Fully Secure MPC and zk-FLIOP Over Rings: New Constructions, Improvements and Extensions

Anders Dalskov[1], Daniel Escudero[2], and Ariel Nof[3]

[1] Partisia
anderspkd@fastmail.com
[2] JP Morgan AI Research & JP Morgan AlgoCRYPT CoE
daniel.escudero@protonmail.com
[3] Bar-Ilan University
ariel.nof@biu.ac.il

Abstract. We revisit the question of the overhead to achieve full security (i.e., guaranteed output delivery) in secure multiparty computation (MPC). Recent works have closed the gap between full security and semi-honest security, by introducing protocols where the parties first compute the circuit using a semi-honest protocol and then run a verification step with sublinear communication in the circuit size. However, in these works the number of interaction rounds in the verification step is also sublinear in the circuit's size. Unlike communication, the round complexity of the semi-honest execution typically grows with the circuit's *depth* and not its size. Hence, for large but shallow circuits, this additional number of rounds incurs a significant overhead. Motivated by this gap, we make the following contributions:

1. We present a new MPC framework to obtain full security, compatible with effectively *any* ring, that has an additive communication overhead of only $O(\log |C|)$, where $|C|$ is the number of multiplication gates in the circuit, and a *constant* number of additional rounds beyond the underlying semi-honest protocol. Our framework works with any linear secret sharing scheme and relies on a new to utilize the machinery of *zero-knowledge fully linear interactive oracle proofs* (zk-FLIOP) in a black-box way. We present several instantiations to the building blocks of our compiler, from which we derive concretely efficient protocols in different settings.
2. We present extensions to the zk-FLIOP primitive for very general settings. The first one is for proving statements over potentially non-commutative rings, where the only requirement is that the ring has a large enough set where (1) every element in the set commutes with every element in the ring, and (2) the difference between any two distinct elements is invertible. Our second zk-FLIOP extension is for proving statements over Galois Rings. For these rings, we present concrete improvements on the current state-of-the-art for the case of constant-round proofs, by making use of *Reverse Multiplication Friendly Embeddings* (RMFEs).

1 Introduction

Secure multiparty computation (MPC) [BGW88; CCD88; GMW87; Yao86] enables a set of n parties $P_1, \ldots, P_n$ to jointly compute a function $f(x_1, \ldots, x_n) = y$ over their private inputs such that only y—the output—is revealed. MPC protocols thus provide a general-purpose method for privacy-preserving computation on sensitive data. As a result, a lot of effort has been undertaken over the years in order to increase the efficiency of MPC protocols.

Security of an MPC protocol is categorized according to the capabilities of an *adversary* that is assumed to control some $t < n$ of the parties. If the adversary is assumed to follow the steps prescribed by the protocol, then the adversary is said to be *semi-honest*; otherwise, the adversary is *malicious*, and can behave in an arbitrary manner. In the presence of a malicious adversary, it is relevant to ask whether "denial-of-service" attacks are permitted.[1] A protocol that is secure against a malicious adversary that is only allowed to perform such attacks, is termed *secure with abort*. On the other hand, if the protocol permits no such attacks—that is, it guarantees termination—then we say the protocol is *fully secure*. While full security is the ultimate goal and the higher level of security, it can only be achieved (for general functions) when an honest majority exists. i.e., if $t < n/2$ [Cle86].

From semi-honest to full security. A useful metric to measure the efficiency of MPC protocols is the ratio between the cost of semi-honest security—which is the minimal level of security one can hope for and is usually easy to achieve—and the cost of malicious security, which is the more realistic model and much more challenging to obtain. Indeed, many protocols follow the design paradigm of starting with a protocol secure against semi-honest adversaries and "compile" the protocol into malicious security. This methodology has turned out to lead to particularly efficient protocols, measured in terms of the overhead that the compilation process adds. The last three decades are rife with research that follows this particular strategy, all of which aim at reducing this overhead; see [BFO12; Chi+18; Cra+18; Dam+12; FL19; Fur+17; GIP15; GMW87; GSZ20; IPS08; LN17] as only a partial list of references.

A natural question that arises, therefore, is whether it is possible to close the gap between semi-honest security and malicious security.[2] More precisely, can we obtain a maliciously secure protocol with a cost matching that of the best corresponding semi-honest protocol, at least when the circuit is large enough? This question remained open until 2019 when Boneh et al. [Bon+19] introduced information-theoretic distributed zero-knowledge proofs over secret shared data (DZKP), based on *zero-knowledge fully linear interactive oracle proofs* (zk-FLIOP).

[1] Such attacks are assumed to not compromise privacy or correctness. They would merely prevent the protocol from terminating.

[2] Without introducing any new assumption but making a black-box use of a pseudorandom generator (PRG). This rules out expensive cryptographic tools such as fully homomorphic encryption [Gen09], where communication is asymptotically small, but the computational costs are very high.

This primitive allows a prover to prove the correctness of a statement over a secret input which is shared across a set of verifiers. Boneh et al. [Bon+19] showed that, when the statement is a *degree-2 computation* and the input is shared across the verifiers via a *linear secret sharing scheme* in a *robust* way, then there exist distributed zero-knowledge proofs with communication that is sublinear in the size of the input. This primitive opened the door to maliciously secure MPC protocols that work in the following way: first run a semi-honest protocol—thereby providing privacy—and then run a lightweight verification step using the distributed zero-knowledge proof machinery, where the computation is being checked for correctness—thereby providing security against tampering. Because the verification step only requires communication sublinear in the size of the circuit being computed, the overall communication cost of the malicious protocol is the same (in an amortized sense) as that of the semi-honest base protocol. The key observation made, that allows applying a distributed zero-knowledge proof in MPC, was that the computation done by the parties in many semi-honest protocols can be seen as degree-2 computations over inputs that are robustly secret-shared. With the introduction of DZKP, Boneh et al. [Bon+19] were the first to obtain malicious security with abort at the same (amortized) cost of semi-honest security. Follow-up works have built on the techniques from [Bon+19] to obtain full security in the honest majority setting [Boy+19; Boy+20; GSZ20]; as well as malicious security in the dishonest majority setting [Boy+20; Boy+21].

Despite this success, some gaps remain open. Existing works have focused on reducing the passive-to-active overhead in terms of *communication*, achieving an additive overhead that grows logarithmically with the size of the circuit. This has been shown to lead to concrete benefits in certain practical settings, *e.g.* [Goy+21]. However, there are other efficiency metrics of interest besides communication complexity for which the passive-to-active overhead question is appropriate. Of particular interest to us is *round complexity*: the number of interactive rounds required to finish the protocol. MPC is a highly distributed application, requiring several sequential message exchanges among the parties. For settings where the number of parties is large, and the network latency is not too fast (*e.g.* WAN), minimizing the number of rounds becomes vital for efficiency—even more so than minimizing communication bandwidth. Unfortunately, for round complexity even a logarithmic additive overhead can be extremely detrimental for efficiency: for passive security the number of rounds grows only with the circuit's *depth*, so if the circuit to be computed is very large but shallow, this logarithmic term can result in adding a number of rounds bigger than the circuit's depth, thereby damaging the overall efficiency. Sadly, with the current state-of-affairs for full security in the honest majority setting, such overhead in round complexity is present in all previous works. The only exception is the protocol of Boyle et al. [Boy+19], which achieves an overhead of logarithmic communication and a constant number of rounds. However, their protocol is specifically designed for the 3-party setting, relying on the semi-honest protocol [Ara+16] designed for this particular setting. This therefore raises the question:

Can full security for an arbitrary number of parties be achieved while incurring in the following additive overheads with respect to state-of-the-art semi-honest protocols: (1) communication overhead that is logarithmic in the circuit size, and (2) ***constant*** *overhead in the round complexity?*

We remark that for security with abort, this question has been solved in [Boy+20]. Unfortunately, and as we explain below, current approaches do not enable us to move from security with abort to full security without incurring in an additional logarithmic overhead for the round complexity, which is what we seek to avoid. Obtaining this goal will lead to highly efficient protocols with full security, particularly for big, shallow circuits and WAN settings.

1.1 Our Results

In this work, we make two main contributions. We provide new and improved constructions to achieve MPC with full security and present improvements to the underlying zk-FLIOP primitive.

Fully secure MPC with sublinear communication and constant round complexity overhead. We present a new MPC framework to lift semi-honest secure protocols to full security when computing any arithmetic circuit C, relying on zk-FLIOPs in a black-box way and with statistical security. Crucially, our compiler works with *any* linear secret sharing scheme, and only assumes the existence of a protocol to convert a sharing of a random secret between two thresholds, which is called only *once* in the entire execution. Furthermore, our compiler supports arithmetic circuits defined over *any* ring R (even non-commutative!). We believe that such level of generality is essential to understand what are the core requirements on the building blocks involved in order to achieve efficient general MPC. In addition, there are non-field rings such as $\mathbb{Z}_{2^k}$ that have been proven to be particularly useful in practice (see e.g. [Sto+23] and the reference therein), and these rings are encompassed in our framework.

We then present three concrete instantiations of our framework, where the *online* communication cost is $|\mathsf{semi\text{-}honest}| + O(n \cdot \log(|C|))$ and the number of rounds is $|\mathsf{depth}(C)|$ (in the random oracle model), thus closing the gap between semi-honest and full security both in terms of communication and round complexity:

– A protocol for *any* linear secret sharing scheme and any ring, with $O(n^2|C|)$ preprocessing (done in a single round). For instance, we can compile the semi-honest Shamir-based protocol by Escudero and Soria-Vazquez [ESV21], which works for any ring to full security. The online phase has the same cost as semi-honest (amortized), both in terms of communication and in terms of number of rounds.

– A protocol relying on replicated secret sharing [ISN89] and any ring with *silent preprocessing* (that is, after a short setup phase independent of $|C|$, the offline phase is non-interactive). As the share size in this scheme grows exponentially with the number of parties, this protocol suits for computations with a small number of parties (*e.g.*, $n < 10$).

– A protocol for any linear secret sharing scheme, any ring, and any preprocessing, but with a two-thirds honest majority $t < n/3$. This protocol can be used for example with Shamir's secret sharing [Sha79] and linear preprocessing [DN07], and so scales with the number of parties. Interestingly, using our framework, we obtain a new improved solution to the so-called "double-dipping" attack [GLS19] which occurs in the $t < n/3$ setting. As opposed to previous solutions [Fur+17; GLS19], our solution does not require changing the underlying semi-honest protocol.

Our results improve over prior works in two different dimensions. First, we introduce a new verification protocol with a constant number of rounds, which, compared to prior works [Boy+20; GSZ20], allows easy identification of cheaters once cheating has been detected. For this we make *black-box* use of zk-FLIOPs over general rings, allowing our framework to immediately benefit by improvements in the zk-FLIOP literature. Second, our framework applies to any finite ring, which may not be necessarily commutative. In contrast, the prior work of [Boy+20] only supports fields and rings of the form $\mathbb{Z}_{2^k}$, while [GSZ20] only supports fields and moreover it is *not* black-box in terms of the zk-FLIOP. Only [ESV21] studies general rings as we do in the honest majority setting, making use of Shamir secret-sharing. In contrast, we rely on any linear secret sharing with reduced communication complexity and achieve the stronger notion of full security, while [ESV21] only achieves security with abort. Our protocol makes use of point-to-point secure channels, as well as a broadcast channel (necessary to achieve full security in this setting, where broadcast is not possible without setup [PSL80]). Fortunately, the number of calls to the broadcast channel is also sublinear in the size of the circuit, and so implementing it using digital signatures [DS83], would give the same amortized cost over the point-to-point channels alone.

In Table 1, we compare our result with all previous works that require only *additive communication cost* beyond the state-of-the-art semi-honest protocol in their setting. As can be seen from the table, the only two works that achieve both sublinear additive communication and constant number of additional rounds are [Boy+19] and [DEN22]. However, [Boy+19] considers only three parties, whereas the protocol of [DEN22] is in the two-thirds honest majority setting and relies on specific properties of replicated secret sharing, implying that the protocol is only concretely efficient for a small number of parties.

While our work provides significant improvements upon state-of-the-art, one gap remains: suppose one wishes to work with Shamir's secret sharing when $t < n/2$ and n is large. In this setting, there exists a preprocessing protocol with linear communication complexity. Our framework can work with this preprocessing only when $t < n/3$ and not $t < n/2$. Insisting on constant round overhead and $t < n/2$ leads to a preprocessing of $O(n^2|C|)$. Hence, the question of achieving sublinear additive communication cost and constant number of additional rounds with *linear* preprocessing for $t < n/2$ and large n remains open.

	(n,t)	Additive Overhead		Security	Secret sharing scheme
		Communication cost	No. of Rounds		
Boneh et al. [Bon+19]	(3,1)	$O(\log\|C\|)$	$O(1)$	with abort	replicated
Boyle et al. [Boy+19]	(3,1)	$O(\log\|C\|)$	O(1)	Full	replicated
Boneh et al. [Bon+19]	$(2t+1,t)$	$O(\sqrt{\|C\|})$	$O(1)$	with abort	replicated
Boyle et al. [Boy+20]	$(2t+1,t)$	$O(\log\|C\|)$	$O(1)$	with abort	Any linear scheme
Boyle et al. [Boy+20]	$(2t+1,t)$	$O(\log^2\|C\|)$	$O(\log\|C\|)$	Full	replicated
Goyal et al. [GSZ20]	$(2t+1,t)$	$O(\log\|C\|)$	$O(\log\|C\|)$	Full	Shamir
This work	$(2t+1,t)$	$O(\log\|C\|)$	$O(1)$	Full	Any linear scheme, $O(n^2\|C\|)$ preprocessing
This work	$(2t+1,t)$	$O(\log\|C\|)$	$O(1)$	Full	replicated
Furukawa et al. [FL19]	$(3t+1,t)$	$O(1)$	$O(\|\mathsf{depth}(C)\|)$	with abort	Shamir
Dalskov et al. [DEN22]	$(3t+1,t)$	$O(1)$	$O(1)$	Full	Replicated
This work	$(3t+1,t)$	$O(\log(\|C\|))$	$O(1)$	Full	Any linear scheme

Table 1: Comparison between works that achieve malicious security with only sublinear additive communication overhead. Note that $|C|$ is the size of the circuit C, measured by the number of multiplication gates, n is the number of parties and t is the number of corrupted parties. In the 'communication cost' column we ignore terms that depend on n and are the *same* for all works. In addition, over rings with a small exceptional set (see Section 2), the communication cost is multiplied by the security parameter in all rows.

Extending and improving zk-FLIOPs over general rings. Our MPC protocol essentially works for any finite ring R, even possibly non-commutative, assuming zk-FLIOP for such rings exists. However, current instantiations [Bon+19] are known only over finite fields or over the ring $\mathbb{Z}_{2^k}$. Moreover, the complexity of zk-FLIOP over $\mathbb{Z}_{2^k}$ is increased by a multiplicative factor depending on the statistical security parameter, which does not occur over finite fields (when the field is large enough). The increase in the complexity is because a ring extension of degree proportional to the security parameter has to be used. It therefore follows that, even if the communication is still technically sublinear, the concrete cost is much larger (compared to the finite field case) as the size of every element is multiplied by a factor proportional to the statistical security parameter.

We improve the current state of affairs by presenting two instantiations of zk-FLIOPs for two very large families of rings. We present a zk-FLIOP system which works for *any* finite and possibly non-commutative ring R, where the only requirements are that there is a large enough subset $\mathbb{A} \subseteq R$ such that (1) for all $x, y \in \mathbb{A}$, $x \neq y$, $x - y$ is invertible in R, and (2) every element of $\mathbb{A}$ commutes with every element in R. We refer the reader to Section 5.1 for details. Our second instantiation for a zk-FLIOP system works for certain family of commutative rings, namely *Galois Rings*, which constitute a generalization of both finite fields $\mathbb{F}_{p^d}$ and the ring $\mathbb{Z}_{p^k}$. In particular, our interest lies on Galois rings for which the maximal exceptional sets are not large enough, which contain as particular relevant cases the field $\mathbb{F}_2$ and the ring $\mathbb{Z}_{2^k}$, considered already by prior works. In this case, previous zk-FLIOP constructions such as [Bon+19] require a large Galois ring extension, resulting in communication which incurs in the aforementioned security parameter overhead. We show in our work how to make use of the technique of *Reverse Multiplication Friendly Embeddings* (RMFEs) [Cas+18] in order to obtain concrete efficiency gains for zk-FLIOP over Galois rings. RMFEs are a useful tool to map multiple products over a Galois ring $\mathsf{GR}(p^k, d)$ with constant p^d, into a larger extension $\mathsf{GR}(p^k, d \cdot m)$, without paying a cost that grows with m. They were introduced over fields by Cascudo et al. [Cas+18], and extended to general Galois rings in [CRX21]. Since their inception, they have been used in multiple works in both the MPC setting [Cas+18; CG20; CRX21; EXY22; PS21] and also zero-knowledge [CG22; KZ22], to improve efficiency over rings like $\mathbb{F}_2$ or $\mathbb{Z}_{2^k}$. We add our work to this list of applications of RMFEs by using these objects to improve the communication complexity of FLIOPs over Galois rings. Interestingly, ours is technically an improvement on the zero-knowledge side, which has direct applications to MPC, so it can be seen as an improvement in both fronts. We refer the reader to Section 5.2 for details.

