On the Power of Sumcheck in Secure Multiparty Computation

Zhe Li[1], Chaoping Xing[2], Yizhou Yao[2], Chen Yuan[2]

Institute of Mathematics and Interdisciplinary Sciences,
Xidian University, Xi'an, China
lizh0048@e.ntu.edu.sg
School of Computer Science,
Shanghai Jiao Tong University, Shanghai, China
{xingcp,yaoyizhou0620,chen_yuan}@sjtu.edu.cn

Abstract. Lund et al. (JACM 1992) invented the powerful Sumcheck protocol that has been extensively used in complexity theory and in designing concretely efficient (zero-knowledge) arguments. In this work, we systematically study Sumcheck in the context of secure multi-party computation (MPC). Our main result is a new unified framework for lifting semi-honest MPC protocols to maliciously secure ones, with a small *constant* multiplicative overhead in *both* computation and communication. In general, our approach applies to any semi-honest, linear secret-sharing based dishonest majority MPC secure up to additive attacks, where linear secret-sharing can be enhanced with an authentication mechanism. At a high-level, our approach has a highly distributive flavor, where the parties jointly emulate a Sumcheck prover to prove the correctness of MPC semi-honest evaluations in zero-knowledge, while simultaneously emulating a Sumcheck verifier to verify the proof themselves.

Equipped with our new techniques, we design a SPDZ-style MPC protocol with $5N + O(\log N)$ online communication per party and sublinear preprocessing based on efficient pseudorandom correlation generators (PCGs), where N is the circuit size. This substantially improves the $6N$ communication achieved in Le Mans (CRYPTO 2022), the state-of-the-art in the SPDZ line of works. Technically, the savings are obtained by using a Sumcheck-based mechanism to check N *unverified* authenticated multiplication triple relations, which requires only $N + 1$ *standard Beaver triples* and $O(\log N)$ random authenticated shares, rather than N additional unverified authenticated triples needed by a "sacrifice" strategy.

We also show concrete benefits for honest majority MPC protocols based on Shamir secret sharing. Compared to the best known approach in this scenario (Goyal et al. CRYPTO 2020) based on *fully linear interactive oracle proofs* (FLIOPs), asymptotically we achieve the same $O(N)$ additive overhead in computation and $O(\log N)$ additive overhead in communication. However, we replace the $O(\log N)$ double sharings used there with $O(\log N)$ random sharings, and reduce the soundness error from $O(\frac{N}{|\mathbb{F}|})$ to $O(\frac{\log N}{|\mathbb{F}|})$, where $\mathbb{F}$ is the underlying field.

Table of Contents

1 Introduction . . . 2
1.1 Our Contributions - A New Framework . . . 4
1.2 Organizations . . . 8
2 Technical Overview . . . 9
3 Preliminaries . . . 15
4 Distributed Sumcheck . . . 17
4.1 Distributed Sumcheck . . . 17
4.2 Sumcheck for Multiplication Verification in MPC . . . 20
5 MPC with an Honest Majority . . . 21
6 MPC with a Dishonest Majority . . . 26
6.1 Checking Unverified Authenticated Beaver Triples . . . 27
6.2 Malicious MPC with a Dishonest Majority . . . 32
A More Preliminaries & Functionalities . . . 39
A.1 Shamir Secret Sharing Scheme . . . 39
A.2 SPDZ Sharing [23] . . . 39
A.3 Functionalities . . . 39
B Related Work . . . 43
C Sumcheck for Other Relations . . . 45
C.1 Multiplication Triple . . . 45
C.2 Circuit Dependent Preprocessing . . . 45
C.3 Rank One Constraint System . . . 46
C.4 Circuit Evaluation . . . 46
D Security Proofs . . . 47
D.1 Proof of Theorem 5.2 . . . 47
D.2 Proof of Theorem 6.1 . . . 49
E Concrete Complexity Analyses . . . 55
E.1 Concrete Computation Complexity of Distributed Sumcheck . . . 55
E.2 Concrete Complexity of [36] . . . 57

