover's running times about 2x slower than Ligero. For larger circuits the communication complexity of Ligero and Ligero++ is smaller than that of our protocol.

We also compare with KKW [KKW18], which has better computational performance than Ligero. For the range of parameters given in [KKW18], we observe (Table 3) that our protocol offers shorter proofs (up to 2x shorter for large circuits), and faster proof generation (up to 2x) assuming the same number of parties and security parameter. KKW implementation has been recently heavily optimized so to handle the verification of 100 SHA-256 verification (i.e. 2227200 multiplication gates) in 4.76s. Our current implementation is incomparable with Reverie as it can be potentially optimised in many ways, however NILimbo can already prove 2^{21} AND gates in 9.49 sec (Table 6) with 4 threads. This is only 2.2 times slower than running times reported for the optimized Reverie implementation on a 32 cores machine, which has a proof size of 22MB compared to 14MB of NILimbo. We plan to further optimise our implementation in future works.

We also stress that our system is, in theory, more efficient when used for arithmetic circuits, and that we could further improve our run times by choosing different values for the compression parameter k, for example larger for larger circuits. We also plan to perform more tests in these directions.

7.5 Limbo Signature

Scheme	n	Rep.	Prover (ms)	Verifier (ms)	Communication (bytes)
Banquet	16	11	2.79	1.78	4484
	32	9	3.51	2.66	3836
	64	7	4.61	3.85	3124
	128	6	7.17	6.39	2792
Picnic3	16	$(72, 12)$	1.73	1.33	4070
	16	$(48, 16)$	1.16	0.92	4750
Our	16	10	1.09	0.99	3967
	32	8	1.69	1.57	3195
	64	7	2.89	2.71	2811
	128	6	4.93	4.65	2425

Table 8: Benchmarks of interactive identification schemes at L1 security. We used a compression factor $k = 4$ and extension field $\mathbb{F}_{2^{8\ell}}$, with $\ell = 4$.

	Banquet AES-128				Limbo-Sign AES-128			
N	(ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes	(k, ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes
16	$(4, 41)$	6.34	4.84	19776	$(6, 6, 40)$	2.7	2	21520
31	(4,35)	9.11	7.53	17456	(6,6,33)	4.6	4.2	18310
57	(6, 27)	12.47	10.77	16188	(6,6,29)	7.3	6.7	16574
107	(4,28)	24.19	21.73	14880	(6,8, 28)	11.1	10	15216
255	(4, 25)	50.95	46.80	13696	(6,6,24)	29	27	14512

Table 9: Comparison between the communication cost of Banquet and the new protocol for AES-128. Picnic for the same security level reports $t_{\mathcal{S}} = 5.33ms$, $t_{\mathcal{V}} = 4.03ms$ and size 12466 bytes.

We use our zero-knowledge protocol to build a Picnic-like post-quantum signature scheme based on AES using the same methodology as BBQ [dDOS19] and Banquet [BdSGK^{+}21]. More precisely,

N	Banquet AES-192				Limbo-Sign AES-192			
	(ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes	(k, ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes
16	(4, 62)	17.23	13.16	51216	(8,6,62)	7.1	6.4	50876
31	(4,53)	25.86	21.72	45072	(8,6,51)	12	11.6	42694
64	(6, 40)	39.07	34.16	39808	(8,6,45)	21.4	19.8	37287
116	(6, 36)	62.07	55.56	36704	(8,6,38)	33.6	30	33068
255	(6 , 32)	119.07	108.50	33408	(6,6,35)	80	76.3	29596

Table 10: Comparison between the communication cost of Banquet and the new protocol for AES-192. Picnic for the same security level reports $t_{\mathcal{S}} = 11.01ms$, $t_{\mathcal{V}} = 8.49ms$ and size 27405 bytes.

N	Banquet AES-256				Limbo-Sign AES-256			
	(ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes	(k, ℓ, τ)	$t_{\mathcal{S}}$ (ms)	$t_{\mathcal{V}}$ (ms)	size bytes
16	(4, 84)	27.63	21.54	83488	(8, 6, 82)	11.3	9.9	83764
31	(6, 63)	37.67	31.77	73114	(8,6,70)	19.1	17.5	73788
62	(6, 54)	60.71	53.47	64420	(8,6,58)	36.2	30.6	63044
119	(6, 48)	100.41	90.58	58816	(8,6,52)	58.2	54.2	58216
256	(6, 43)	190.73	174.54	54082	(6,8,46)	125	117	53004

Table 11: Comparison between the communication cost of Banquet and the new protocol for AES-256.Picnic for the same security level reports $t_{\mathcal{S}} = 18.82ms$, $t_{\mathcal{V}} = 13.56ms$ and size 48437 bytes.

the scheme works as follows. Given a private key $\mathtt{k}$ and public values (x, y), such that $\mathtt{AES}_{\mathtt{k}}(x) = y$, a signature on a message μ is generated by binding together μ with a non-interactive zero-knowledge proof of knowledge of $\mathtt{k}$.