1.2 High-Level Technical Overview

Overview of our MPC Protocol. As explained above, zk-FLIOP is a cryptographic primitive where a prover wishes to prove that a secret x is in a language $\mathcal{L}$ to a verifier V. In each round of the protocol, the prover outputs a proof, and then the verifier is allowed to make only linear queries to the input and proof. The queries are chosen based on a random public challenge. At the end, the verifier accepts or rejects based on the queries' answers. This abstract primitive can be realized in the setting where x is distributed across a set of verifiers, say via a linear secret-sharing scheme. The prover can thus share the proof to the verifiers via the same scheme and then each verifier runs the queries locally over its shares of proof and input. At the end, the verifiers reconstruct the shared answers and run the decision predicate.

Boneh et al. [Bon+19] showed that when the statement to be proven is a degree-2 computation over an input that is shared in a robust way, then there exists a protocol where the communication is sublinear in the input size. When we say robust, we mean that shares held by the honest parties suffice to determine the secret and the other shares. This is turned out to be useful for verifying correctness in MPC, as typically the statement to be verified consists of many equations of the form $z_k - x_k \cdot y_k$, where x_k, y_k and z_k are the inputs and the output of the kth multiplication gate respectively. Note that this is a degree-2 computation and all values are shared across the parties. The only difficulty here is that no one knows the secrets, and hence no one can play the role of the prover. The solution taken in [Bon+19; Boy+22] is to let all parties also emulate the role of the prover. Hence, this approach is called the "distributed-prover" approach. While this approach leads directly to malicious security with abort, achieving full security requires solving one more problem. Specifically, to obtain full security, it is necessary to identify cheaters. However, when the proof is rejected, the parties only know that for some k, $z_k - x_k \cdot y_k \neq 0$. At a high level, the approach taken in both [Boy+20] and [GSZ20] is to recursively iterate over the set of multiplication triples, until locating a single incorrect triple, and then apply some constant-cost protocol to analyze this triple and find the cheater. This is the reason for the $\log(|C|)$ factor in the round complexity, which we want to remove.

A different approach was taken in the 3-party protocol of [Boy+19]. Instead of proving correctness for the values on the output wires of each multiplication gate, the authors let each party prove that it sent the correct message in the semi-honest multiplication protocol. We call this approach the "single-prover" approach. The advantage of this approach is that once the proof is rejected, we can blame the prover and there is no need for running a recursive search as before. This works perfectly for the 3-party semi-honest protocol of [Ara+16], as in this protocol, each party sends one message to another party, which is computed as a degree-2 computation over shared inputs that are known entirely by the prover (this property is unique to replicated secret sharing used in that work).

Technically, our goal is therefore to use the single-prover approach in the n-party setting. However, applying this approach to the state-of-the-art multiplication protocol, the DN protocol [DN07] raises multiple problems. Let us first recall how this popular protocol works. Denote by $[\![x]\!]$ a robust t-out-of-n sharing of x and by $\langle x\rangle$ an additive sharing of x. In the DN protocol, the parties preprocess a pair $[\![r]\!], \langle r\rangle$ for some random r for each multiplication gate. In the online protocol, for the kth multiplication gate, the parties locally compute $[\![x_k]\!] \cdot [\![y_k]\!] - \langle r_k\rangle$, send the result to P_1, who reconstructs $x_k \cdot y_k - r_k$ and send back the result to the parties. Finally, the parties locally compute $[\![x_k \cdot y_k]\!] = [\![r]\!] + x_k \cdot y_k - r_k$. Our main contribution is designing a verification protocol for this protocol, where each party proves it sent the correct message. If the verification fails, then the protocol's output is a semi-corrupt pair, which is a pair of parties with the guarantee that one of them is corrupt. Given this pair, the parties can run a recovery protocol to remove these parties from the computation. Towards achieving this goal, we briefly outline some of the challenges we had to address:

- The second-round message of P_1 is not a degree-2 computation over robustly shared inputs. We show that it is actually not necessary to verify this message. Instead, we let the parties first reach an agreement on a "compressed" transcript, from which they can derive the *implicit* first round message of P_1 and verify this message. Hence, the parties need to prove correctness for first-round messages only.
- The first-round message is a degree-2 computation. However, while $[\![x_k]\!]$ and $[\![y_k]\!]$ are robustly shared, this is not the case for $\langle r_k\rangle$. We thus need to convert it to a robust secret sharing. We characterize this conversion and show that it suffices to run it once for the entire execution, over a random linear combination of all r_ks. Then, we present different implementations for the conversion in different settings and trade-offs. Since it is called only once, we can afford also expensive realizations.
- Typically, in a setting with a single prover, the prover knows the entire input, including the shares held by the verifiers. This indeed was the case considered in [Bon+19] and [Boy+20]. However, we deal with a different setting: given a

robust sharing of $[\![x]\!]$, each party proves it performed some computation over its *shares* of x, without knowing the shares held by other parties. This implies that in our distributed zero-knowledge proof, we need to maintain privacy also against the prover and not only against the verifiers. We present a new simple transformation from zk-FLIOP to a distributed zero-knowledge proof for this particular setting, which we later use in our verification protocol. We also show that the protocol can be made non-interactive in the random oracle model. This construction may be of independent interest.

– While the queries' answers in the emulation of the zk-FLIOP are shared in a robust way, this only means that corrupted parties cannot change the shared secret. However, they can cause the reconstruction to fail due to inconsistency. We thus need to develop a mechanism to locate a semi-corrupt pair when an opened secret could not be reconstructed. We show that due to the linearity of the queries, each answer can be written as a sum of n sharings, each dealt by one party. Given this decomposition, we present a protocol that outputs a semi-corrupt pair.

Finally, as mentioned earlier, another side contribution of this paper is a new solution to the so-called "double-dipping" attack on DN-style protocols [GLS19]. This attack takes place in the strong honest majority setting. We show that using our framework, we do not need to change the DN protocol, as opposed to previous works in this domain [FL19; GLS19]. This is described in Section 4.3.

Overview on our zk-FLIOP contribution. Showing that existing techniques in the instantiations of [Bon+19] can be extended to larger class of rings is done in a similar way to [ESV21] by noticing that polynomial interpolation (and hence notions like Reed-Solomon codes or the Schwartz-Zippel Lemma) holds over more general rings than fields, such as the ones mentioned above. This is discussed in Section 5.1. Now, to understand how our improvements for Galois rings work, let us consider a prover that wishes to prove the relation $\{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}) \mid \boldsymbol{x} \star \boldsymbol{y} = \boldsymbol{z}\} \subseteq (\mathbb{Z}_{2^k}^M)^3$, which is a particular case of the languages we consider in this work (recall that this is what we need for verifying the computation of *any* arithmetic circuit). Our goal is to design a zk-FLIOP for this language, and we focus for simplicity on constant round proofs during this section. This can be already achieved from the results in [Bon+19], which supports statements over $\mathbb{Z}_{2^k}$ by embedding this ring into a larger Galois ring extension of degree $m \approx \kappa$ that has the desired properties. The resulting zk-FLIOP has a proof length of $O(\kappa\sqrt{M})$ (there is a $\log M$ variant with more rounds, we focus on the $\sqrt{M}$ approach with $O(1)$ rounds). The main downside of this construction is that it represents every element in $\mathbb{Z}_{2^k}$ as a single Galois ring element, which is much larger and is in fact the origin for the multiplicative term κ, which does not appear when working over large fields. This is precisely what we address in our work, and even though we do not remove the κ term, we lower $\sqrt{M}$ to $M^{1/3}$.

The main idea behind our techniques consists of making use of RMFEs to represent a batch of $\Omega(m)$ elements over $\mathbb{Z}_{2^k}$ with a single Galois ring element. This has proven to be a successful approach to remove similar $\times\kappa$ overheads in other MPC and ZK contexts [CG22; EXY22]. Let $S = \mathsf{GR}(p^k, m)$, and let (ϕ, ψ) be an RMFE with $\phi : \mathbb{Z}_{2^k}^\ell \to S$ and $\psi : S \to \mathbb{Z}_{2^k}^\ell$, which means that $\boldsymbol{x} \star \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot_S \phi(\boldsymbol{y}))$ for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_{2^k}^\ell$. Further, ϕ is injective, ψ is surjective, and we can take the RMFE such that $\psi \circ \phi$ is the identity. The existence of RMFEs with a constant rate, *i.e.*, m/ℓ is constant, has been shown in [CRX21].

Our goal is to provide a zk-FLIOP for proving that a set of triples (x_i, y_i, z_i) satisfies $x_i y_i = z_i$ for $i \in [M]$. Assume for simplicity that $\ell \mid M$, and let us use the notations $\boldsymbol{x}_j = (x_{(j-1)\ell+1}, \ldots, x_{j\ell}) \in \mathbb{Z}_{2^k}^\ell$ for $j \in [M/\ell]$, and similarly for the y's and z's. Due to the properties of RMFEs, proving that $\boldsymbol{x}_j \star \boldsymbol{y}_j = \boldsymbol{z}_j$ for all j's is equivalent to proving $\psi(\phi(\boldsymbol{x}_j) \cdot \phi(\boldsymbol{y}_j)) = \psi(\phi(\boldsymbol{z}_j))$, or put differently that $\psi(\phi(\boldsymbol{x}_j) \cdot \phi(\boldsymbol{y}_j) - \phi(\boldsymbol{z}_j)) = \boldsymbol{0}$. This is the same as saying that $h_j = \phi(\boldsymbol{x}_j) \cdot \phi(\boldsymbol{y}_j) - \phi(\boldsymbol{z}_j) \in S$ is in the kernel of ψ.

Now, if ψ was injective, this would mean that $h_j = 0$ for $j \in [M/\ell]$, and we could apply the results from [Bon+19] to the triples $\{(\phi(\boldsymbol{x}_j), \phi(\boldsymbol{y}_j), \phi(\boldsymbol{z}_j))\}_{j=1}^{M/\ell}$, which leads to a proof size of $O(\kappa \cdot \sqrt{\frac{M}{\ell}})$, or $O(\sqrt{\kappa M})$ once we factor in the fact that $m \approx \kappa$ and $m = \Theta(\ell)$. Unfortunately, this approach does not work since ψ is not injective, so each h_j may not be identically zero. However, as we will see, we can still leverage the fact that each h_j is supposed to be in $\ker(\psi)$ for improving over the naive approach.

Let $\{\beta_1, \ldots, \beta_{m-\ell}\} \subseteq S$ be a basis of $\mathsf{ker}(\psi)$, seen as a $\mathbb{Z}_{2^k}$-module. This way, the claim $h_j \in \ker(\psi)$ becomes equivalent to the existence of $(t_1^{(j)}, \ldots, t_{m-\ell}^{(j)}) \in \mathbb{Z}_{2^k}^{m-\ell}$ such that $h_j = \sum_{i=1}^{m-\ell} t_i^{(j)} \beta_i$. Our key observation is that this is precisely the type of statements that can be proven using the techniques in [Bon+19]! Indeed, we can consider the language of tuples $\{(\alpha_j, \beta_j, \gamma_j, (t_1^{(j)}, \ldots, t_{m-\ell}^{(j)}))\}_{j=1}^{M/\ell} \subseteq S^3 \times \mathbb{Z}_{2^k}^{m-\ell}$ such that $\alpha_j \beta_j - \gamma_j - \sum_{i=1}^{m-\ell} t_i^{(j)} \beta_i = 0$, and map our original statement to this one by taking $\alpha_j = \phi(\boldsymbol{x}_j)$, $\beta_j = \phi(\boldsymbol{y}_j)$ and $\gamma_j = \phi(\boldsymbol{z}_j)$. Unfortunately, proving this statement directly does not improve over the naive solution and in fact it makes it worse: using the zk-FLIOP from [Bon+19] the cost would be $O(\kappa\sqrt{M/\ell})$ elements over $\mathbb{Z}_{2^k}$; however, we have to append the $\{(t_1^{(j)}, \ldots, t_{m-\ell}^{(j)})\}_{j=1}^{M/\ell}$ elements to the proof, and there are $(m - \ell)(M/\ell - 1) = \Theta(M)$ such values, which is not sublinear in M.

Our final solution consists of minimizing the amount of extra elements in the proof arising from the kernel basis. This is achieved by adding an extra round to the proof where the verifier samples some challenges that will be used to compress the t elements, reducing their amount to something that depends on κ, but not on M. In a bit more detail, the verifier samples $\gamma_1, \ldots, \gamma_{M/\ell} \in \mathbb{Z}_{2^k}$ as challenges,[3] and the prover computes $h = \sum_{j=1}^{M/\ell} \gamma_j h_j$ and $t_i = \sum_{j=1}^{M/\ell} \gamma_j t_i^{(j)}$ for $i \in [m - \ell]$. Then, it should hold that $h - \sum_{i=1}^{m-\ell} t_i \beta_i = 0$, and this is precisely the statement we prove using the techniques from [Bon+19] for Galois rings. The

[3] In fact, these can be taken over $\mathbb{Z}_2$, and this is what we do in our actual construction, but this does not affect proof size.

advantage now is that there are only $m - \ell$ additional elements to the proof, in contrast to the prior $(m - \ell)(M/\ell - 1)$.

One can show that the "random linear combination" approach from above adds a term of 1/2 to the soundness error, so to keep it negligible this must be repeated κ times. As a result, the actual extra term to the final proof size is $\kappa \cdot (m - \ell) = \Theta(\kappa m)$, so the total proof size is $O(\kappa(\sqrt{M/m} + m))$. This is minimized when $m = \Theta(M^{1/3})$, which leads to a total proof size of $O(\kappa M^{1/3})$, as required. We refer the reader to Section 5 for our detailed zk-FLIOP construction, which works for more general statements and Galois rings than the ones considered here.

2 Preliminaries

Notation. Let $P_1, \ldots, P_n$ be the set of parties and let t be such that $n = 2t + 1$. In this work, we assume that an honest majority exists and so the number of corrupted parties is at most t. We use $[n]$ to denote the set $\{1, \ldots, n\}$. We denote by $\mathbb{F}$ a finite field and by R a ring. If S is a set of n elements, then we write $S_I \subseteq S$ to denote the set $\{s_i | s_i \in S, i \in I\}$ with $I \subseteq [n]$. We denote vectors with bold letters, like $\boldsymbol{x}$ or $\boldsymbol{y}$. Entries are denoted by non-bold letters with subscripts, like x_i or y_i. $\boldsymbol{x} \cdot \boldsymbol{y}$ denotes the inner product between two vectors.

2.1 Background in Ring Theory

Let R be *any* finite ring. We only assume procedures for adding and multiplying ring elements, as well as sampling uniformly random elements. A set $A \subseteq R$ is called *exceptional* if, for all $x, y \in A$ with $x \neq y$, $x - y$ is invertible.[4] The *center* of a ring R, denoted by $Z(R)$, is the ring of elements that commute with every other element of R.

For the rest of the paper, let $\mathbb{A}$ be the any of the largest exceptional subsets of R, and let $\omega_R = |\mathbb{A}|$. We will need the following lemma in our protocol.

Lemma 1. *Let $a, b \in R$, with $a \neq 0$. Then* $\Pr[x \cdot a + b = 0 | x \xleftarrow{\$} \mathbb{A}] \leq 1/\omega_R$.

Proof. Let $x, y \in \mathbb{A}$ such that $x \cdot a + b = 0$ and $y \cdot a + b = 0$, then $(x - y) \cdot a = 0$, but since $x - y$ is invertible, this implies that $a = 0$, which is a contradiction. This shows that there can be at most one $x \in \mathbb{A}$ that satisfies $x \cdot a + b = 0$, and therefore the probability of this event happening for a random sample in $\mathbb{A}$ is at most $1/|\mathbb{A}| = 1/\omega_R$. □

Observe that if R is a field then we may take $\mathbb{A} = R$, and therefore $\omega_R = |R|$. On the other hand, if R is the ring of integers modulo 2^k, it can be shown that there are no exceptional sets of size 3 or more, so we may take $\mathbb{A} = \{0, 1\}$, and hence $\omega_R = 2$.

[4] In a finite non-commutative ring, a is invertible if there exists b such that $a \cdot b = b \cdot a = 1$.

Rings with large exceptional set in center. It turns out that when ω_R is large enough, and when $\mathbb{A} \subseteq Z(R)$, then one can use polynomial interpolation over R in essentially the same way as over finite fields. This fact was used extensively in the work of Escudero and Soria-Vazquez [ESV21] to enable secure computation over arbitrary rings using Shamir secret-sharing, and we use it here to adapt distributed zero-knowledge proof techniques. The main result is the following.

Proposition 1 (Polynomial interpolation over R, Proposition 1 in [ESV21]). *Let $\{\alpha_0, \alpha_1, \ldots, \alpha_d\} = \mathbb{A}$, and suppose that $\mathbb{A} \subseteq Z(R)$. Let $y_0, \ldots, y_d \in R$. Then, there exists a unique polynomial $f(X)$ over R of degree at most d such that $f(\alpha_i) = y_i$ for every $i \in \{0, \ldots, d\}$.*

Proposition 1 is simple, but also not obvious: non-commutativity in general does not interact well with polynomials, but it turns out that the fact that the evaluation points alone do commute with the elements in R is enough to enable unique polynomial interpolation. Using this proposition, one can easily show Schwartz-Zippel, which says that a non-zero degree-d polynomial over a ring R as above cannot have more than d roots over the set $\mathbb{A}$, or in terms of probability, that a random point from $\mathbb{A}$ can only be a root of a non-zero degree-d polynomial with probability at most d/ω_R. This is Lemma 4 in [ESV21].