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to jointly compute a function on their inputs, where no information would be revealed except the output, in spite of the existence of adversaries. Since its birth in mid 80s [57,32,15,3], numerous works have been investigating on the boundaries of MPC in terms of feasibility and complexity. Though, the focus of intensive research has shifted to designing concretely efficient MPC protocols in the past decade. These works often follow a modular design, where a simple protocol with semi-honest security is put forward firstly, then powerful but usually heavy techniques will be applied to achieve a stronger notion of security, e.g., malicious security. This

methodology has proven to yield efficient protocols, measured by the overhead introduced during the compilation process.

Hence, a dream goal in the context of MPC is to realize protocols with higher security as efficient as in semi-honest setting. Towards this end, tremendous progress has been made and various frameworks have been established and developed, fitting wide scenarios. In summary, these include GMW compiler [32], IPS compiler [41], AMD circuits [30,29], dual execution [17,47], fully linear probabilistically checkable proofs (FLPCPs) and fully linear interactive oracle proofs (FLIOPs) [7], etc. At the current stage, dual execution and FLIOP turn out to be the most practically efficient solutions among these frameworks.

Informally, the dual execution approach evaluates the circuit one more time for checking, hence incurring a constant multiplicative overhead in both communication and computation to the semi-honest protocol. Moreover, as shown in [17] and [47], the communication overhead in honest majority and dishonest majority can be as low as only $2\times$ and $3\times$, respectively.

For honest majority MPC protocols, FLIOP-based approaches [36,11] outperform dual execution in the sense that the malicious security comes for free in terms of communication, since FLIOPs [7] can prove N multiplications in $O(\log N)$ communications and $O(N)$ computation. However, FLIOPs cannot directly apply to dishonest majority MPC, since the underlying secret sharing scheme is required to be *robust*, i.e., the secret as well as the corrupted parties' shares can be determined by honest parties' shares. To circumvent this issue, [12] propose a star-sharing framework, admitting sublinear communication for checking evaluations in the presence of a trust dealer. A follow-up work then shows that the trusted dealer can be emulated in $O(\sqrt{N})$ communications [13] by introducing additively homomorphic encryptions. However, this comes at the cost of $O(N^{1.5})$ additional computation per party. It is worth to mention that the sub-quadratic computation in [13] has recently been optimized to linear by an independent concurrent work [14]. However, their online protocol no longer remains information-theoretically secure in the preprocessing model and the concrete efficiency appears limited due to the heavy use of LPN assumptions.

Despite impressive progress, significant concrete efficiency gaps persist between semi-honest and malicious security, in particular in dishonest majority.

Can we beat the $3\times$ *online communication overhead in [47] without resorting to additional expensive cryptographic assumptions?*

We answer the above question in the affirmative by introducing the powerful Sumcheck technique [46] from complexity theory, originally used to establish the seminal result $\mathsf{IP} = \mathsf{PSPACE}$ [49]. Since the milestone work GKR [33,34], Sumcheck has been widely employed in concretely efficient succinct non-interactive arguments of knowledge (SNARKs) [54,56,59,58,25], and recently in interactive zero-knowledge proofs [24,43]. However, in the context of MPC, the full potential of Sumcheck has not been fully recognized. While [7,18] showed that the GKR protocol can be understood as an FLIOP (making it applicable to MPC scenarios), it has not been studied whether GKR or Sumcheck can elevate MPC to a high level.

1.1 Our Contributions - A New Framework

In this work, we focus on MPC for computing arithmetic circuits over a sufficiently large finite field $\mathbb{F}$. We distill the powerful sumcheck technique from the interactive proofs literature, and present a generic, information-theoretic compiler for lifting semi-honestly secure MPC to malicious security, which simultaneously preserves the *computation* and *communication* complexity. Our compiler supports an arbitrary number of parties, tolerating any fraction of corrupted parties, and can be naturally extended to arbitrary finite fields or rings, such as $\mathbb{Z}_{p^k}$.