We compare our resulting signature scheme with Picnic and Banquet, which is, as far as we know, the fastest MPCitH-based signature using AES, for security levels L1,L3,L5 as specified by NIST [oST20]. In Tables 9, 10 and 11 we show this comparison for different sets of parameters. Across all three security levels, Limbo not only provides a significant speed-up over Banquet in both Prover and Verifier running time but also produces consistently shorter signatures. These are also much closer to (and sometimes better than) the performance of Picnic both in running times (for $n = 16$) and in signature size (for $n = 255$).

In Table 8 we compare Limbo used as an interactive identification scheme with the equivalent variant of Picnic. Note that for Picnic, which is based on KKW, when we show the number of repetitions we consider the total number of repetitions and the online executions, because the underline MPC protocol is in the preprocessing model. As before, using more parties, this communication can be further reduced (down to 2.43KB) at the expense of slightly longer computation time (still under 5ms for both Prover and Verifier).

Acknowledgements

We thank Daniel Kales for the implementation code of Banquet and Greg Zaverucha for suggestions in relation to the work in this paper. This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) under contract No. HR001120C0085, and by CyberSecurity Research Flanders with reference number

VR20192203. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ERC, DARPA, the US Government or Cyber Security Research Flanders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

AHIV17. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

ARS$^+$15. Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.

BBC$^+$17. Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear PCPs. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 551–579. Springer, Heidelberg, April / May 2017.

BBC$^+$19. Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97. Springer, Heidelberg, August 2019.

BBHR19. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, Heidelberg, August 2019.

BCC88. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

BCGT13. Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 585–594. ACM Press, June 2013.

BCI$^+$13. Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Heidelberg, March 2013.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

Bd20. Ward Beullens and Cyprien de Saint Guilhem. LegRoast: Efficient post-quantum signatures from the Legendre PRF. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 130–150. Springer, Heidelberg, 2020.

BdSGK$^+$21. Carsten Baum, Cyprien Delpech de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. *IACR Cryptol. ePrint Arch.*, 2021:68, 2021.

BFH$^+$20. Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

BGIN19. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 869–886. ACM Press, November 2019.

BMN18. Alexander R. Block, Hemanta K. Maji, and Hai H. Nguyen. Secure computation with constant communication overhead using multiplication embeddings. In Debrup Chakraborty and Tetsu Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 375–398. Springer, Heidelberg, December 2018.

BMRS20. Carsten Baum, Alex J. Malozemoff, Marc Rosen, and Peter Scholl. 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.

BN20. Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426. Springer, Heidelberg, August 2018.

CDF$^+$20. Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. Buffing signature schemes beyond unforgeability and the case of post-quantum signatures. Cryptology ePrint Archive, Report 2020/1525, 2020. https://eprint.iacr.org/2020/1525.

CDG$^+$17. Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017.

dDOS19. Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 669–692. Springer, Heidelberg, August 2019.

DIO20. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. Cryptology ePrint Archive, Report 2020/1446, 2020. https://eprint.iacr.org/2020/1446.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

GGPR13. Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

GMR85. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th ACM STOC*, pages 291–304. ACM Press, May 1985.

GS20. Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority MPC. Cryptology ePrint Archive, Report 2020/134, 2020. https://eprint.iacr.org/2020/134.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

Kil92. Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

Kil95. Joe Kilian. Improved efficient arguments (preliminary version). In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 311–324. Springer, Heidelberg, August 1995.

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.

KR08. Yael Tauman Kalai and Ran Raz. Interactive PCP. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 536–547. Springer, Heidelberg, July 2008.

LDK$^+$20. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

Mer90. Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238. Springer, Heidelberg, August 1990.

MHA. William Woodruff Mathias Hall-Andersen, Ben Perez. https://github.com/trailofbits/reverie.

Mic94. Silvio Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

oST20. National Institute of Standards and Technology. Round 3 submissions for the nist post-quantum cryptography project. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, accessed 11-11-2020, 2020.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996.

RRR16. Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In Daniel Wichs and Yishay Mansour, editors, *48th ACM STOC*, pages 49–62. ACM Press, June 2016.

WYKW20. Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020. https://eprint.iacr.org/2020/925.

YSWW21. Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field, 2021.