Galois Rings. The rings considered above are very general, but for many rings ω_R may not be large enough. For some rings, however, one may be able to extend the ring in such a way that ω_R is increased. This is indeed the case for a very useful family of rings that generalize in particular finite fields $\mathbb{F}_p$ and the ring $\mathbb{Z}_{2^k}$. These are *Galois rings*. For given $d, k \in \mathbb{N}$ and a prime p, the Galois ring of degree d with base ring $\mathbb{Z}_{p^k}$, denoted by $\mathsf{GR}(p^k, d)$, is the quotient ring $\mathbb{Z}_{p^k}[X]/(f(X))$, where $f(X)$ is a monic degree-d polynomial over $\mathbb{Z}_{p^k}$ that is irreducible over $\mathbb{F}_p$ when its coefficients are taken modulo p. Notice that $\mathsf{GR}(p^k, 1) = \mathbb{Z}_{p^k}$ and $\mathsf{GR}(p, d) = \mathbb{F}_{p^d}$. The residue ring mod p of $\mathsf{GR}(p^k, d)$ is the field $\mathbb{F}_{p^d}$, which enables $\mathsf{GR}(p^k, d)$ to behave, for many practical purposes, as the field $\mathbb{F}_{p^d}$. In particular, one can prove that $\omega_{\mathsf{GR}(p^k,d)} = p^d$.

Similarly to fields, one can consider Galois ring extensions. Given a Galois ring $R = \mathsf{GR}(p^k, d)$, one can extend it to a Galois ring $S = \mathsf{GR}(p^k, d \cdot m)$ for any m, and this structure is isomorphic to R^m as R-modules. Note that $\omega_S = \omega_R^m$, so this way, by embedding R into S, we managed to "increase" the size of the maximal exceptional set in R, at the expense of working over the larger ring S.

Reverse Multiplication-Friendly Embeddings (RMFEs). RMFEs are, in a nutshell, a technique that allows mapping a vector of ring elements unto an extension of the ring, and back again, in such a way that multiplication of elements in the extension corresponds to element-wise multiplication. In this way, an RMFE permits operating on a ring in a SIMD[5] fashion. We are chiefly concerned with RMFEs for Galois Rings, as defined above:

[5] Single Instruction, Multiple Data.

Definition 1. *Let $R = \mathsf{GR}(p^k, d)$ and $S = \mathsf{GR}(p^k, d \cdot m) \cong R^m$. A pair (ϕ, ψ) is called an $(\ell, m)_d$-RMFE if $\phi : R^\ell \to S$ and $\psi : S \to R^\ell$ are two R-linear maps satisfying*

$$\boldsymbol{x} \star \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})),$$

for $\boldsymbol{x}, \boldsymbol{y} \in R^\ell$ and where $\star$ denotes the element-wise multiplication of vectors.

A major result is that RMFEs with constant ratio, *i.e.* $\ell/d = \Theta(1)$, exist [CRX21].

2.2 Linear Secret Sharing Schemes

Definition 2 (Threshold Secret sharing schemes). *A t-out-of-n secret sharing scheme is a protocol for a* dealer *holding a secret value v and n parties $P_1, \ldots, P_n$. The scheme consists of two interactive algorithms:*

- $\mathsf{share}(v)$ *which takes a secret v and outputs shares $[\![v]\!] = (v_1, \ldots, v_n)$. The dealer runs* $\mathsf{share}(v)$ *and provides party P_i with the share v_i.*
- $\mathsf{reconstruct}([\![v]\!]_T, i)$ *which takes a subset of shares $[\![v]\!]_T$, with $T \subseteq [n]$, and outputs either v or a distinguished error symbol $\perp$ to P_i.*
  In the paper, we abuse notation and write $\mathsf{reconstruct}([\![v]\!])$*, whenever we run* $\mathsf{reconstruct}([\![v]\!], i)$ *for each $i \in [n]$.*

The scheme must ensure that no subset of t shares (or less) reveal information about v. We say that the scheme is consistent *if* $\mathsf{reconstruct}([\![v]\!]_T, i) = \mathsf{reconstruct}([\![v]\!]_{T'}, i)$ *for any two sets of* honest *parties $T, T' \subseteq [n]$ and $|T|, |T'| \geq t + 1$.*

In addition to the above, we define the following notations and procedures:

- $\mathsf{share}(v, [\![v]\!]_T)$ *takes a secret v and a set of fixed shares $[\![v]\!]_T = \{v'_j\}_{j | P_j \in T}$ where $|T| \leq t$, and outputs $[\![v]\!] = (v_1, \ldots, v_n)$, where $v_j = v'_j$ for each $j \in T$.*
- $\mathsf{complete}(\{v_j\}_{j \in T})$ *takes a set of shares of size $|T| = t + 1$ that define some secret v and compute the remaining $n - t - 1$ shares.*
- $[\![x_i|_i]\!]$ *is a consistent secret sharing of some value x where x_i is the share held by P_i.*

Local operations. We are only interested in secret sharing schemes which are *linear*, that is, secret sharing schemes for which linear operations can be performed locally. Given $[\![x]\!], [\![y]\!]$ and public constants c, d, the scheme is said to be linear if

$$d \cdot [\![x]\!] + [\![y]\!] + c = [\![d \cdot x + y + c]\!],$$

can be computed by the parties without communication.

We furthermore assume that the linear secret sharing scheme employed supports "*partial multiplication*". This means that it is possible for parties to *locally* compute an *additive* [6] secret sharing $\langle x \cdot y \rangle$ given $[\![x]\!]$ and $[\![y]\!]$. Moreover, we

[6] By *additive* secret sharing we mean the vector $\langle v \rangle = (v_1, \ldots, v_n)$ of shares where v_i is held by P_i and where $v = \sum_{i=1}^{n} v_i$.

assume that the local computation of each party involves *a degree-2 computation* over its shares. We write $[\![x]\!] \cdot [\![y]\!] = \langle x \cdot y \rangle$ to denote this operation.

Two secret sharing scheme that satisfy the above properties are the Shamir's secret sharing scheme [Sha79] and replicated secret sharing [ISN89].

2.3 The DN Semi-Honest Multiplication Protocol

As we use linear secret sharing schemes, the parties need to interact for multiplication gates only. The state-of-the-art multiplication protocol for $n > 3$ parties is an optimized variant of the DN protocol [DN07], which works as follows:

Protocol 1: Π_{mult}

- **Preprocessing**: The parties receive $[\![r]\!]$ and $\langle r \rangle$ for some random $r \in R$.
- **Input**: $[\![x]\!]$ and $[\![y]\!]$, where x and y are the values on the input wires.
- **The protocol**:
  1. The parties locally compute $\langle x \cdot y - r \rangle = [\![x]\!] \cdot [\![y]\!] - \langle r \rangle$ (see Section 2.2) and send their shares to P_1.
  2. Party P_1 reconstructs $x \cdot y - r$ and sends it to $n - t$ parties.
- **Output**: The parties locally compute $[\![x \cdot y]\!] = [\![r]\!] + x \cdot y - r$ and output the result.

Note that it suffices for P_1 to send its message to $n-t$ parties only, because the parties can now define $[\![xy - r]\!]$ by setting t shares to be 0. The, the parties simply compute $[\![r]\!]+[\![x \cdot y - r]\!]$. Overall, the communication cost is $n-1+n-t = 2n-t-1$ elements. As shown in [Goy+21], it is possible to reduce the number of rounds from 2 to 1 by slightly increasing communication. Our protocols can be easily adapted to their variant of the DN protocol.

Preprocessing. The state-of-the-art protocol to generate any number of $[\![r]\!]$ and $\langle r \rangle$ without interaction is to use the pseudorandom secret sharing (PRSS) techniques from [CDI05] (given a set of replicated keys distributed to the parties), or with interaction and linear communication complexity using Hyper-invertible matrices [DN07].

The $\mathcal{F}_{\mathsf{rand}}$ and $\mathcal{F}_{\mathsf{doubleRand}}$ ideal functionalities. Let $\mathcal{F}_{\mathsf{rand}}$ be an ideal functionality for producing $[\![r]\!]$ for a random secret r. $\mathcal{F}_{\mathsf{rand}}$ allows the ideal-world adversary to choose the corrupted parties' shares. Then, it chooses a random r and determine the honest parties' shares given r and the corrupted parties' shares. Similarly, $\mathcal{F}_{\mathsf{doubleRand}}$ generates $[\![r]\!]$ and $\langle r \rangle$, while letting adversary choose its shares.

2.4 Fully Linear Proof Systems

A main technical building block in our protocols is a *fully linear* proof system [Bon+19], which was shown to enable information-theoretic sublinear-communication zero-knowledge proofs on secret-shared input statements [Bon+19].

More concretely, we can use any public-coin *zero-knowledge fully linear interactive oracle proof*, abbreviated as zk-FLIOP. In a nutshell, a zk-FLIOP is an information-theoretic proof system in which a prover P wishes to prove a given statement about an input x to a verifier V. In each round of the protocol, P produces a proof which, together with x, can be queried by V using *linear queries* only. Then, a public random challenge is generated and the parties proceed to the next round. At the end, the verifier V accepts or rejects based on the answers it received to its queries. We will use the same notion of round as in [Bon+19]: because the first round involves only a random challenge, we subtract 0.5 from the total number of rounds. We use the below definition:

Definition 3 (Public-coin zk-FLIOP [Bon+19]). *A ρ-round, ℓ-query* public-coin fully linear interactive proof *protocol over a ring R with message lengths $(u_1, \dots, u_\rho) \in \mathbb{N}^\rho$, consists of a randomized prover algorithm* P *and a deterministic verifier algorithm* V*. Let $\boldsymbol{x} \in R^m$ be the input of* P *and let $P_0 = C_0 = \emptyset$ For each round $i \in [\rho]$:*

1. P *outputs a proof $\boldsymbol{\pi}_i \in R^{u_i}$, computed as a function of $\boldsymbol{x}$, $C_1, ..., C_{i-1}$ and its private randomness.*
2. *A random challenge C_i is chosen uniformly at random from a finite set S_i.*
3. *ℓ queries $\boldsymbol{q}_1^i, \dots, \boldsymbol{q}_\ell^i \in R^{m+u_i}$ are computed based on C_i. Then,* V *receives ℓ answers $(\boldsymbol{q}_1^i \cdot (\boldsymbol{x}||\boldsymbol{\pi}_i), \dots, \boldsymbol{q}_\ell^i \cdot (\boldsymbol{x}||\boldsymbol{\pi}_i))$.*

In round ρ, the verifier V *outputs either* accept *or* reject.

Let $\mathcal{L} \subseteq R^m$ be a language. A ρ-round, ℓ-query public-coin fully linear interactive proof protocol (P, V) *over R is a* zero-knowledge fully linear interactive proof protocol *for $\mathcal{L}$ with soundness error ε if:*

- ***Completeness:*** *If $x \in \mathcal{L}$, then* V *always outputs* accept.
- ***Soundness:*** *If $x \notin \mathcal{L}$, then for all* P^**, the verifier* V *outputs* reject *except with probability $2^{-\varepsilon}$.*
- ***Zero Knowledge:*** *There exists a simulator* Sim *such that for all $x \in \mathcal{L}$*

$$\mathsf{Sim}(\cdot) \equiv \mathsf{view}_{[\mathsf{P}(\boldsymbol{x},\boldsymbol{w}),\mathsf{V}(\boldsymbol{x})]}(\mathsf{V})$$

where the view of V *are the challenges $C_\rho = (c_1, \dots, c_\rho)$ and the queries answers $\{(\boldsymbol{q}_1^i \cdot (\boldsymbol{x}||\boldsymbol{\pi}_i), \dots, \boldsymbol{q}_\ell^i \cdot (\boldsymbol{x}||\boldsymbol{\pi}_i))\}_{i\in[\rho]}$*

We note that when zk-FLIOP system has a single round as defined above, then it is referred to as *fully linear probabilistically checkable proofs* (FLPCP).

Our MPC protocol, which we present in Section 4, works for any finite ring, assuming the existence of a zk-FLIOP over this ring for degree-2 languages, that is, languages for which membership can be checked using a degree-2 polynomial. In the work of Boneh et al. [Bon+19], efficient proofs with *sublinear* size (in the input) for such languages were given, for the concrete cases where R is either a finite field, or the ring $\mathbb{Z}_{2^k}$. However, for more general rings, no zk-FLIOP constructions are known. We revisit this question in Section 5, where we show concrete constructions for very general families of rings.

2.5 Security Definition

In this work, we consider full security against malicious adversaries controlling a minority of the parties. We prove the security of our protocol via the standard ideal-real world definition for MPC [Can01; Gol04].

Ideal Functionalities. Finally, we assume functionalities for coin tossing and broadcast, denoted by $\mathcal{F}_{\mathsf{coin}}$ and $\mathcal{F}_{\mathsf{BC}}$ respectively, which can be instantiated efficiently. We discuss this in detail in Section A in the Supplementary Material.

3 From zk-FLIOP to DZKP: A New Transformation

In their work, Boneh et al. [Bon+19] showed that the abstract primitive of zk-FLIOP can be realized in the practical setting of *distributed zero-knowledge proofs* (DZKP). In this setting, a prover wishes to prove some statement to a set of verifiers. The prover P holds an input x which is distributed across multiple verifiers via a linear secret sharing scheme. Then, the prover can prove that $x \in \mathcal{L}$ for some language $\mathcal{L}$ relying on a zk-FLIOP proof system. Informally, the idea is to emulate the role of the prover $\mathsf{P}_{\mathsf{fliop}}$ and the verifier $\mathsf{V}_{\mathsf{fliop}}$ of the zk-FLIOP for proving membership in $\mathcal{L}$ in the following way:

1. In each round of j the zk-FLIOP, the prover runs $\mathsf{P}_{\mathsf{fliop}}$ on the current state, to obtain a proof $\boldsymbol{\pi}_j$ and shares it to the verifiers as $[\![\boldsymbol{\pi}_j]\!]$.
2. A random challenge is chosen from which the linear queries $\boldsymbol{q}_1^j, \ldots, \boldsymbol{q}_\ell^j$ are derived.
3. The verifiers locally compute the shared answers $[\![\boldsymbol{a}_{j,1}]\!], \ldots, [\![\boldsymbol{a}_{j,\ell}]\!] \leftarrow (\boldsymbol{q}_1^j \cdot ([\![x]\!] \,||\, [\![\boldsymbol{\pi}_j]\!]), \ldots, (\boldsymbol{q}_\ell^j \cdot ([\![x]\!] \,||\, [\![\boldsymbol{\pi}_j]\!]))$

In the final round, the parties reconstruct the answers and apply the predicate of $\mathsf{V}_{\mathsf{fliop}}$ over the answers, to output accept or reject. Two properties of the secret sharing scheme are required for the above protocol to be correct and sound. The linearity property ensures that the verifiers can compute the linear queries locally over their shares of the proof and input, thereby achieving correctness. The robustness property is what needed in order to achieve soundness. If robustness holds, then the prover is committed to the proof before it knows the challenge. Given that the input x is shared in the same way, it follows that the queries answers' are also robustly shared. This means that corrupted parties cannot cause the queries' answers to be opened to any other value but the correct answer (they can only cause the opening to fail; we will discuss this later). The above protocol is thus resilient against collusion between the prover and a subset of the verifiers, as long as the secret sharing is robust. Finally, privacy is also maintained since verifiers see only random shares of the proof and by the zero-knowledge property of the zk-FLIOP.

The above construction assumes that the prover knows the entire input, i.e., the shares held by the verifiers. However, in the context of MPC, this is not always the case. Assume that the parties hold a sharing $[\![x]\!]$ and some party

P_i wishes to prove that it performed some computation over its *share* of x. In this case, we wish to leverage the properties of the secret sharing as before (i.e., linearity and robustness). However, unlike the previous setting, we now need to maintain privacy also against the prover, i.e, we need to make sure that nothing is learned about the other parties' shares of x. If we use the above protocol in this setting, it does not necessarily hold. The zero-knowledge property of the zk-FLIOP guarantee privacy as long as one can only make linear queries to the proof and input. Here however the prover knows the proof and the shares of the proof held by all parties. Hence, when the verifiers publish their shares of the queries' answers, it may leak information about their shares of x.

We propose a different protocol, where the proof is shared, without the prover knowing the other parties' shares. The formal description appears in Protocol 2. The idea is to let the parties choose a random secret sharing and then "fix" the share of P_i, such that its share will hold the proof. By doing this, we obtain that the proof is shared in the same way as the input x, without the prover knowing the other parties' shares. Then, the protocol proceeds as before. As a result, the honest parties run linear queries over shares of the proof and input which are unknown to the prover. Due to lack of space, the security proof is in Appendix C.1.

Protocol 2: Π_{distZK}

The parties hold $[\![\boldsymbol{x}]\!]$ and a circuit C.
In the protocol, P_i is the prover and the other parties are the verifiers. Denote the share held by P_i by $\boldsymbol{x}_i$
Let $(\mathsf{P}_{\mathsf{fliop}}, \mathsf{V}_{\mathsf{fliop}})$ be a zk-FLIOP protocol with ρ rounds, ℓ-queries per round and message length $u_1, \ldots, u_\rho \in \mathbb{N}$ for proving that $C(\boldsymbol{x}_i) = 0$.