In an interactive proof for some language $\mathcal{L} \subseteq \{0,1\}^n$, given a public input $x \in \{0,1\}^n$, the prover convinces to the verifier that x belongs to $\mathcal{L}$. We observe that the settings of interactive proofs and MPC are quite different as follows:

1. Informally, most secret-sharing based MPC protocols are symmetric and highly distributive, whereas interactive proofs are not. In MPC, all parties have equal rights and similar computational resources[1]. In contrast, interactive proofs ensure soundness against the prover, i.e., the prover must not be able to fool the verifier with a false statement. Furthermore, the verifier is assumed to be computationally weaker than the prover, so that he could not independently verify the statement himself. Otherwise, the proofs become trivial.
2. In MPC, the computation is "shared" among parties, with each holding a piece of the input, such that no single party knows the entire input. In contrast, in interactive proofs, both the prover and the verifier have access to the complete statement.

While the above two major differences inherently limit the benefits of interactive proofs, we will show an exception through the Sumcheck protocol. A feasible approach is to follow the GMW paradigm [32] by designing tailored zero-knowledge protocols from Sumcheck, and having parties prove that each message is honestly generated. However, such method incurs a significant computational overhead.

To minimize this overhead, we instead have the parties first compute (via semi-honest secure MPC protocols) and later verify the values. At a high level, the verification is performed in a distributed manner: the parties generate shares of the proof from their respective shares of the statement and jointly verify the secret-shared proof.

Since the statement to be verified is shared and not known to any single party, we require the underlying secret sharing scheme to offer an "authenticated" property[2]. Informally, authentication means a secret-shared value is "committed" such that dishonest openings are detected with overwhelming probability. Otherwise, an adversary could arbitrarily modify the shared statement, making the check meaningless. In addition, we require the semi-honest protocols to be secure up to additive attacks as discussed in [30], meaning the adversary can only inject

[1] Though some MPC protocols introduce a king party to optimize communication complexity, the essence of MPC protocol remains symmetric.

[2] This property is slightly weaker than the "robust" property in the FLIOP approach [7,11,12,20]. Informally, robustness means that honest parties' shares determine the secret as well as corrupted parties' shares.

additive errors into the input wires of gates. As shown in [30], many semi-honest MPC protocols based on secret sharing, with either an honest majority or a dishonest majority, satisfy this property. Hence, it suffices to verify the computation of multiplication gates in the circuit for most linear secret sharing schemes.

New protocols in the dishonest majority setting. We first consider a simple problem of checking N SPDZ-style unverified authenticated multiplication triples. The previous "sacrificing" method requires N additional unverified authenticated triples, incurring at least a $2\times$ overhead. Such overhead is undesirable, since typically producing authenticated multiplication triples dominates the overall complexity. Notably the method of [13] only requires $O(\sqrt{N})$ communication, but introduces $O(N^{1.5})$ computation. By leveraging the Sumcheck techniques, we offer an efficient alternative solution. The results are summarized as follows, which is given by Theorem 6.1.

Lemma 1.1 (informal). *There exists a maliciously secure protocol that checks N unverified SPDZ-style authenticated multiplication triples with $O(N)$ computation per party, $2N + O(\log N)$communication per party, $O(\log N)$ correlated randomness, $O(\log N)$ rounds, and soundness error $O(\frac{\log N}{|\mathbb{F}|})$.*

Specifically, the correlated randomness required for the protocol consists of $N+1$ *standard Beaver triples* (respect to a PCG seed of size $O(\log N)$), $3\log N + 5$ authenticated secret shares, which is significantly easier and cheaper to produce than N authenticated triples. A major downside is the inherent $O(\log N)$ round complexity, while both the approaches of [13] and sacrifice can have $O(1)$ rounds.