ZCD$^+$20. Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

A Amortized Evaluations for the Multi-instance case

While our MPC and ZK protocols work over fields of any size, the multiplications check requires large fields to obtain a reasonable soundness error. So, when the evaluation field $\mathbb{F}$ is small, for example when $\mathbb{F} = \mathbb{F}_2$, this step seems to be wasteful. For this reason, it would be convenient to batch several checks into a single one. Ideally, when we prove the satisfiability of a certain circuit, it would be helpful to perform the check of all the τ repetitions needed to obtain the desired soundness, in one go. Unfortunately, since in the ZK protocol the verifier opens different sets of parties across the τ MPC evaluations, packing these checks together seems difficult, if not impossible. However, we can apply the same idea to batch together the checking phases in the case of multiple evaluations of the same circuit.

More precisely, if we want to prove satisfiability of a certain circuit multiple times, say h, we can amortized these instances using the reverse multiplication-friendly embedding (RMFE) [BMN18,CCXY18], which provides a way to embed the ring $\mathbb{F}_q^h$, for some $h > 1$, into a field $\mathbb{F}_{q^s}$, for some $s > h$, so that coordinate-wise products "map" to multiplications in the extension field. More formally, we recall the following definition.

Definition 9. *Given a prime power q and $h, s \in \mathbb{N}$, let us consider two $\mathbb{F}_q$-linear maps $\phi : \mathbb{F}_q^h \to \mathbb{F}_{q^s}$, and $\psi : \mathbb{F}_{q^s} \to \mathbb{F}_q^h$. A pair $(\phi, \psi)_q$ is called an (h, s)-*reverse multiplication-friendly embedding (RMFE) *if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{F}_q^h$ it holds:*

$$\mathbf{x} \odot \mathbf{y} = \psi(\phi(\mathbf{x}) \cdot \phi(\mathbf{y})),$$

where $\odot$ is the component-wise product.

Note that ϕ is an injective map. In the following, we only focus on the case $q = 2$ and leave other cases to future works. In [CCXY18], the authors give both asymptotic and concrete results on the existence of RMFE. In particular:

Lemma 4. *For all $u \leq 33$, there exists a $(3u, 10u - 5)_2$-RMFE. For any $u \leq 16$, there exists a $(2u, 8u)_2$-RMFE.*

Different Options to Improve Efficiency We explore different options in order to deal with the multi-instance case with better efficiency. For each of these alternatives we briefly discuss advantages and disadvantages.

Check with identical challenges. The first approach simply consists of $\tau \cdot h$ MPC evaluations of the circuit over $\mathbb{F}_2$, followed by a checking phase. We can apply the optimization described in the previous section and use the some challenge across all the evaluations. This option will require, at least in the non-interactive case, bigger τ, but this can be mitigated in part by using larger extension fields for the check. We expect in this case an improvement in prover run times comparable to that observed for the single instance case.

Evaluations in extension fields. Alternatively, we can use a RMFE and "pack" h MPC evaluations over $\mathbb{F}_2$ into a single evaluation over $\mathbb{F}_{2^s}$, and hence perform the entire proof and verification in this extension field. The advantage of this approach is to perform the computation only one, but over a larger field. In term of communication this approach will be roughly 2/3 times more costly.

Check in extension fields. Our third option works as follows. In Phase 1, the prover runs h MPC evaluations over $\mathbb{F}_2$ in its head, exactly as described in the previous sections. Before the next phases, $\mathcal{P}$, using a $(h, s)_2$ (ϕ, ψ)-RMFE, consistently maps all the $h \cdot m$ multiplication triples in $\mathbb{F}_2$ that need to be checked to m triples in $\mathbb{F}_{2^s}$, and proceeds to the next phases. In more details, given $\langle \mathbf{x}_i \rangle$, $\langle \mathbf{y}_i \rangle$, $\langle \mathbf{z}_i \rangle \in \mathbb{F}_2^h$, we can apply ϕ to these m vectors and obtains $\phi(\mathbf{x}_i) = x_i, \phi(\mathbf{y}_i) = y_i, \phi(\mathbf{z}_i) = z_i$. Note that if $\mathbf{x} \odot \mathbf{y} = \mathbf{z} \odot \mathbf{1}$, then

$$\psi(\phi(\mathbf{x}_i) \cdot \phi(\mathbf{y}_i) - \phi(\mathbf{z}_i) \cdot \phi(\mathbf{1})) = 0. \tag{2}$$

Setting $z_i \cdot \phi(\mathbf{1}) = \xi_i$, the prover needs to prove that the relation above holds. The analysis of the soundness of this option differs from that done in the previous sections, so we leave it for future works. However, we note that, if the maps ϕ, ψ are efficiently implemented, this approach can lead potentially to better prover run times.