The protocol:

1. For each round j of the zk-FLIOP:
   (a) The parties call $\mathcal{F}_{\mathsf{rand}}$ to receive $[\![\boldsymbol{r}_j]\!]$.
   (b) If $j = 1$, party P_i lets $\boldsymbol{\pi}_j^i = \mathsf{P}_{\mathsf{fliop}}(\boldsymbol{x}_i, \perp)$. Otherwise, it sets $\boldsymbol{\pi}_j^i = \mathsf{P}_{\mathsf{fliop}}(\boldsymbol{x}_i, \boldsymbol{\pi}_{j-1}^i, \tau_{j-1}^i)$.
   (c) Let $\boldsymbol{r}_{j,i}$ be the share of $\boldsymbol{r}_j$ held by P_i. The prover P_i broadcasts $\boldsymbol{e}_i^j = \boldsymbol{\pi}_j^i - \boldsymbol{r}_{j,i}$ to the other parties [a].
   (d) The parties define the secret sharing $[\![\boldsymbol{e}_j^i|_i]\!]$ by running $\mathsf{complete}(\{v_j\}_{j\in T})$, where $T = \{i, j_1, \ldots, j_t\}$, $v_i = \boldsymbol{e}_i^j$ and $v_{j_1} = \cdots = v_{j_t} = 0$ (where $j_1, \ldots, j_t$ are fixed in advance).
   Then, the parties locally compute $[\![\boldsymbol{v}_j^i]\!] = [\![\boldsymbol{r}_j]\!] + [\![\boldsymbol{e}_j^i|_i]\!]$.
   (e) The parties call $\mathcal{F}_{\mathsf{coin}}$ to receive a random challenge τ_j^i.
   (f) The parties derive the queries $\boldsymbol{q}_{j,1}^i, \ldots, \boldsymbol{q}_{j,\ell}^i$ by running $\mathsf{V}_{\mathsf{fliop}}$ on τ_j^i. Then, the parties locally compute the linear queries:

$$\left[\!\left[a_{j,1}^i\right]\!\right], \ldots, \left[\!\left[a_{j,\ell}^i\right]\!\right] \leftarrow \boldsymbol{q}_{j,1}^i \cdot [\![\boldsymbol{x}_i]\!] \left|\right| \left[\!\left[\boldsymbol{v}_j^i\right]\!\right], \ldots, \boldsymbol{q}_{j,\ell}^i \cdot [\![\boldsymbol{x}_i]\!] \left|\right| \left[\!\left[\boldsymbol{v}_j^i\right]\!\right]$$

2. The parties run the $\mathsf{reconstruct}$ algorithm on all queries answers they received, while broadcasting their shares (via $\mathcal{F}_{\mathsf{BC}}$). Then, there are two cases:

- There exists an answer which couldn't be reconstructed, due to inconsistency. In this case, the parties output abort.
- All answers were reconstructed successfully. In this case, the parties run the predicate of $\mathsf{V}_{\mathsf{fliop}}$ on party P_i's shares of the queries' answers and outputs whatever it outputs (acceptor reject).

[a] Note that a weak broadcast suffices here, e.g., the prover sends the message to the parties and then each party can broadcast a hash of the message.

Non-interactive DZKP via the Fiat-Shamir transform. Our protocol Π_{diztZK} can be made non-interactive using the Fiat-Shamir transform [FS86]. Concretely, let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function modeled as a random oracle (where κ is a security parameter). The idea is that the random challenge in each round will be generated by applying H over the masked proofs and challenges seen so far. Hence, the prover can perform its computation locally for all rounds, and sends the masked proofs in a single message at the end. The verifiers can replay this process, computing the challenges based on the same public information, and then locally computing their shares of the queries' answers. Finally, the answers are reconstructed in a single round of interaction between the parties.

4 Fully Secure MPC via Distributed ZK Proofs

In this section, we present our fully secure MPC protocol. Our protocol follows the general player elimination framework [HMP00]. According to this approach, the circuit is divided into $O(n)$ segments. The computation of each segment has three steps: (i) *semi-honest computation:* the parties first evaluate the segment using a semi-honest protocol (which is private in the presence of malicious adversaries), (ii) *Verification*: the parties run a verification step to ensure correctness. If verification succeeds, the parties proceed to the next segment. Otherwise, the parties find a *semi-corrupt* pair which is a pair of parties with the guarantee that one of them is corrupt, and proceed to the next step. (iii) *Recovery*: the parties remove this pair from the protocol and recompute the current segment.

Our contribution is designing a new protocol for (ii), i.e., a new verification protocol. The semi-honest computation is carried-out using the DN protocol (Section 2.3). For the recovery step, one can use any known method; e.g., [Boy+20; GSZ20; Ish+16].

4.1 Building Blocks

Verifiable secret sharing. Let $\mathsf{vss.share}(\boldsymbol{x}, i)$ be a protocol to deal a vector of secrets $\boldsymbol{x}$ by a dealer P_i. At the end of the protocol, either the parties hold a consistent secret sharing $[\![\boldsymbol{x}]\!]$ or a semi-corrupt pair.

To realize this, the following simple method can be used. First, the dealer P_i shares $\boldsymbol{x}$ to the parties and a random r. Then, the parties call $\mathcal{F}_{\mathsf{coin}}$ to receive $\boldsymbol{\gamma}$ and locally compute $[\![\boldsymbol{x} \cdot \boldsymbol{\gamma} + r]\!]$. Finally, the parties open the result by broadcasting their shares. In case of inconsistency, party P_i broadcasts an index k of some party who has published an incorrect share. In this case, the parties output (i, k).

We use the notation $\mathsf{vss.share}(\boldsymbol{x}, \{v_j\}_{j \in T}, i)$, where the secret sharing is carried-out while fixing the share of each party P_j where $j \in T$ to be v_j.

Resolve inconsistency. Let $\mathsf{ss.check}([\![x]\!])$ be a protocol that takes a sharing $[\![x]\!]$ that was found to be inconsistent and output a semi-corrupt pair. The input of the parties includes all the shares of x as was published by the parties (as the inconsistency was discovered when trying to reconstruct x). The protocol is inspired by [GSZ20]. The formal description and analysis can be found in Appendix B.

Converting between thresholds. Let $\{[\![x_i|_i]\!]\}_{i=1}^n \leftarrow \mathsf{ss.convert}(\langle x \rangle, [\![x]\!])$ be a procedure that takes a sharing of x shared with two thresholds. At the end, the procedure outputs $[\![x_i|_i]\!]$ for each $i \in [n]$, where x_i is P_i's share of $\langle x \rangle$, or a semi-corrupt pair. There are different ways to realize this, which we discuss later.

4.2 Verification with Cheating Identification

In this section, we present a verification protocol which allows the parties to detect cheating, with the property that if cheating was detected, then the parties will identify a *semi-corrupt* pair. As mentioned before, a semi-corrupt pair must contain at least one corrupted party. Our main goal is to achieve this while using the "single-prover" approach, where each party proves it acted honestly and sent the correct messages during the semi-honest evaluation of the circuit. To this end, observe first that none of the messages in the DN protocol are private. Indeed, the only reason for communicating via P_1 is to reduce bandwidth. Thus, the first step of the parties in our protocol is to agree on a "compressed" transcript of the semi-honest computation. Hence, each party publishes a random linear combination of the messages it sent and the messages it received during the computation of the circuit. If there is a contradiction between any two parties, i.e., P_1 and some P_i, then (P_1, P_i) is defined as a semi-corrupt pair. Once there is an agreement regarding the transcript, the parties verify correctness. In the semi-honest protocol, there are two types of messages sent: from parties to P_1 and from P_1 to the other parties. Our first observation is that there is no need to verify the correctness of P_1's message explicitly. If all messages sent to P_1 were correct, then it remains only to ensure that P_1 added its own correct share to its message. Specifically, given the second round message $x \cdot y - r$ and all the first round messages from the other parties to P_1, it is possible to compute the additive share that P_1 added in order to obtain $x \cdot y - r$. It suffices to verify the correctness of this additive share, which can be viewed as P_1's implicit message, together with the other first round messages.

We thus turn our attention to a method for verifying the correctness of the first round messages. Let msg^i be the compressed message of P_i. Let $x_{k,i}$ and $y_{k,i}$ be the shares of P_i on the input wire of the kth gate, and let $r_{k,i}$ be the share of the mask used by P_i. Given a circuit of m gates, party P_i aims to prove that

$$\mathsf{msg}^i - \sum_{k=1}^{m} \gamma_k \cdot (x_{k,i} \cdot y_{k,i} - r_{k,i}) = 0 \tag{1}$$

where γ_k is a random public coefficient.

To leverage the zk-FLIOP machinery, Eq. (1) must be a degree-2 computation (i.e., has a multiplicative depth of exactly one) over robustly shared inputs. Set $x'_{k,i} = \gamma_k \cdot \boldsymbol{x}_{k,i}$ and $r_i = \sum_{k=1}^{m} \gamma_k \cdot r_{i,k}$. We can write Eq. (1) as

$$(\mathsf{msg}^i - r_i) - \sum_{k=1}^{m} x'_{k,i} \cdot y_{k,i} = 0 \tag{2}$$

While the equation has the right degree, r_i is not robustly shared. We thus reduce its degree to t. For this task, we use the $\mathsf{ss.convert}$ procedure. Since this is carried-out once for the entire protocol, any implementation of this procedure will suffice.

We can now call Π_{distZK} (Protocol 2) from Section 3 to verify the correctness of Eq. (2) for each party. If all proofs are accepted, then the parties approve the computation of this segment. If any proof is rejected, we know that the prover is corrupt. In this case, the prover is eliminated from the protocol. There is however another option: the queries' answers cannot be reconstructed due to inconsistency. In this case, we run the protocol $\mathsf{ss.check}$ from above on the inconsistent shared answer and locate a semi-corrupt pair. For this to work, we need to show that each answer can be expressed as the summation of n secret sharings, each dealt by some party. We prove this in Claim 4.2. The formal description of our protocol is shown in Protocol 3.

Protocol 3: Π_{vrfy}

Let m be the number of multiplication operations to verify. Let $\mathsf{msg}_{i,k}$ be the message sent from each P_i to P_1 and let $\mathsf{msg}_{1,k}$ be the message sent by P_1, in the computation of the k'th gate.

The protocol:
Part I: reach an agreement on a compressed transcript:

1. The parties call $\mathcal{F}_{\mathsf{coin}}$ to receive $\gamma_1, \ldots, \gamma_m$.
   Then, each party P_i ($i \neq 1$) broadcasts (via $\mathcal{F}_{\mathsf{BC}}$) $\mathsf{msg}^i = \sum_{k=1}^{n} \gamma_k \cdot \mathsf{msg}_{i,k}$ and $\mathsf{msg}^{rnd\ 2} = \sum_{k=1}^{m} \gamma_k \cdot \mathsf{msg}_{1,k}$ to the other parties.
   In parallel, P_1 broadcasts (via $\mathcal{F}_{\mathsf{BC}}$) msg^i and $\mathsf{msg}^{rnd\ 2}$ computed in the same way for each $i \neq 1$.

2. If there is any contradiction between a message published by some P_i $(i \neq 1)$ and P_1, then the parties output $(1, i)$ and halt. Otherwise, they proceed to the next step.

Part II: verify correctness of compressed messages:
Let $\mathsf{msg}^1 = \mathsf{msg}^{rnd\ 2} - \sum_{i=2}^{n} \mathsf{msg}^i$.
For each gate $k \in [m]$, the parties hold $[\![x_k]\!]$, $[\![y_k]\!]$ and $(\langle r_k \rangle, [\![r_k]\!])$.
Let $x_{k,i}, y_{k,i}$ and $r_{k,i}$ be the shares of held by P_i in $[\![x_k]\!], [\![y_k]\!]$ and $\langle r_k \rangle$.
The parties work as follows:

1. The parties locally compute $\langle r \rangle = \sum_{k=1}^{m} \gamma_k \cdot \langle r_k \rangle$ and $[\![r]\!] = \sum_{k=1}^{m} \gamma_k \cdot [\![r_k]\!]$.
2. For each $i \in [n]$, the parties run $[\![r_i|_i]\!] = \mathsf{ss.convert}(\langle r \rangle, [\![r]\!], i)$.
3. Set $[\![\boldsymbol{x}']\!] = (\gamma_1 \cdot [\![x_k]\!], \ldots, \gamma_m \cdot [\![x_m]\!])$, $[\![\boldsymbol{y}]\!] = ([\![y_k]\!], \ldots, [\![y_m]\!])$ and $[\![z^i]\!] = [\![\mathsf{msg}_i|_i]\!] - [\![r_i|_i]\!]$, where $[\![\mathsf{msg}_i|_i]\!]$ is defined by running $\mathsf{complete}(\{v_j\}_{j \in T})$ where $T = \{j_1, \ldots, j_t, i\}$, for some fixed set $j_1, \ldots, j_t$ and $v_i = \mathsf{msg}_i$ and $v_{j_1} = \cdots = v_{j_t} = 0$.
4. For each $i \in [n]$, the parties run Π_{distZK} (Protocol 2) on $([\![z^i]\!] \,||\, [\![\boldsymbol{x}']\!] \,||\, [\![\boldsymbol{y}]\!])$ and the circuit $C = z_i^i - \boldsymbol{x}'_i \cdot \boldsymbol{y}_i$.
5. If there exists an answer which couldn't be reconstructed, due to inconsistency, then:
   Let $[\![a_{j,l}^i]\!]$ be the first answer that couldn't be reconstructed. The parties run $\mathsf{ss.check}([\![a_{j,l}^i]\!])$ to receive a pair (w, v). The parties output (w, v) and halt.
6. Otherwise, all answers were reconstructed successfully. If the output of all invocations of Π_{distzk} was accept, the parties output accept. Otherwise, let i be the invocation with the smallest index, where the proof was rejected. In this case, the prover P_i broadcasts an index k, and the parties output (i, k).

Output: The parties output accept or a semi-corrupt pair as described above.

The correctness of the protocol is straightforward, given the completeness property of the zk-FLIOP.

Claim. For each query answer $[\![a]\!]$ in Protocol 3 there exist $[\![a^1]\!], \ldots, [\![a^n]\!]$ such that $[\![a]\!] = \sum_{i=1}^{n} [\![a^i]\!]$ and a^i was dealt by party P_i.

Proof. By definition, each query answer $[\![a]\!]$ is a result of computing the inner product between a public vector of the queries and a vector of the input concatenated with the proof. Observe that the shared proof is computed by generating a random $[\![r]\!]$ and adding a public sharing. The input itself consists of $[\![z^i]\!]$, $[\![\boldsymbol{x}']\!]$ and $[\![\boldsymbol{y}]\!]$. Note that $[\![z^i]\!] = [\![\mathsf{msg}_i|_i]\!] - [\![r_i|_i]\!]$, where $[\![\mathsf{msg}_i|_i]\!]$ is public. Similarly, note that $\boldsymbol{x}$ and $\boldsymbol{y}$ are values on input wires to multiplication gates, which are the result of a linear computation over outputs from the previous multiplication layer (or input wires). In our semi-honest protocol, each such value is computed by taking some $[\![r]\!]$ and add to it a public value (e.g., $x \cdot y - r$). It follows that if each random sharing that is generated throughout the protocol satisfies the claim, then so does the answers to the queries. This indeed holds for any random sharing during the protocol (due to the way it is generated) and thus the claim follows. □

We prove in Section C in the Supplementary Material that the cheating probability is bounded:

Lemma 2 (Cheating probability). *Let R be a finite ring, where ω_R is the size of the largest exceptional subsets of R (as defined in Section 2.1). If a corrupted P_i sent an incorrect message in the computation of some multiplication gate, then the honest parties will output* accept *at the end of Π_{vrfy} with probability of at most $\frac{1}{\omega_R} + \epsilon$, where ϵ is the soundness error of the zk-FLIOP.*

Next, we prove in Section C in the Supplementary Material that the protocol is secure. This is proven by showing that there exists a simulator, that does not know the honest parties' inputs, and yet can produce a transcript that is distributed identically to a real execution.

Proposition 2 (Security). *Let ε be the soundness error of Π_{vrfy}. Then, for every malicious adversary $\mathcal{A}$ controlling up to t parties, there exists a simulator $\mathcal{S}$ who receives: (i) all the messages $\mathsf{msg}_{i,k}$ and $\mathsf{msg}_{1,k}$ for which either P_i or P_1 is corrupted; and (ii) all shares of $[\![x_k]\!], [\![y_k]\!], (\langle r_k\rangle, [\![r_k]\!])$ held by corrupted parties, and outputs a transcript $\mathsf{view}_{\mathcal{S}}$, such that $SD(\mathsf{view}_{\mathcal{S}}, (\mathsf{view}_{\mathcal{A}}^{\Pi_{\mathsf{vrfy}}}, \mathsf{out}_{\mathsf{Honest}}^{\Pi_{\mathsf{vrfy}}})) \leq \varepsilon$ (where $SD(X,Y)$ is the statistical distance between X and Y).*

4.3 Realizing the ss.convert() Subprotocol

Recall that the goal of the conversion protocol is to generate $[\![r_i|_i]\!]$ for each i, where r_i is P_i's additive share in $\langle r\rangle$, and $\langle r\rangle = \sum_{k=1}^{m} \gamma_k \cdot \langle r_k\rangle$. Note that the shares in $[\![r]\!]$ are independent and different from the shares in $\langle r\rangle$ (i.e., P_i's share in $[\![r]\!]$ in *not* r_i!), which is what makes the subprotocol not trivial to realize. We present two protocols, which require $O(n^2)$ communication. However, since this protocol is called exactly once in our verification protocol, its cost is amortized away.

First solution: $t < n/2$, naive preprocessing Consider the following preprocessing protocol to generate the double sharing $[\![r_k]\!], \langle r_k\rangle$ for the kth gate: each party chooses its *additive* share $r_{k,i}$ of r_k and then runs $\mathsf{vss.share}(r_{k,i})$ towards the other parties. Hence, the parties obtain $[\![r_{k,i}]\!]$ and then can locally compute $[\![r_k]\!] = \sum_{i=1}^{n} [\![r_{k,i}]\!]$.