We then show concrete benefits of applying distributed Sumcheck in dishonest majority MPC. Following the paradigm of [47], our preprocessing uses programmable PCGs to generate *partially* authenticated multiplication triples. While the online phase of [47] employs a "dual execution" mechanism to verify multiplication gates, achieving sublinear preprocessing communication and $6 + o(1)$ field elements per multiplication gate per party. At a high level, we pay one opening to transform a *partially* authenticated triple to an unverified authenticated triple and replace dual execution with distributed Sumcheck, reducing communication to $5+o(1)$, only 2.5 *times* that of the semi-honest security setting.

Theorem 1.2 (informal). *There exists a maliciously secure dishonest majority MPC protocol, where the online phase is information-theoretic and sends $5 + o(1)$ field elements per multiplication gate per party, while the offline phase has $o(1)$ communications assuming secure programmable PCGs.*

New protocols in the honest majority setting. As a byproduct, we offer concrete benefits against [36] in honest majority. Let $\mathcal{C}$ be a circuit over $\mathbb{F}$ with N multiplication gates and consider an M-party setting with $t < M/2$ corruptions. Let $[\cdot]_t$ denote a Shamir secret sharing scheme (refer to Appendix A.1), which is robust (hence authenticated), linear and moreover multiplicative. We have the following results implied by Theorem 5.2.

Theorem 1.3 (informal). *Given a semi-honest protocol based on Shamir secret sharing (e.g., the DN protocol [22]), there exists a maliciously secure protocol that computes $\mathcal{C}$ with additional $10N + O(M \log N)$ computation per party, $O(\log N)$ communication per party, $O(\log N)$ random Shamir shares and $O(\log N)$ random coins, $O(\log N)$ rounds, and soundness $O(\frac{\log N}{|\mathbb{F}|})$ over the semi-honest protocol.*

Specifically, the $10N$ term in computation is from emulating the Sumcheck prover, while the $O(M \log N)$ term comes from reconstructing the Sumcheck proof (Sumcheck verification only requires $O(\log N)$ computation in our case). Concretely, each party broadcasts $4 \log N + 6$ field elements (for reconstructions), makes $\log N + 1$ calls to a coin-tossing functionality $\mathcal{F}_{\mathsf{Coin}}$, and consumes $3 \log N + 5$ random Shamir shares (for privacy). In particular, we can employ the king party idea to further reduce the broadcast messages and computation per party. Consequently, the king party runs additional $O(N + M \log N)$ field operations, while each remaining party only runs additional $O(N)$ field operations.

Compared to [36], we only consume additional $O(\log N)$ random sharings rather than $O(\log N)$ double sharings, which are cheaper to generate. Moreover, we show an efficient approach that reduces the soundness error of [36] from $O(\frac{N}{|\mathbb{F}|})$ to $O(\frac{\log N}{|\mathbb{F}|})$, which enhance the security level by ≈ 15-bit for $N = 2^{20}$.

Methodology of our compiler. Generally speaking, our compiler works for any semi-honest MPC protocol based on linear secret sharing schemes and secure up to additive attacks, where authentication can be added. Moreover, our approach only incurs a constant multiplicative overhead in both computation and communication for both honest majority and dishonest majority. The compiler follows from the following key steps:

1. We first transform the verification of N multiplication gates into a related sumcheck problem, on which we can apply the Sumcheck protocol. The transformation critically leverages the parallelism of multiplication gates.
2. We then carefully translate the sumcheck problem to an equivalent form that preserves zero-knowledge. Informally, the verifier learns nothing about the inputs to the multiplication gates when applying the Sumcheck protocol on this new form. This step consumes only $O(\log N)$ correlated randomness.
3. The parties jointly emulate both the Sumcheck prover and Sumcheck verifier, effectively executing a "virtual" Sumcheck protocol on the transformed problem from Step 2. In the end, each party obtains a complete Sumcheck proof and decides its validity based on the transcripts.
   The zero-knowledge property guarantees the privacy of the inputs to multiplication gates. The additional communication consists of the Sumcheck proof size ($O(\log N)$) and the cost of distributively generating the proof (free for multiplicative secret sharing, otherwise $O(N)$).
4. The distributed proof generation is optimized so that each party performs only $O(N)$ field multiplications, independent of the number of parties.