Given that the parties see $[\![r_{k,i}]\!]$, they can store them and in Π_{vrfy} compute $[\![r_i]\!] = \sum_{k=1}^{m} \gamma_k \cdot [\![r_{k,i}]\!]$ locally. This is however not enough. In $[\![r_i]\!]$, r_i is the secret, while what we need is $[\![r_i|_i]\!]$, which is a robust sharing of some secret, where r_i is the share of P_i. To achieve this, we can let each P_i secret share $[\![r_i|_i]\!]$ and then run a simple test for checking correctness with respect to $[\![r_i]\!]$:

Protocol 4: $\mathsf{checkEquality}([\![r_i]\!], [\![r_i|_i]\!], i)$

The parties hold a pair of sharings $[\![r_i]\!], [\![r_i|_i]\!]$ and an index i of party P_i who dealt both sharings.

The protocol:

1. Party P_i chooses random secrets s_i and s_i'.
   Then, run: (i) $[\![s_i]\!] \leftarrow \mathsf{vss.share}(s_i, i)$ (ii) $[\![s_i|_i]\!] \leftarrow \mathsf{vss.share}(s_i', \{v_i\})$ where $v_i = s_i$.
2. Call $\mathcal{F}_{\mathsf{coin}}$ to obtain a random α_i.
3. Locally compute $[\![u_i]\!] = \alpha \cdot [\![r_i]\!] + [\![s_i]\!]$ and $[\![v_i]\!] = \alpha \cdot [\![r_i|_i]\!] + [\![s_i|_i]\!]$.
4. Run $\mathsf{reconstruct}([\![u_i]\!])$ and $\mathsf{reconstruct}([\![v_i]\!])$.
   – If reconstruction fails, party P_i broadcasts an index k of party who published an incorrect share.
     Then, output (i, k).
   – Check equality between u_i and P_i's share of v_i. If they equal, output accept. Otherwise, P_i broadcasts an index k and the parties output (i, k).

By Lemma 1, the success cheating probability of a malicious party who cheats when sharing $[\![r_i|_i]\!]$ over a ring R is $\frac{1}{\omega_R}$. The random masking ensures that nothing is leaked when reconstructing the secrets.

With this approach, we can for example take the semi-honest protocol of Escudero and Soria-Vazquez [ESV21], which extends the Shamir's scheme to any ring and lift it to full security.

Silent preprocessing via replicated secret sharing. By using a set of replicated keys, the preprocessing can become non-interactive, except for a short setup. Specifically, each party hands a key K_T to each subset of parties T of size $n - t$. From this set of replicated keys, the parties can locally generate infinite number of $[\![r_{k,i}]\!]$ by applying a pseudorandom function over the index k with their keys. This allows silent preprocessing, but the overall number of keys is $n \cdot \binom{n}{t}$, thereby growing exponentially with the number of parties. Note that as shown in [CDI05], it is possible to locally convert any replicated secret sharing to Shamir's secret sharing. Hence, the computational costs associated with the replicated secret sharing scheme can be bounded to the offline only.

Second solution: $t < \frac{n}{3}$, general preprocessing. When $t < n/3$, any multiplication of two shared values results with a degree-$2t$ sharing. Hence, for the multiplication protocol it suffices for $\langle r \rangle$ to be a degree-$2t$ sharing. Note that in this setting, degree-$2t$ sharings are robust, since there are $2t + 1$ honest shares which determine the secret and the other shares. As a consequence, it is easy to carry-out the conversion from $\langle r \rangle$ to $[\![r_i|_i]\!]$. Specifically, party P_i shares r_i as $[\![r_i|_i]\!]$ and then the parties check correctness. The check can be done in a similar way to $\mathsf{checkEquality}()$ above. The parties produce $\langle s \rangle$ for some random s, where P_i's share is s_i, and then party P_i shares $[\![s_i|_i]\!]$. The parties then reconstruct $\alpha \cdot \langle r \rangle + \langle s \rangle$ and $\alpha \cdot [\![r_i|_i]\!] + [\![s_i|_i]\!]$, to check equality between P_i's share in both reconstructions. Given that both sharings are robust, the adversary can only cause the reconstruction to fail due to inconsistency. In this case, we call ss.check on the sharing, for which the reconstruction failed. By Lemma 1, the cheating probability is $\frac{1}{\omega_R}$ for a ring R.

Application: large scale computation with Shamir's secret sharing. When the number of parties is large, Shamir's secret sharing is preferable over replicated secret sharing due to the size of the share being constant. The state-of-the-art method to compute the double-sharing relies on Hyper-invertible matrices [DN07]. Using this method, the parties share secrets with $O(n^2)$ communication, from which they locally extract $O(n)$ random secrets. Hence, the amortized communication is $O(n)$, beating the naive preprocessing from the first solution. As our conversion protocol does not depend on the way the double-sharing is generated, we can use this type of preprocessing as well. In the preprocessing itself, it is necessary to verify that each party secret shares the same secret with degree-t and degree-$2t$. This can be easily done by taking a random linear combination of many secrets (dealt by the same party) together and reconstruct the result. If the reconstruction fails, the dealer can point at a cheater and then the parties output the dealer and that party as the new semi-corrupt pair. If the test itself fails, then the dealer is identified as the cheater.

A new improved solution to the double-dipping attack. While the above solution seems straight-forward once we reduce the threshold to $t < \frac{n}{3}$, there is a subtle issue that arises in this setting and requires our attention. In [GLS19], the authors presented an attack on DN-style protocols when $2t + 1 < n$ (as opposed to honest majority with full threshold, i.e., $n = 2t + 1$). The attack breaks the privacy property of the "textbook" version of DN, which is essential for the security of frameworks like ours, where verification is deferred to the end, after the circuit was computed. The attack is carried-out over two layers of multiplication gates, using the fact that once P_1 receives $2t$ sharings, it can compute $x \cdot y - r$ and then compute the remaining $n - 2t - 1$ shares. Now, assume that P_n is honest and a malicious P_1 sends him- and only him - an incorrect message (recall that consistency of messages from P_1 is verified only at the end, in part 1 of our verification protocol!). As a result, in the next layer, party P_n will hand P_1 an incorrect share. However, due to the redundancy in the secret sharing, P_1 will be able to compute the share P_n should have sent from other $2t + 1$ correct shares it sees.Then, P_1 can compute the difference between the actual message received from P_n and the message it should have sent. From this difference, P_1 is able to extract private information. We remark that the attack works even if we decide that only $2t$ parties will communicate their shares to P_1, as some of the remaining parties may collude with P_1 and hand him their shares on the wires.

There are two solutions in the literature to this attack. Goyal et al. [GLS19] suggested to share the mask using an additive secret sharing instead of using $2t$-sharing. This solves the attack as now there is no redundancy in the secret sharing anymore. However, it increases the communication of the protocol, since now all $n - 1$ parties need to send messages to P_1 instead of just $2t$. Furukawa and Lindell [FL19] solved the attack by running a constant-cost consistency check between each two layers. This solves the attack since any inconsistency from the side of P_1 is identified before the computation of the next layer. However, this adds extra rounds of communication for each layer, which is something we crucially want to avoid. Our framework yields a new solution that allows using the DN protocol without any modification. Specifically, in our framework, we need more than $2t+1$ parties only once during the entire execution - for the conversion protocol, which uses $\langle r \rangle$ only! This means that we can use a preprocessing protocol which generates $[\![r]\!]$ shared across $2t+1$ parties with degree-t and $\langle r \rangle$ shared across n parties with degree-$2t$. In the online protocol, only $2t+1$ parties participate in the computation and the remaining parties rejoin the computation in the verification step for running the conversion protocol. Therefore, only $2t+1$ parties hold shares on the wires of the circuit. The remaining $n-2t-1$ parties do not hold any shares and thus there is no redundancy that allows computing the expected messages of any party.

Discussion. The crux of our solution is that $[\![r]\!]$ is shared only across $2t+1$ parties. If it was shared across all parties, then every party who has access to P_1's message would be able to compute their shares on every wire. In this case, a collusion between P_1 and a malicious party who is not in the $2t+1$ parties communicating in the circuit evaluation, will suffice to carry out the attack. Our framework enables the solution of sharing $[\![r]\!]$ to a subset of $2t+1$ parties only, since Π_{distZk} called in the verification, is executed entirely over degree-t sharings, and so the participation of $2t+1$ parties is sufficient to ensure robustness. We only deal with a degree-$2t$ sharing once - when converting $\langle r \rangle$ to $[\![r_i|_i]\!]$. For this task, the parties need not to know any shared wire values. Hence, there is no need for $[\![r]\!]$ to be shared across all parties.

4.4 Putting It All Together

The protocol to compute arithmetic circuits works in a natural way. For each segment of the circuit, the parties run the semi-honest protocol and then run our verification Π_{vrfy} to verify its correctness. If Π_{vrfy} outputs accept, then the parties proceed to the next segment. Otherwise, it outputs a semi-corrupt pair. This pair is eliminated and the parties recompute the last segment. The removal of the parties itself should be carried-out in a secure manner. Ishai et al. [Ish+16] proposed a generic method for recovery with cost that grows linearly with the width of the circuit. Boyle et al. [Boy+20] showed how the removal can be done essentially for free, when working with replicated secret sharing. Both the above methods follow the blueprint of resharing the secrets with a lower threshold. Goyal et al. [GSZ20] presented a different approach where the threshold remains the same, but the shares of eliminated parties are set to be 0, using a refresh protocol, carried out for each multiplication. Our protocol can be combined with any of these methods to obtain an end-to-end fully secure protocol. Security is easily proven by the privacy of the semi-honest protocol and using the security of Π_{vrfy} proven in Proposition 2.

Communication and round complexity. In the verification protocol, the first step requires $O(n)$ broadcast messages. Then, the parties call n times to the conversion protocol, which at most has roughly $O(n^2)$ communication. Then, each party proves correctness using Protocol 3. In each invocation, the prover sends the masked proof to the parties, and the parties reconstruct the answers. Denote by $|\mathsf{proof}|$ and $|\mathsf{query}|$ the size of the zk-FLIOP proof and the number of queries to reconstruct respectively. Hence, the overall communication cost is

$$|\mathsf{semi-honest}| + O(n^2) + n \cdot (n \cdot |\mathsf{proof}| + n^2 \cdot |\mathsf{query}|).$$

Naively, all messages are sent over a broadcast channel. However, it is possible to improve this significantly by noticing that the required property is that if the parties do not agree on a sent message, it suffices to find a semi-corrupt pair. Hence, we can let each party P_i send its messsages via point-to-point channels and then the parties broadcast a hash of all messages sent by P_i. If the hashes are not the same, then P_i broadcasts an index k of a party who cheated, and the pair (P_i, P_k) is the new semi-corrupt pair. Hence, the number of broadcast messages is simply $O(n^3)$. The above expression thus refers to point-to-point messages only. Note that the number of rounds in the verification protocol is the same as in Protocol 3, since all other components of the verification protocol have a constant number of rounds.

From Theorem 5.8 in [Bon+19], there are two possible realizations to the zk-FLIOP. In the first instantiation, the size of the proof and the number of queries is square-root in the input size, but the number of rounds is constant. In the second instantiation, the proof size and the number of queries are logarithmic in the input size, but so is the number of rounds. Nevertheless, it can be made constant using the Fiat-Shamir transform. We also note that in the logarithmic realization, it is possible to batch the reconstruction of the queries, such that the number of reconstructions is constant. In our protocol, the size of the input is roughly $O(m)$ (where $m = |C|$). Overall, when instantiating the zk-FLIOP with the logarithmic construction from [Bon+19], the communication complexity is therefore dominated by $O(n^3) + n^2 \cdot O(\log m))$ sent messages. The communication complexity of the entire protocol is thus $|\mathsf{semi-honest}| + n^2 \cdot O(\log |C|) + O(n^3)$ sent ring elements and $O(n^3)$ broadcast messages. The overall number of rounds, in the random oracle model, is $\mathsf{depth}(C)$.

Working over small fields and rings. When ω_R is small, we can work over an extension field or ring to obtain sufficiently small statistical error. Alternatively, one can simply repeat the verification multiple times. This increases the communication complexity by a multiplicative factor, determined by the degree of extension (or the number of repetitions). In addition, note that [Bon+19] considers only rings which are either finite fields or rings of integers modulo 2^k. When working over such rings or if the field is too small, their theorem states that there is a multiplicative factor, added to the complexity measures of the zk-FLIOP. This is caused by working over an extension field or ring, needed to obtain sufficiently small soundness error. As the verification is anyway executed over the bigger field or ring, then the zk-FLIOP is called over the same ring.

Note that the above raises two issues. First, our protocol itself can work over any arbitrary ring. A natural question to ask is whether there exist zk-FLIOP with sublinear communication, beyond fields or the ring $\mathbb{Z}_{2^k}$. A second question is whether one can avoid some of the additional cost caused by the need to embed each element in the bigger ring. We address these exact two questions in the next section.

5 ZK Fully Linear IOPs over Arbitrary Rings

Let R be an arbitrary finite ring, not necessarily commutative, and let $\mathbb{A} \subseteq R$ be a exceptional set of maximal size, with $\omega_R = |\mathbb{A}|$. Our fully secure MPC protocol from Section 4 is designed to work over this type of rings, but crucially, it requires as a building block a FLIOP for degree-2 languages, over R. We are not aware of any generic construction of such FLIOPs in the literature: the ones from [Bon+19], which serve as the starting point for the other adaptations in the literature, work over a finite field $\mathbb{F}_{p^d}$ or the ring $\mathbb{Z}_{2^k}$. In this section we present a construction for fully linear PCPs and IOPs over rings, which can be plugged into the MPC protocol we considered in Section 4. Unfortunately, we are not able to design FLIOPs for completely general rings, and instead we add the requirement that (1) $\mathbb{A} \subseteq Z(R)$, and (2) $|\mathbb{A}| \geq 2M$, where M is the amount of multiplications in the statement to prove. The first limitation is to enable the use of Proposition 1, which allows us to use for example Schwartz-Zippel lemma over R, and the second requirement is in order to be able to have enough interpolation points.[7]

Even though the family of rings we consider is quite general, it does include an important class, namely Galois rings $\mathsf{GR}(p^k, d)$ where $p^d = \omega_R$ is constant. Of course, the commutativity requirement (1) above is not a problem since Galois rings are commutative, but property (2) does not hold in this case. Particular instances of this are the relevant cases of $\mathbb{F}_2$ and $\mathbb{Z}_{2^k}$, so this issue demands a solution. Fortunately, Galois rings are very special in that they admit extensions, which, as discussed in Section 2.1, enable increasing ω_R, and in fact this is the approach taken in [Bon+19] to accommodate for the ring $\mathbb{Z}_{2^k}$. However, this comes with a price in terms of communication due to the use of this these extensions. Our second contribution in this section consists of a method to leverage RMFEs in order to obtain gains in terms of communication, when working not only over $\mathbb{Z}_{2^k}$, but over any Galois ring $\mathsf{GR}(p^k, d)$ for which p^d is small.

This section is organized as follows. First, in Section 5.1 we show how the results presented in [Bon+19] can be naturally extended from rings like finite fields or $\mathbb{Z}_{2^k}$, which is what the authors consider in that work, to our more

[7] A weaker requirement than (1) could be that every element in $\mathbb{A}$ commutes with every other element in $\mathcal{A}$. In this case, Reed-Solomon codes are not multiplicative. This setting is far more challenging and it was considered in [ESV21]. We leave it as an interesting future work to extend our techniques to that setting.

general case of any ring R. This includes the constructions of FLPCPs, and both constant and non-constant round sublinear FLIOPs. Then, in Section 5.2 we focus our attention on Galois rings $\mathsf{GR}(p^k, d)$ specifically, which include as particular cases the ones considered in [Bon+19]. We consider the most interesting and challenging case in which $p^d = O(1)$ (*e.g.* $R = \mathbb{F}_2$). In this case, our general FLPCP/IOPs from Section 5 (and the ones from [Bon+19] for that matter) add an extra κ multiplicative overhead to the proof length—which does not occur if $p^d = \Omega(2^\kappa)$. Our main result in this section is a *constant-round* FLIOP for Galois rings that achieves better scalability in terms of the input length with respect to the more naive approach followed in [Bon+19].

5.1 Extending Previous Results to Arbitrary Rings

Throughout this section we let R be an arbitrary finite ring with a maximal exceptional set $\mathbb{A} = \{\alpha_0, \ldots, \alpha_{\omega_R}\}$, and we assume that $\mathbb{A} \subseteq Z(R)$.

Fully Linear PCPs. Our first observation is that the fully-linear PCP from [Bon+19, Theorem 4.3] can be naturally extended to work not only over finite fields or $\mathbb{Z}_{2^k}$, as presented there, but actually over any finite ring R, as long as the center of the ring contains a large enough exceptional set. We note that, unlike the work of Boneh et al. [Bon+19] which considers languages of low degree d, we focus here on degree-2 languages (which is what is used in the context of MPC). Due to space constraints, we defer proofs to Section D in the Supplementary Material.

Theorem 1. *Assume that $\omega_R \geq 2M$. Let C be an arithmetic circuit over R containing M G-gates*[8] *and any number of affine gates, with $G : R^L \to R$ of degree 2. Assume that the output of the circuit is given by the last G-gate. Then there exists a fully linear PCP with strong HVZK for the language $\mathcal{L} = \{\boldsymbol{x} \in R^n \mid C(\boldsymbol{x}) = 0\}$, with the following efficiency measures: Proof size is $L + 2M + 1$ elements over R, query complexity $L + 2$ and soundness error* $\max\{2M/(\omega_R - M), 1\}$

Using Theorem 1, we can obtain the following corollary, which is analogous to [Bon+19, Corollary 4.9], and can be proven in the same way.

Corollary 1. *Let $C : R^L \to R$ be a degree-2 arithmetic circuit. Let $A : R^n \to R$ and $A_1, \ldots, A_M : R^n \to R^L$ be affine functions. Then there exists a strong HVZK fully linear PCP for the language*

$$\mathcal{L}_{C,A,A_1,\ldots,A_m} = \{\boldsymbol{x} \in R^n \mid \sum_{i=1}^{M} C(A_i(\boldsymbol{x})) = A(\boldsymbol{x})\} \tag{3}$$

that has the following efficiency metrics: Proof length $O(L\sqrt{M})$ elements of R, query complexity of $O(\sqrt{M} \cdot L)$ and soundness error of $\frac{2\sqrt{M}}{\omega_R - \sqrt{M}}$.

[8] A G-gate is a arithmetic sub-circuit of fixed degree, e.g., 2 in the case of a multiplication. See Section 4.2 in [Bon+19]

From fully linear PCPs to IOPs via recursion. The following is a natural adaptation of [Bon+19, Theorem 5.1], and the exact same proof in that setting works as it is "ring-agnostic".

Theorem 2. *Let $C : R^L \to R$ be an arithmetic circuit over R of arithmetic degree 2. Let $A : R^n \to R$ and $A_1, \ldots, A_M : R^n \to R^L$ be affine functions. Then, there exists an $O(\log M)$-round strong HVZK fully linear IOP for the language $\mathcal{L}_{C,A,A_1,\ldots,A_m}$ from Equation (9), with the following features:*

- *Proof length $L + O(\log M)$ elements over R;*
- *Query complexity $L + O(\log M)$;*
- *Soundness error $O(\log M/\omega_R)$.*

Fully Linear IOPs for SIMD circuits. Of particular interest to us are SIMD circuits, which are comprised of the same building block repeated multiple times, and their output is 0 only if all individual outputs are 0. More precisely, given an arithmetic circuit $C : R^L \to R$ and affine functions $A_1, \ldots, A_M : R^n \to R^L$, we consider the language

$$\mathcal{L}_{\mathsf{SIMD}} = \{\boldsymbol{x} \in R^n \mid \forall j \in [M] : C(A_j(\boldsymbol{x})) = 0\}. \tag{4}$$

It is shown in [Bon+19, Theorem 5.3] that an $r+1$-round FLIOP for $\mathcal{L}_{\mathsf{SIMD}}$ can be built from any r-round FLIOP for languages of the form $\mathcal{L} = \{\boldsymbol{x} \in R^{n+M} \mid \sum_{i=1}^{M} \tilde{C}(\tilde{A}_i(\boldsymbol{x})) = 0\}$. We state this theorem below, adapted to our more general ring case and also modified to use larger random challenges, which in turn reduce the resulting soundness. This is acceptable in our case since when these proofs are compiled in our MPC protocol, the parties use PRGs to sample the random challenges.

Theorem 3. *Consider an r-round fully linear strong HVZK IOP with soundness ϵ for the language $\mathcal{L} = \{\boldsymbol{x} \in R^{n+M} \mid \sum_{i=1}^{M} \tilde{C}(\tilde{A}_i(\boldsymbol{x})) = 0\}$, where $\tilde{C} : R^{L+1} \to R$ is a degree-2 arithmetic circuit and $\tilde{A}_1, \ldots, \tilde{A}_M : R^{n+M} \to R^{L+1}$ are affine functions. Then, there exists a strong HVZK fully linear IOP for the language $\mathcal{L}_{\mathsf{SIMD}}$ from Equation (4). The resulting proof has the same proof length and query complexity, its soundness is $\epsilon + 1/\omega_R$, and it involves $r+1$ rounds.*

Proof (Sketch). In the first round of interaction V samples M random challenges $r_1, \ldots, r_M \leftarrow \mathbb{A}$, and then P proves to V that $\sum_{i=1}^{M} r_i \cdot C(A_i(\boldsymbol{x})) = 0$. This new circuit is indeed supported by the assumed FLIOP in the theorem statement. If at least one of the terms $C(A_i(\boldsymbol{x}))$ is not zero, the sum above can only be zero with probability at most $1/\omega_R$, which follows directly from Proposition 1. □

The following corollaries—similar to Corollaries 5.4 and 5.5 in [Bon+19]—can be obtained from combining Corollary 1 and Theorem 2 with Theorem 3.

Corollary 2. *There exists a 1.5-round fully linear IOP with strong HVZK for the language $\mathcal{L}_{\mathsf{SIMD}}$ from Equation (4) with the following efficiency features:*

– *Proof length $O(L\sqrt{M})$ elements of R;*
– *Query complexity $O(L\sqrt{M})$;*
– *Soundness error $\frac{2\sqrt{M}}{\omega_R - \sqrt{M}} + \frac{1}{\omega_R}$.*

Corollary 3. *There exists an $O(\log M)$-round fully linear IOP with strong HVZK for the language $\mathcal{L}_{\mathsf{SIMD}}$ from Equation (4) with the following efficiency features:*

– *Proof length $L + O(\log M)$ elements of R;*
– *Query complexity $L + O(\log M)$;*
– *Soundness error $\frac{1+\log M}{\omega_R}$.*

5.2 Improved SIMD FLIOPs for Galois Rings

Basic SIMD proofs. Here, we focus our attention concretely on the ring $R = \mathsf{GR}(p^k, d)$, which contains as particular cases $\mathbb{Z}_{p^k}$ and $\mathbb{F}_{p^d}$ by setting $d = 1$ and $k = 1$ respectively. Our goal is to obtain an improved version of Corollary 2 for SIMD circuits, in the relevant case in which $\omega_R = p^d$ is constant.[9] As noted in [Bon+19, Remark 4.6], lowering the soundness of the proofs in these corollaries can be achieved by embedding R in an extension ring $S = \mathsf{GR}(p^k, d \cdot m)$ such that $\omega_S = p^{d \cdot m} \gg M$, and using this ring for the proof instead. More concretely, in Corollary 2 over the ring S, getting soundness $2^{-\kappa}$ requires $\frac{2\sqrt{M}}{\omega_S - \sqrt{M}} + \frac{1}{\omega_S} \leq 2^{-\kappa}$, for which it suffices to take roughly $\omega_S \approx 2^\kappa \cdot \sqrt{M}$, which translates into $m = \Theta(\kappa + \log(\sqrt{M}))$. We summarize this in the following result, reporting complexities over the base ring R instead of S.

Corollary 4. *There exist FLIOPs with strong HVZK for the language $\mathcal{L}_{\mathsf{SIMD}}$ from Equation (4) with soundness $2^{-\kappa}$, and with the following efficiency features: 1.5 rounds, proof length $O(m \cdot L\sqrt{M})$ elements over R, and query complexity $O(L\sqrt{M})$ linear combinations over $S = \mathsf{GR}(p^k, d \cdot m)$ with $m = \Theta(\kappa + \log(\sqrt{M}))$.*

We notice that Corollary 4 already can be used as the missing building block in Section 4, for the case in which the ring R is a Galois ring. In what follows we will present a more efficient alternative.

Optimized SIMD proofs for Galois rings. Now we present our optimized proofs for SIMD circuits over Galois rings, that make use of reverse multiplication-friendly embeddings (RMFEs) in order to improve the parameters from Corollary 4, specifically for the case of constant-round proofs. Our goal is to prove the statement $\mathcal{L}_{\mathsf{SIMD}} = \{\boldsymbol{x} \in R^n \mid \forall j \in [M] : C(A_j(\boldsymbol{x})) = 0\}$, where $C : R^L \to R$ is a degree-2 circuit and $A_1, \ldots, A_M : R^n \to R^L$ are affine functions.

Let $S = \mathsf{GR}(p^k, m)$, and let (ϕ, ψ) be an RMFE with $\phi : R^\ell \to S$ and $\psi : S \to R^\ell$. Recall that this means that $\boldsymbol{x} \star \boldsymbol{y} = \psi(\phi(\boldsymbol{x}) \cdot_S \phi(\boldsymbol{y}))$ for every

[9] Unfortunately, our techniques do not lead any noticeable benefits for the non-constant round case from Corollary 3. The non-constant round achieves logarithmic proof size, but it also presents several downsides. We discuss this at the end of the section.

$\boldsymbol{x}, \boldsymbol{y} \in R^{\ell}$. Further, ϕ is injective, ψ is surjective, and we can assume that $\mathbf{1} = \psi(\phi(\mathbf{1}))$. RMFEs with constant rate, *i.e.* m/ℓ is constant, exist [CRX21]. Also, a useful fact we will make use of is that $\dim(\ker(\psi)) = m - \ell$. A proof of the following theorem can be found in Section C of the Supplementary Material.

Theorem 4. *Let $R = \mathsf{GR}(p^k, d)$ for $p^d = O(1)$, and let $S = \mathsf{GR}(p^k, d \cdot m) \cong R^m$ for some m. Consider the language $\mathcal{L}_{\mathsf{SIMD}} = \{\boldsymbol{x} \in R^n \mid \forall j \in [M] : C(A_j(\boldsymbol{x})) = 0\}$. Then there exists a 2.5-round fully linear strong HVZK IOP with soundness $2^{-\kappa}$ for the language $\mathcal{L}_{\mathsf{SIMD}}$, with the following efficiency features:*

- *Proof size of $O(\kappa \cdot L^{2/3}M^{1/3})$ elements in R;*
- *Query complexity of $O(\kappa \cdot L^{2/3}M^{1/3})$ linear combinations over $\mathsf{GR}(p^k, \kappa)$*

This should be contrasted against the proof size from Corollary 4, which has a proof size and query complexity of $O(\kappa \cdot L\sqrt{M})$: our proof constitutes an improvement factor of $L^{1/3}M^{1/6}$. For context, if $L = 1$ and $M = 2^{24} \approx 15$ million, our improvement factor is roughly $\times 16$.

On the benefits of a constant amount of rounds. Theorem 4 shows that careful use of RMFEs leads to improvements over the naive extension approach to FLIOP, specifically for the constant-round case. A natural question is whether the same ideas could lead to improvements to the non-constant round setting as well, where the communication can be even better than squared root: it can be made logarithmic.

Unfortunately, this is not the case. The fundamental reason is that, even though there is indeed an improvement stemming from the use of RMFEs, we must recall that there is also an additional cost that originates on the need to distribute shares of certain extra kernel elements. In the case of constant-round proofs, we are able to identify an appropriate extension degree for which this additional cost balances out with the improvements of the RMFEs. In contrast, in the non-constant round case, the improvement coming from the RMFEs is smaller than in the constant round setting, but the extra cost required by the kernel elements does not decrease. As a result, it is not possible to find a parameter regime for which there is an improvement.

On the other hand, even though constant-round protocols such as the ones we improve on here have generally a higher communication complexity than non-constant round protocols, it must be noted that in certain scenarios round-count may become a valuable resource, such as network settings where latency is very high with respect to bandwidth. Having a protocol that runs in a bit more than 2 rounds, such as ours, can prove to be much more beneficial than running a, say, 20-round protocol needed to perform a check on $2^{20} \approx 1M$ multiplications. In fact, using DZKP techniques to prove the correctness of a circuit with 2^{20} multiplications requires far more than 20 rounds, since 2^{20} must be multiplied by the amount of shares each party locally handles when performing secure multiplication.

Finally, we note that even though non-constant round DZKP techniques can be in principle compiled to constant round by using the Fiat-Shamir transform, this is only a heuristic approach and it is not appropriate in all cases. Furthermore, it is known that as the number of rounds in an interactive protocol increase, the tightness of the Fiat-Shamir heuristic decreases, and larger parameter sets are needed to keep the same level of security. Such degradation is already very evident even for five round protocols, as shown in [KZ20], and for protocols with more rounds such as the one appearing in our use case, there is even less understanding of the resulting security guarantees.

Acknowledgments

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2024 JP Morgan Chase & Co. All rights reserved.

References

[Ara+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. "High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority". In: *ACM CCS*. 2016.

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. "Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority". In: *CRYPTO 2012*. 2012, pages 663–680.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *ACM STOC 1988*. 1988, pages 1–10.

[Bon+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: *CRYPTO*. 2019.

[Boy+19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. "Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs". In: *ACM CCS*. 2019.

[Boy+20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. "Efficient Fully Secure Computation via Distributed Zero-Knowledge Proofs". In: *ASIACRYPT*. 2020.

[Boy+21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. "Sublinear GMW-Style Compiler for MPC with Preprocessing". In: *CRYPTO 2021*. 2021, pages 457–485.

[Boy+22] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. "Secure Multiparty Computation with Sublinear Preprocessing". In: *EUROCRYPT 2022*. 2022, pages 427–457.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS 2001*. 2001, pages 136–145.

[Cas+18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. "Amortized complexity of information-theoretically secure MPC revisited". In: *CRYPTO 2018*. 2018, pages 395–426.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty Unconditionally Secure Protocols (Extended Abstract)". In: *ACM STOC 1988*. 1988, pages 11–19.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. "Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation". In: *TCC*. 2005.

[CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. "A secret-sharing based MPC protocol for Boolean circuits with good amortized complexity". In: *TCC 2020*. 2020, pages 652–682.

[CG22] Ignacio Cascudo and Emanuele Giunta. "On interactive oracle proofs for boolean r1cs statements". In: *Financial Cryptography and Data Security, FC 2022*. 2022, pages 230–247.

[Chi+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries". In: *CRYPTO 2018*. 2018, pages 34–64.

[Cle86] Richard Cleve. "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)". In: *ACM STOC 1986*. 1986, pages 364–369.

[Cra+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *CRYPTO 2018*. 2018, pages 769–798.

[CRX21] Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. "Asymptotically-good arithmetic secret sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with strong multiplication and its applications to efficient MPC". In: *CRYPTO 2021*. 2021, pages 656–686.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *CRYPTO 2012*. 2012, pages 643–662.

[DEN22] Anders P. K. Dalskov, Daniel Escudero, and Ariel Nof. "Fast Fully Secure Multi-Party Computation over Any Ring with Two-Thirds Honest Majority". In: *ACM CCS 2022*. 2022, pages 653–666.

[DN07] Ivan Damgård and Jesper Buus Nielsen. "Scalable and Unconditionally Secure Multiparty Computation". In: *CRYPTO*. 2007.

[DS83] Danny Dolev and H. Raymond Strong. “Authenticated Algorithms for Byzantine Agreement”. In: *SIAM J. Comput.* 12.4 (1983), pages 656–666.

[ESV21] Daniel Escudero and Eduardo Soria-Vazquez. “Efficient information-theoretic multi-party computation over non-commutative rings”. In: *CRYPTO 2021*. 2021, pages 335–364.

[EXY22] Daniel Escudero, Chaoping Xing, and Chen Yuan. “More Efficient Dishonest Majority Secure Computation over Z 2 k via Galois Rings”. In: *CRYPTO 2022*. 2022, pages 383–412.

[FL19] Jun Furukawa and Yehuda Lindell. “Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest”. In: *ACM CCS 2019*. 2019, pages 1557–1571.

[FS86] Amos Fiat and Adi Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO 1986*. 1986, pages 186–194.

[Fur+17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. “High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority”. In: *EUROCRYPT 2017*. 2017, pages 225–255.

[Gen09] Craig Gentry. “A fully homomorphic encryption scheme”. PhD thesis. Stanford University, USA, 2009. URL: https://searchworks.stanford.edu/view/8493082.

[GIP15] Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. “Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits”. In: *CRYPTO 2015*. 2015, pages 721–741.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. “Communication-Efficient Unconditional MPC with Guaranteed Output Delivery”. In: *CRYPTO 2019*. 2019, pages 85–114.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority”. In: *ACM STOC 1987*. 1987, pages 218–229.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press, 2004.

[Goy+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. “ATLAS: Efficient and Scalable MPC in the Honest Majority Setting”. In: *CRYPTO*. 2021.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. “Guaranteed Output Delivery Comes Free in Honest Majority MPC”. In: *CRYPTO 2020*. 2020, pages 618–646.

[HMP00] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. “Efficient Secure Multi-party Computation”. In: *ASIACRYPT 2000*. Edited by Tatsuaki Okamoto. 2000, pages 143–161.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. “Founding Cryptography on Oblivious Transfer - Efficiently”. In: *CRYPTO 2008*. 2008, pages 572–591.

[Ish+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. “Secure Protocol Transformations”. In: *CRYPTO 2016*. 2016, pages 430–458.

[ISN89] Mitsuru Ito, Akira Saito, and Takao Nishizeki. “Secret sharing scheme realizing general access structure”. In: *Electronics and Communications in Japan* (1989).

[KZ20] Daniel Kales and Greg Zaverucha. “An attack on some signature schemes constructed from five-pass identification schemes”. In: *CANS 2020*. 2020, pages 3–22.

[KZ22] Daniel Kales and Greg Zaverucha. “Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures”. In: *Cryptology ePrint Archive* (2022).

[LN17] Yehuda Lindell and Ariel Nof. “A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority”. In: *ACM CCS 2017*. 2017, pages 259–276.

[PS21] Antigoni Polychroniadou and Yifan Song. “Constant-overhead unconditionally secure multiparty computation over binary fields”. In: *EUROCRYPT 2021*. 2021, pages 812–841.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. “Reaching Agreement in the Presence of Faults”. In: *J. ACM* 27.2 (1980), pages 228–234.

[RB89] Tal Rabin and Michael Ben-Or. “Verifiable Secret Sharing and Multiparty Protocols with Honest Majority”. In: *ACM Symposium on Theory of Computing*. 1989.

[Sha79] Adi Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (1979), pages 612–613.