Intuition of our benefits and relations to FLIOP. Here, we first address a critical difference from FLIOP-based approaches in the distributed prover setting [7,37,26]. That is we no longer require the full Sumcheck proof to be robustly

secret-shared and reconstructed. Instead, we rely on secret sharing schemes with authentication and in fact the secret-shared statement is "committed". Such a difference is particularly crucial in the dishonest majority setting. Let us briefly explain this interesting phenomenon.

Let us briefly recap the Sumcheck protocol. The classical Sumcheck protocol enables a lightweight verifier to validate the computation of the summation of a multi-variate polynomial over a binary hypercube, e.g., $H \stackrel{?}{=} \sum_{\mathbf{X}\in\{0,1\}^n} f(\mathbf{X})$. In a bare-bone sketch, the Sumcheck protocol works as follows: in each round i, the prover sends a univariate polynomial

$$f_i(X_i) := \sum_{x_{i+1},\ldots,x_n\in\{0,1\}} f(r_1,\ldots,r_{i-1},X_i,x_{i+1},\ldots,x_n)$$

and the verifier returns a challenge r_i. The verifier checks that

$$H = f_1(0)+f_1(1), f_{i-1}(r_{i-1}) = f_i(0)+f_i(1) \text{ for } i \in [2,n] \text{ and } f_n(r_n) = f(r_1,\ldots,r_n).$$

In particular, the Sumcheck protocol is *doubly* efficient for some structured polynomials (e.g., the product of ℓ multilinear polynomials), requiring $O(\ell \cdot 2^n)$ computation for the prover and $O(\ell n)$ for the verifier. Essentially, the Sumcheck protocol boils down the original Sumcheck problem (i.e., $H \stackrel{?}{=} \sum_{\mathbf{X}\in\{0,1\}^n} f(\mathbf{X})$) to checking a single random evaluation of f (i.e., $f_n(r_n) \stackrel{?}{=} f(r_1,\ldots,r_n)$). The prover's consistency and truthfulness are enforced in the sense that when starting with a wrong claim, if the prover fools in intermediate rounds, then eventually with high probability $f_n(r_n) \neq f(r_1,\ldots,r_n)$. Therefore, there is no need to guarantee each $f_i(X_i)$ provided by the prover is computed correctly. Informally, interactions buy the verifier the ability to ensure that the prover faithfully computes and adheres to intermediate values of the computation without requiring the prover to explicitly provide or materialize those values. Reflected in MPC settings, each $f_i(X_i)$ has not to be robustly shared and reconstructed. However, H and $f(r_1,\ldots,r_n)$ should be authenticated and reconstructed correctly, making authenticated secret sharing an inherent requirement.

FLIOP v.s. Sumcheck. Surprisingly, we observe that known FLIOP constructions [7] are essentially equivalent to Sumcheck when applied to honest majority MPC. To illustrate this, we analyze a concrete instantiation from [36]. Their approach can be interpreted as follows:

1. Reduce the verification of $\mathbf{W}_o \stackrel{?}{=} \mathbf{W}_\ell * \mathbf{W}_r$ to a sumcheck problem:

$$c = \sum_{i=0}^{N-1} \lambda^i \cdot W_o(i) \stackrel{?}{=} \sum_{i=0}^{N-1} \underbrace{\lambda^i \cdot W_\ell(i)}_{\mathbf{a}(i)} \cdot \underbrace{W_r(i)}_{\mathbf{b}(i)},$$

   where $\lambda \xleftarrow{\$} \mathbb{F}$ and $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$.