[Sto+23] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. “Grotto: Screaming fast (2+ 1)-PC or Z2n via (2, 2)-DPFs”. In: *ACM CCS 2023*. 2023, pages 2143–2157.

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *FOCS 1986*. 1986, pages 162–167.

Supplementary Material

A Ideal Functionalities

We will make use of two ideal functionalities: One for generating random values $\mathcal{F}_{\mathsf{rand}}$, and one for broadcasting messages $\mathcal{F}_{\mathsf{BC}}$. Both are necessary to achieve full security and are called only small number of times (hence, any implementation will suffice).

Generating random values. We define an ideal functionality that hands the parties fresh random coins. This functionality is typically realized by generating a random secret sharing and then open it. We can reduce the number of calls to this functionality to the size of the security parameter (as it is possible to call it only to generate a seed r from which all the required randomness is derived, even in an information-theoretic way, by taking $r, r^2, \ldots$). Hence, any way to implement it will suffice.

Functionality 1: $\mathcal{F}_{\mathsf{coin}}$

On (Coin, R) from all parties, select $x \in R$ uniformly at random and send x all parties.

Broadcast. We define a secure broadcast functionality which allows the parties to broadcast a message to all the other parties. A broadcast channel is necessary to achieve full security in the weak honest majority setting, where broadcast is not possible without setup [PSL80]. Full security of $\mathcal{F}_{\mathsf{BC}}$ is achievable given PKI setup [RB89]. Fortunately, the number of times this functionality is called will be sublinear in the size of the circuit and so any reasonable implementation will suffice.

Functionality 2: $\mathcal{F}_{\mathsf{BC}}$

On (BC, x) from party P_i send x to all parties.

B Resolve Inconsistency of a Secret Sharing

Protocol 5: $\mathsf{ss.check}(\llbracket x \rrbracket)$

The parties hold a sharing $\llbracket x \rrbracket$ which was found to be inconsistent. Let x_i be the share of x that was published by P_i.
Let $\llbracket x^i \rrbracket$ be a consistent sharing that was dealt by P_i such that $\llbracket x \rrbracket = \sum_{j=1}^{n} \llbracket x^i \rrbracket$.

The protocol:

1. Each party P_i chooses a random r^i and runs $\mathsf{share}(r^i)$ to the other parties.
2. The parties locally compute $[\![r]\!] = \sum_{i=1}^{n} [\![r^i]\!]$ and run $\mathsf{reconstruct}([\![r]\!])$ by broadcasting their shares. Let r_i be the share published by P_i.
3. If r cannot be reconstructed, due to inconsistency, then the parties set $[\![z]\!] = [\![r]\!]$ and $[\![z^i]\!] = [\![r^i]\!]$ for each $i \in [n]$.
   Otherwise, they set $[\![z]\!] = [\![x]\!] + [\![r]\!]$ and $[\![z^i]\!] = [\![x^i]\!] + [\![r^i]\!]$ for each $i \in [n]$.
4. For each $i \in [n]$, the parties run $\mathsf{reconstruct}([\![z^i]\!]))$ by broadcasting their shares. Let z_j^i be the share of z^i published by P_j. Then:
   – If there exists $i \in [n]$ for which $\mathsf{reconstruct}([\![z^i]\!])) = \perp$, then P_i broadcasts (accuse, k) where k is an index of a party P_k that published an incorrect share. In this case, the parties output (i, k).
   – Otherwise, the parties proceed to the next step.
5. Let j be the minimal index for which $\sum_{i=1}^{n} z_j^i \neq z_j$. Then, the parties output (j, k) where k is the smallest index such that $k \neq j$.

Correctness. To see why the protocol is correct, consider the following cases:

– *Case 1: r could not be reconstructed.* In this case, $z = r = \sum_{i=1}^{n} r^i$ and the parties reconstruct r^i for each $i \in [n]$. If the reconstruction of any of these values fails, the party who dealt that value can identify the cheater. If the dealer itself is corrupted, then regardless of the other party, the output is semi-corrupt. If all r^is were successfully reconstructed, then it means that parties can compute each party P_j's share r_j of r by taking $\sum_{i=1}^{n} r_j^i$. However, since r could not be reconstructed, then it means that there must be some party P_j for which $r_j \neq \sum_{i=1}^{n} r_j^i$. Hence, the parties can locate a corrupted party P_j. In this case, any pair (j, k) is a semi-corrupt pair.
– *Case 2: r was successfully reconstructed.* In this case, $z = x + r = \sum_{i=1}^{n} x^i + r^i$. Note that since the parties have already published their shares of x and r, then all shares z_i are known. Then, the parties also reconstruct z^i for each $i \in [n]$. If for some i, the reconstruction fails, then as in the previous case, the dealer of that value can identify the cheater. Otherwise, all shares are consistent, which means that all shares $\sum_{i=1}^{n} z_j^i$ form a consistent sharing of z. However, we know that the shares of z are inconsistent. Hence, there must be some j for which $z_j \neq \sum_{i=1}^{n} z_j^i$, implying that P_j is corrupted. Then, we can simply add any other party to obtain a semi-corrupt pair as required.

Privacy: simulating the protocol. We describe a simulator $\mathcal{S}$ that receives as an input all shares of x, and all shares of x_i which was dealt by a corrupted P_i and all shares held by corrupted parties of x_j that was dealt by an honest P_j. In the simulation, $\mathcal{S}$ sends the adversary $\mathcal{A}$, controlling the corrupted parties, random shares as their shares of r^j for each honest party P_j. In addition, it receives the honest parties shares of r^i for each corrupted party P_i. This suffices for $\mathcal{S}$ to determine whether r is consistent or not. Then:

– *Case 1: r is not consistent.* In this case, $\mathcal{S}$ chooses a random r^j for each honest P_j and runs $\mathsf{share}(r^j, [\![r_j]\!]_T)$ where T is the set of corrupted parties. Then, it simulates the protocol by playing the role of the honest parties. Note that since in this case, the protocol is executed over random independent shares, the simulation is perfect.

– *Case 2: r is consistent.* In this case, $\mathcal{S}$ can compute the corrupted parties' shares of r^i for each corrupted P_i. This means that it knows the corrupted parties' shares of all r^i and can compute their shares of r. Then, $\mathcal{S}$ chooses a random r, and runs $\mathsf{share}(r, [\![r]\!]_T)$, where T is the set of corrupted parties.
  At this point, $\mathcal{S}$ knows $z = x + r$, all shares of z and the shares held by corrupted parties of each $z^i = x^i + r^i$. Then, $\mathcal{S}$ chooses random z^i under the constraint that $z = \sum_{i=1}^{n} z^i$ and runs $\mathsf{share}(z^i, [\![z^i]\!]_T)$ where T is the set of corrupted parties. From this point on, $\mathcal{S}$ can simulate the protocol by playing the role of the honest parties running the protocol's instructions.
  It is easy to see that $\mathcal{A}$'s view in the simulation is distributed the same as in the real execution, due to the masking with a random uniformly chosen r^i.

C Missing Proofs

C.1 Security Proof for Π_{distZK}

To prove the that protocol Π_{distZK} is secure, we define the ideal functionality $\mathcal{F}_{\mathsf{distzk}}$ in Functionality 3. The ideal functionality receives inputs only from the honest parties, as this suffices to compute the corrupted parties' inputs, due to the robustness of the secret sharing. Note that it allows the adversary to cause the parties to abort. This corresponds to the event where the answers cannot be reconstructed due to corrupted parties publishing inconsistent shares. We note that in our MPC protocol, we will not use this ideal functionality, since whenever an abort event happens, we need to locate a cheater. Nevertheless, our DZK protocol may have other uses, and thus we prove its security via a standard ideal functionality.

Functionality 3: $\mathcal{F}_{\mathsf{distZK}}$

The functionality $\mathcal{F}_{\mathsf{diskzk}}$ works with n parties $P_1, \dots, P_n$. Let $\mathcal{S}$ be the ideal world adversary controlling a subset T of the parties, where $|T| < n/2$.
Upon receiving from the honest parties their shares of x, an index i and a circuit C, the functionality $\mathcal{F}_{\mathsf{diskzk}}$ works as follows:

1. Compute the corrupted parties shares of x and send them to $\mathcal{S}$.
2. If $P_i \notin T$: upon receiving a command $\mathsf{out} \in \{\mathsf{continue}, \mathsf{abort}\}$ from $\mathcal{S}$, if $\mathsf{out} = \mathsf{abort}$ send abort to the parties. Otherwise, send accept.
3. If $P_i \in T$:
   – If $C(x_i) = 0$, then upon receiving $\mathsf{out} \in \{\mathsf{continue}, \mathsf{abort}\}$ from $\mathcal{S}$, if $\mathsf{out} = \mathsf{abort}$ send abort to the parties. Otherwise, send accept.
   – If $C(x_i) \neq 0$, then upon receiving $\mathsf{out} \in \{\mathsf{continue}, \mathsf{abort}\}$ from $\mathcal{S}$, if $\mathsf{out} = \mathsf{abort}$ send abort to the parties. Otherwise, send reject.

We prove the following:.

Proposition 3. *Protocol 2 (Π_{diztzk}) securely computes $\mathcal{F}_{\mathsf{diszk}}$ in the $(\mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{coin}})$-hybrid model, in the presence of malicious adversaries controlling $t < n/2$ parties.*

Proof. Let $\mathcal{S}$ be the ideal world simulator and let $\mathcal{A}$ be the real world adversary. The simulation begins with $\mathcal{S}$ receiving the corrupted parties' shares of $\boldsymbol{x}$. $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{coin}}$ handing $\mathcal{A}$ the random challenge and $\mathcal{F}_{\mathsf{rand}}$, where it receives from $\mathcal{A}$ the corrupted parties' shares of $\boldsymbol{r}_j$. There are two cases:

– *Case 1: P_i is honest.* In this case, $\mathcal{S}$ first chooses a random $\boldsymbol{e}_i^j$ for each round j and hands it to $\mathcal{A}$. Then, given the shares of $\boldsymbol{r}_j$ held by corrupted parties known to $\mathcal{S}$, it can compute their shares of $\boldsymbol{v}_j^i$ and run the queries over their shares of $\boldsymbol{x}$ and $\boldsymbol{v}_j^i$, to obtain the corrupted parties' shares of the queries' answers. Then, $\mathcal{S}$ invokes the simulator of the zk-FLIOP to receive the queries answers for each round j. These answers are set to be P_i's shares of $a_{j,1}, \ldots, a_{j,\ell}$. Finally, $\mathcal{S}$ chooses shares to the other honest parties, given the corrupted parties' shares and P_i's share, such that the secret sharing will be consistent. The simulator $\mathcal{S}$ then plays the role of the honest parties in the reconstruction procedure. If $\mathcal{A}$ sends incorrect shares, that result with inconsistency, $\mathcal{S}$ sends abort to the trusted party computing $\mathcal{F}_{\mathsf{distZK}}$. Otherwise, it sends $\mathsf{continue}$. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.
  Observe that the $\mathcal{A}$'s view consists of random shares, the masked proof, and the queries' answers. By the privacy of the secret sharing scheme and zero-knowledge property of the zk-FLIOP, it follows immediately that the simulation and the real-world execution are distributed the same.
– *Case 2: P_i is malicious.* In this case, $\mathcal{S}$ receives from $\mathcal{A}$ the masked proof $\boldsymbol{e}_i^j$ in each round. Since it knows P_i's share of $\boldsymbol{r}_j$, it can compute the proof $\boldsymbol{\pi}_j^i$. As in the previous case, it then computes the corrupted parties' shares of the answers queries', including P_i's share, which contains the answers themselves. Then, it chooses the honest parties' shares randomly under the constraint that the secret sharing will be consistent. The simulation of the reconstruction procedure is the same as in the previous case.
  The view of $\mathcal{A}$ consists of its own shares of the proof and random shares of the answers. Although we cannot rely directly on the zero-knowledge property as in the first case, since the prover now know the queries' answers, it can be seen that $\mathcal{A}$'s view is the same as its view when the prover is honest. Therefore, if its view in both executions is not distributed the same in this case, they are not distributed the same in that case as well.

□

C.2 Proof of Lemma 2

Proof. We consider two cases:

– *Case 1: P_1 is honest.* Thus, assume that some P_i sent an incorrect message to P_1 in the computation of the kth gate. Then, there are two events which can cause the honest parties to output accept: (i) the compressed message

msg^i is correct due to the random linear combination; (ii) $\mathsf{V}_{\mathsf{fliop}}$ outputs accept, although the statement to be proven is incorrect. By Lemma 1, the first event happens with probability $\frac{1}{\omega_R}$, and so the overall success cheating probability is $\frac{1}{\omega_R} + \epsilon$ as required.

– *Case 2: P_1 is malicious.* Note that in this case, the only message honest parties receive from a corrupted party is the second round message of P_1. Assume that P_1 sent an incorrect message to the other parties in the computation of the kth gate. The two events that can cause the honest parties to output accept are the same as in the first case. However, we need to show that cheating in its message is equivalent to adding an incorrect share to the messages received from other parties (since this is being verified for correctness in the protocol). This follows since $\mathsf{msg}_{1,k} = x_k \cdot y_k - r_k$, which is computed by adding the shares of $x_k \cdot y_k - r_k$ held by all parties. Thus, if $\mathsf{msg}_{1,k}$ is incorrect, it implies that P_1 has added an incorrect share and this is verified by the protocol. Thus, the success cheating probability is the same as in the previous case.

□

C.3 Proof of Proposition 2

Proof. We describe a simulator for the protocol. Upon receiving its input, the simulator $\mathcal{S}$ interacts with $\mathcal{A}$ while playing the role of the honest parties and the functionalities $\mathcal{F}_{\mathsf{coin}}$ and $\mathcal{F}_{\mathsf{BC}}$, as follows:

– *Simulating Part I*: The simulator chooses random $\gamma_1, \ldots, \gamma_m$ and hands them to $\mathcal{A}$. Then, it simulates the honest parties by following the protocol's instructions. Note that if P_1 is corrupted, then $\mathcal{S}$ knows all the messages sent in the protocol, and so it can compute the compressed messages and hand them to $\mathcal{A}$. If P_1 is honest, then $\mathcal{S}$ does not know the messages sent from honest parties to him. To simulate these, $\mathcal{S}$ can simply publish random messages. As each message in the semi-honest protocol is computed by masking with a random secret value, the distribution in the simulated and real execution are the same.
– The simulator $\mathcal{S}$ locally computes the corrupted parties' shares of r.
– *Simulating the execution of* $\mathsf{ss.convert}$: When P_i is honest, $\mathcal{S}$ simply hands a random share to $\mathcal{A}$ for each corrupted party. When P_i is corrupted, $\mathcal{S}$ receives the honest parties' shares from the procedure and then computes the corrupted parties shares in each $[\![r_i|_i]\!]$.
– The simulator $\mathcal{S}$ locally computes the corrupted parties' shares of each z^i.
– *Simulating the emulation of Π_{distZK} for each $i \in [n]$*: $\mathcal{S}$ runs the simulator from Proposition 3. Note that the simulator receives all the corrupted parties' shares of the input, which $\mathcal{S}$ knows, and thus the simulation can be executed. There are several cases:
  - *All messages in the semi-honest protocol are correct.* In this case, the simulation can end with the verifiers accepting all proofs or with a query answer $\left[\!\left[a^i_{j,l}\right]\!\right]$ that cannot be reconstructed. In the former case, the output of the simulation is accept. In the second case, the procedure $\mathsf{ss.check}$ is executed. In this case, $\mathcal{S}$ runs the simulator we describe in Section B. Note that for

this to work, $\mathcal{S}$ must hold all shares of the decomposition of $\left[\!\left[a^i_{j,l}\right]\!\right]$ held by corrupted parties. This holds since for any secret dealt by honest parties, $\mathcal{S}$ sent random shares to $\mathcal{A}$ on behalf of the honest parties. For secrets dealt by corrupted parties, $\mathcal{S}$ received the honest parties, from which it could compute the corrupted parties' shares. Hence, $\mathcal{S}$ can run the simulation of ss.check according to the description of that simulator. At the end, § outputs a semi-corrupt pair.

- *Cheating took place in the semi-honest execution.* The simulation here follows the guidelines of the previous case, with one exception: if the simulation ends with the honest parties outputting accept, then $\mathcal{S}$ outputs fail.

Given that the simulation of Π_{diszk} and ss.check are distributed identically to the real execution, the only difference between the simulation and the real execution is the event where $\mathcal{S}$ outputs fail. Note that this happens when cheating took place, but the honest parties in the execution still output accept. By Lemma 2, the probability that this event occurs equals to the soundness error, which is allowed by the Lemma. This concludes the proof. □

C.4 Proof of Theorem 4

Proof (of Theorem 4). Let us write $C(\boldsymbol{u}) = \boldsymbol{u}^\intercal \boldsymbol{B}\boldsymbol{u} + \boldsymbol{b}\cdot\boldsymbol{u} + b$, where $\boldsymbol{B} \in R^{L\times L}$, $\boldsymbol{b} \in R^L$ and $b \in R$. P's goal is then to prove that $\boldsymbol{u}_i^\intercal \boldsymbol{B}\boldsymbol{u}_i + \boldsymbol{b}\cdot\boldsymbol{u}_i + b = 0$ for $i \in [M]$, where $\boldsymbol{u}_i = A_i(\boldsymbol{x})$. By setting $\boldsymbol{U} \in R^{L\times M}$ be the matrix whose columns are $\boldsymbol{u}_1, \dots, \boldsymbol{u}_M$, we can write the above as $\boldsymbol{U}^\intercal\boldsymbol{B}\boldsymbol{U} + \boldsymbol{U}^\intercal\boldsymbol{b} = \boldsymbol{0}$. Also, assuming without loss of generality that ℓ divides M, let us partition $\boldsymbol{U}$ into $1\times\ell$ blocks as follows:

$$\boldsymbol{U} = \begin{pmatrix} \boldsymbol{v}_{11} & \cdots & \boldsymbol{v}_{1,M/\ell} \\ \boldsymbol{v}_{21} & \cdots & \boldsymbol{v}_{2,M/\ell} \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_{L1} & \cdots & \boldsymbol{v}_{L,M/\ell} \end{pmatrix},$$

where each term $\boldsymbol{v}_{ij}$ is not thought of as a "block", but instead it is interpreted as an element of the R-algebra R^ℓ, with the operation $\star$ corresponding to element-wise product. This way, we interpret $\boldsymbol{U}$ as an $L \times M/\ell$ matrix over the R-algebra R^ℓ. With this notation, the statement to prove becomes

$$\begin{pmatrix} \boldsymbol{v}_{11} & \cdots & \boldsymbol{v}_{L1} \\ \boldsymbol{v}_{12} & \cdots & \boldsymbol{v}_{L2} \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_{1,M/\ell} & \cdots & \boldsymbol{v}_{L,M/\ell} \end{pmatrix} \cdot \boldsymbol{B} \cdot \begin{pmatrix} \boldsymbol{v}_{11} & \cdots & \boldsymbol{v}_{1,M/\ell} \\ \boldsymbol{v}_{21} & \cdots & \boldsymbol{v}_{2,M/\ell} \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_{L1} & \cdots & \boldsymbol{v}_{L,M/\ell} \end{pmatrix} + \begin{pmatrix} \boldsymbol{v}_{11} & \cdots & \boldsymbol{v}_{L1} \\ \boldsymbol{v}_{12} & \cdots & \boldsymbol{v}_{L2} \\ \vdots & \ddots & \vdots \\ \boldsymbol{v}_{1,M/\ell} & \cdots & \boldsymbol{v}_{L,M/\ell} \end{pmatrix} \cdot \boldsymbol{b} + b\boldsymbol{I} = \boldsymbol{0} \in (R^\ell)^{M/\ell}. \tag{5}$$

We extend the definition of ϕ and ψ to work on arrays of elements of R^ℓ and S, respectively, by applying the mapping to each entry. This way, $\phi(\boldsymbol{U}) \in S^{L\times M/\ell}$ denotes the matrix that results by applying ϕ to every entry of $\boldsymbol{U}$, when interpreted as a $L \times M/\ell$ matrix over the R-algebra R^ℓ. Equation (5) is a quadratic expression over the R-algebra R^ℓ. Due to the properties of the RMFEs, we have that the equation above can be rewritten as

$$\psi(\underbrace{\phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{B} \cdot \phi(\boldsymbol{U}) + \phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{b} + b\boldsymbol{I}}_{\in S^{M/\ell}}) = \boldsymbol{0} \in (R^\ell)^{M/\ell}, \tag{6}$$

where ψ on the left-hand side is acting on each of the M/ℓ entries.

Consider a vector $\boldsymbol{h} \in R^{M/\ell}$. We can take dot product at both sides of Eq. (6) with $\boldsymbol{h}$, and due to the R-linearity of ψ, we can bring $\boldsymbol{h}$ inside of ψ. Hence, we have that if Eq. (6) above holds, then

$$\psi(\underbrace{\boldsymbol{h} \cdot (\phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{B} \cdot \phi(\boldsymbol{U}) + \phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{b} + b\boldsymbol{I})}_{\in S}) = \boldsymbol{0} \in R^\ell. \tag{7}$$

However, even more interestingly, we can prove that Equation (7) implies Equation (6) with certain bounded probability. More precisely:

Claim. Assume that Equation (6) does not hold. Then, for a uniformly random $\boldsymbol{h} \in R^{M/\ell}$, Equation (7) does not hold either, except with probability p^{-d}.

Proof (of claim). If Equation (6) does not hold, then there is at least one entry on the left-hand side that is non-zero. By Lemma 1, a linear combination of the left-hand side cannot be zero, except with probability $\omega_R^{-1} = p^{-d}$. Now, we simply have to notice that multiplying by $\boldsymbol{h} \in R^{M/\ell}$ is precisely taking a random linear combination. This completes the proof of the claim.

With the claim above at hand, we see that it suffices for P to prove that $h = \log_{p^d}(2^\kappa) = \Theta(\kappa)$ instances of Equation (7) hold, for multiple $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)} \in R^{M/\ell}$.

Now we turn our attention to designing a proof for Equation (7). Our first step is to turn this into a statement over S, instead of R. For this, simply notice that Equation (7) can be equivalently written as $z \in \ker\psi$, where $z = \boldsymbol{h} \cdot (\phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{B} \cdot \phi(\boldsymbol{U}) + \phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{b} + b\boldsymbol{I}) \in S$. Let $\{\beta_1, \ldots, \beta_{m-\ell}\} \subseteq S$ be a basis of $\mathsf{ker}(\psi)$. This is equivalent to there existing $t_1, \ldots, t_{m-\ell} \in R^h$ such that $z = \sum_{i=1}^{m-\ell} t_i\beta_i$ (recall that $\dim(\ker\psi) = m - \ell$).

At this point we are finally ready to explicitly describe the language over S that the parties will use. At a high level, P receives $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)} \in R^{M/\ell}$ from V and proves that, for each $j \in [h]$, $(\phi(\boldsymbol{U}), (t_1^{(j)}, \ldots, t_{m-\ell}^{(j)}), \boldsymbol{h}^{(j)})$ is in the language of tuples satisfying $z^{(j)} = \sum_{i=1}^{m-\ell} t_i^{(j)}\beta_i$, where $z^{(j)}$ is derived from $\phi(\boldsymbol{U})$ and $\boldsymbol{h}^{(j)}$ as described above. For proving this, V sends yet another set of challenges $\gamma_1, \ldots, \gamma_{M/\ell} \in S$, and P proves instead that $\sum_{j=1}^h \gamma_j \cdot z^{(j)} = \sum_{j=1}^h \gamma_j \cdot \left(\sum_{i=1}^{m-\ell} t_i^{(j)}\beta_i\right)$.

In a bit more detail, note we can write $z^{(j)} = \boldsymbol{h}^{(j)} \cdot (\phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{B} \cdot \phi(\boldsymbol{U}) + \phi(\boldsymbol{U})^\intercal \cdot \boldsymbol{b} + b\boldsymbol{I})$. Furthermore, we can write the right argument to the inner product above as

$$(\quad C(\ \phi(\boldsymbol{U})_1\ ), \ldots, C(\ \phi(\boldsymbol{U})_{M/\ell}\ ) \quad) \in S^{M/\ell},$$

where $\phi(\boldsymbol{U})_i$ is the i'th column of $\phi(\boldsymbol{U})$, which is equal to $(\phi(\boldsymbol{v}_{1j}), \ldots, \phi(\boldsymbol{v}_{Lj})) \in S^L$. Hence:

$$\sum_{j=1}^{h} \gamma_j z^{(j)} = \sum_{j=1}^{h} \left(\sum_{i=1}^{M/\ell} h_i^{(j)} \cdot C(\phi(\boldsymbol{U})_i) \right) = \sum_{i=1}^{M/\ell} C(\phi(\boldsymbol{U})_i) \left(\sum_{j=1}^{h} \gamma_j h_i^{(j)} \right).$$

Consider the following functions

– For $i \in [M/\ell]$, $\tilde{A}_i : R^n \times R^{h \cdot M/\ell} \times S^h \times R^{h\cdot(m-\ell)} \to S \times S^L$ maps $(\boldsymbol{x}, (\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)}), (\gamma_1, \ldots, \gamma_h), (\boldsymbol{t}^{(1)}, \ldots, \boldsymbol{t}^{(h)}))$ to $(\sum_{j=1}^{h} \gamma_j h_i^{(j)}, \phi(\boldsymbol{U})_i)$, which is an R-affine map (but clearly not S-affine).
– $\tilde{A} : R^n \times R^{h\cdot M/\ell} \times S^h \times R^{h\cdot(m-\ell)} \to S$ maps $(\boldsymbol{x}, (\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)}), (\gamma_1, \ldots, \gamma_h), (\boldsymbol{t}^{(1)}, \ldots, \boldsymbol{t}^{(h)}))$ to $\sum_{j=1}^{h} \gamma_j \sum_{i=1}^{m-\ell} t_i^{(j)} \beta_i$.
– $\tilde{C} : S \times S^L \to S$ takes $(q, \boldsymbol{w})$ and outputs $q \cdot C(\boldsymbol{w})$;

We will let $\boldsymbol{y} = (\boldsymbol{x}, (\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)}), (\gamma_1, \ldots, \gamma_h), (\boldsymbol{t}^{(1)}, \ldots, \boldsymbol{t}^{(h)})) \in R^n \times R^{h\cdot M/\ell} \times S^h \times R^{h\cdot(m-\ell)}$ be the input for the FLIOP we will use. Putting together what we have seen above, the statement $z^{(j)} = \sum_{i=1}^{m-\ell} t_i^{(j)} \beta_i$ for $j \in [h]$ can be written as

$$\sum_{i=1}^{M/\ell} \tilde{C}(\tilde{A}_i(\boldsymbol{y})) = \tilde{A}(\boldsymbol{y}). \tag{8}$$

Round 1. P receives the challenges $\boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(h)} \in S^{M/\ell}$.
Round 2. P computes $\boldsymbol{t}^{(1)}, \ldots, \boldsymbol{t}^{(h)} \in R^{m-\ell}$ as above, sends the proof $\pi = (\boldsymbol{t}^{(1)}, \ldots, \boldsymbol{t}^{(h)}) \in R^{h\cdot(m-\ell)}$.
Round 3. V sends challenges $\gamma_1, \ldots, \gamma_{M/\ell} \in S$
Remaining rounds. P proves that Equation (8) holds using an existing FLIOP.
Notice that this is the language from Equation (9), with $M' = M/\ell$ and $L' = L + 1$, and we use a ring extension $T = \mathsf{GR}(p^k, dm\eta)$ of S of degree η.

Recall that $\ell = \Theta(m)$, $m - \ell = \Theta(m)$ and $h = \Theta(\kappa)$. Using Corollary 1, this last proof can be done in one round (*i.e.* a FLPCP), with proof length $O(L'\sqrt{M'})$ elements of T plus the $h \cdot (m - \ell)$ elements over R from the second round, query complexity $O(\sqrt{M'} \cdot L')$ linear combinations over T, and soundness error $\frac{2\sqrt{M'}}{\omega_S - \sqrt{M'}}$. Plugging in our parameters, we get:

– 2.5 rounds
– Proof size of $O(\eta m L\sqrt{\frac{M}{m}} + \kappa m)$ elements over R
– Query complexity of $O(\eta m L\sqrt{\frac{M}{m}})$ linear combinations over T
– Soundness error $\frac{2\sqrt{M/m}}{p^{d\cdot m\cdot\eta} - \sqrt{M/m}}$.

For soundness $2^{-\kappa}$, we take $p^{d\cdot m\cdot\eta} = \omega_T \approx 2^\kappa\sqrt{M/m}$, so $m\eta = \Theta(\kappa + \log(\sqrt{M/m}))$. Hence, ignoring the $\log(\sqrt{M/m})$ term, the proof size becomes $O(\kappa\left(L\sqrt{M/m} + m\right))$. We minimize this over m by taking $m = L^{2/3}M^{1/3}$, obtaining a proof size of $O(\kappa \cdot L^{2/3}M^{1/3})$ elements over R.
This concludes the proof of the theorem. □

D Extending Existing FL-IOPs to Arbitrary Rings

Theorem 5 (Theorem 1, restated). *Assume that $\omega_R \geq 2M$. Let C be an arithmetic circuit over R containing M G-gates*[10] *and any number of affine gates, with $G : R^L \to R$ of degree 2. Assume that the output of the circuit is given by the last G-gate. Then there exists a fully linear PCP with strong HVZK for the language $\mathcal{L} = \{\boldsymbol{x} \in R^n \mid C(\boldsymbol{x}) = 0\}$, with the following efficiency measures:*

- *Proof size is $L + 2M + 1$ elements over R*
- *Query complexity $L + 2$*
- *Soundness error $\max\{2M/(\omega_R - M), 1\}$*

Proof. The proof turns out to be very similar to the one of Theorem 4.3 in [Bon+19], and our contribution is not precisely the proof itself, but rather the crucial observation that one can relax massively the requirements on the ring that is used for constructing the zk-FLIOP.

Prover. The prover evaluates $C(\boldsymbol{x})$, and gets access to all intermediate wires. Let $\boldsymbol{f}$ be a vector of L polynomials over R of degree $\leq M$ each, such that

- $\boldsymbol{f}(\alpha_0)$ is uniformly random in R^L
- $\boldsymbol{f}(\alpha_j) \in R^L$ is the input to the j'th G-gate, for $j \in [M]$.

Consider the polynomial $g = G(\boldsymbol{f})$, which has degree $\leq 2M$ and satisfies that $g(\alpha_j) = G(\boldsymbol{f}(\alpha_j))$ is the output of the j'th G-gate, and in particular $g(\alpha_M) = C(\boldsymbol{x})$. Let $\boldsymbol{c} \in R^{2M+1}$ be $(g(\alpha_0), \ldots, g(\alpha_{2M}))$. Such polynomial exists and is unique, thanks to Proposition 1. The prover outputs the proof $\boldsymbol{\pi} = (\boldsymbol{f}(\alpha_0), \boldsymbol{c}) \in R^{L+2M+1}$.

Verifier. The verifier samples a random challenge $\alpha_r \in \mathbb{A}$. Consider $\boldsymbol{\pi}$ as above, provided as input to V. We make the following observations. First, for each $i \in [L]$, the value of $f_i(\alpha_r)$ is a linear combination of $f_i(\alpha_0), \ldots, f_i(\alpha_M)$, since $\deg f_i \leq M$. Second, the values $f_i(\alpha_0), \ldots, f_i(\alpha_M)$ can themselves be derived as affine combinations of $(\boldsymbol{x}\|\boldsymbol{\pi})$. This is clear for $f_i(\alpha_0)$ since this is part of $\boldsymbol{\pi}$. For $f_i(\alpha_j)$ for $j \in [M]$, notice that this value is equal to the i'th input to the j'th G-gate which is given as an affine combination of elements that are either in $\boldsymbol{x}$, or outputs of a previous G-gate $g(\alpha_{j^*})$ for some $j^* < j$, which are in $\boldsymbol{\pi}$. These two facts imply the existence of $\boldsymbol{\lambda}_i \in R^{n+L+2M+1}$ and $\delta_i \in R$ such that $(\boldsymbol{x}\|\boldsymbol{\pi}) \cdot \boldsymbol{\lambda}_i + \delta_i = f_i(\alpha_r)$, for $i \in [L]$.

In a similar note, we observe that there exist $\boldsymbol{\lambda} \in R^{n+L+2M+1}$ and $\delta \in R$ such that $(\boldsymbol{x}\|\boldsymbol{\pi}) \cdot \boldsymbol{\lambda} + \delta = g(\alpha_r)$. The verifier makes these $L + 1$ queries to learn $\boldsymbol{f}(\alpha_r)$ and $g(\alpha_r)$, and another query to learn $g(\alpha_M)$. Finally, V checks that $g(\alpha_r) = G(\boldsymbol{f}(\alpha_r))$, and also that $g(\alpha_M) = 0$.

Security. The security analysis, including completeness and soundness, follows in the same way as in the proof of Theorem 4.3 in [Bon+19]. ☐

[10] A G-gate is a arithmetic sub-circuit of fixed degree, e.g., 2 in the case of a multiplication. See Section 4.2 in [Bon+19]

Corollary 5 (Corollary 1, restated). *Let $C : R^L \to R$ be a degree-2 arithmetic circuit. Let $A : R^n \to R$ and $A_1, \ldots, A_M : R^n \to R^L$ be affine functions. Then there exists a strong HVZK fully linear PCP for the language*

$$\mathcal{L}_{C,A,A_1,\ldots,A_m} = \{\boldsymbol{x} \in R^n \mid \sum_{i=1}^{M} C(A_i(\boldsymbol{x})) = A(\boldsymbol{x})\}, \tag{9}$$

that has the following efficiency metrics:

– *Proof length $O(L\sqrt{M})$ elements of R;*
– *Query complexity of $O(\sqrt{M} \cdot L)$;*
– *Soundness error of $\frac{2\sqrt{M}}{\omega_R - \sqrt{M}}$.*

Proof. We proceed as in [Bon+19, Corollary 4.9]. Let $K \mid M$. Define the degree-2 gate $G : R^{LK} \to R$ as follows: on input $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_K) \in R^{LK}$, where $\boldsymbol{z}_j \in R^L$, G outputs $\sum_{j=1}^{K} C(\boldsymbol{z}_j)$. We can see then that $\mathcal{L}_{C,A,A_1,\ldots,A_m}$ is recognized by a circuit with $M' = M/K$ G-gates, each having $L' = LK$ inputs. By Theorem 1, this can be proven with a PCP of length $L' + 2M' + 1 = LK + 2M/K + 1$, query complexity $L' + 2 = LK + 2$, and soundness $2M'/(\omega_R - M') = 2MK^{-1}/(\omega_R - MK^{-1})$. We can take $K = \sqrt{M}$ (assuming M is a perfect square for simplicity) to obtain a proof length of $O(L\sqrt{M})$, query complexity of $O(L\sqrt{M})$, and soundness $\frac{2\sqrt{M}}{\omega_R - \sqrt{M}}$. □