2. Inductively reduce the underlying sumcheck problem by half at each step using the evaluation set $\{0, 1\}$ until get a constant size. This step is essentially equivalent to applying the Sumcheck protocol to the above equation. Informally, suppose $[c']_t, [\mathbf{a}']_t, [\mathbf{b}']_t$ is the sumcheck problem after one reduction with respect to challenge $r_1 \xleftarrow{\$} \mathbb{F}$ in [36], and $f_1(X_1)$ is the polynomial that the prover sends in the first Sumcheck round. Through appropriately ordering the inputs, we obtain $f_1(r_1) = c'$, while $[\mathbf{a}']_t, [\mathbf{b}']_t$ specify a polynomial $f_2(X_2)$ such that $c' = f_1(r_1) = f_2(0) + f_2(1)$. This demonstrates that [36] naturally fits to the Sumcheck framework. See Appendix E.2 for details.

In sketch, there are two major technical differences between our distributed Sumcheck approach and [36]:

(i) Instead of using $(1, \lambda, \ldots, \lambda^{N-1})$ as in [36], we employ a more structured choice of random coefficients for Step 1. This substantially improves soundness, reducing the error probability to $O(\frac{\log N}{|\mathbb{F}|})$ rather than $O(\frac{N}{|\mathbb{F}|})$ in [36].

(ii) We fully exploit the round-by-round soundness nature of Sumcheck by directly reconstructing the secret-shared polynomial $[f_j(X_j)]_{2t}$ without consuming double sharings (refer to Definition A.2) to compute $[f_j(X_j)]_t$ and thus $[f_j(r_j)]_t$ as in [36]. However, we need to pay more attention to make the Sumcheck protocol zero-knowledge, as plain $f_j(X_j)$ would leak information about the inputs.

Remark 1.4. On the other hand, the above (ii) indicates that [36] can be simply extended into dishonest majority: first using standard Beaver triples to compute additive sharings $[f_j(X_j)]$ and then turn them into authenticated sharings $[\![f_j(X_j)]\!]$ via consuming random authenticated sharings. This shows an alternative way to use FLIOP in dishonest majority, while previous works follow the star-sharing framework [12,13,14].

Concrete Comparisons. Our compiler improves over the state-of-the-art in both dishonest majority and honest majority settings. Below, we present a comparison table. W.l.o.g., we consider SPDZ-style unverified authenticated secret sharing for dishonest majority, and Shamir secret sharing for honest majority. For other secret sharing schemes, our compiler is plausible to offer concrete benefits as well. We leave it as future work.

As discussed above, our approach shares similarities with [36], which builds on FLIOPs. In addition to the advantage of avoiding double sharings, our method also offers higher security-level for large circuits compared to [36] (due to (i)), e.g., $\approx$ 15-bit higher security for circuits size 2^{20}. Moreover, we believe that adapting our distributed Sumcheck approach to the paradigm of [12,13,14] in the dishonest majority setting would yield further benefits. We leave it as future work.

1.2 Organizations

In Section 2, we present an overview of our techniques. Section 3 gives necessary preliminaries (more in Appendix A). For a better readability to readers who are

Corrupt.	Methods	Computation	Communication	Randomness	Round
$t < M$	Le Mans [47]	$O(MN)$	$4MN$	$O(\log N)$	$O(1)$
	[13]	$O(MN^{1.5})$	$O(M^2\sqrt{N})$	$O(\sqrt{N})$	$O(1)$
	This work	$O(MN)$	$3MN + O(M \log N)$	$O(\log N)$	$O(\log N)$
$t < M/2$	[36]	$O(MN)$	$O(M \log N)$	$3 \log N$ DS $+ 2$ SS	$O(\log N)$
	[11]	$O(M^2N)$	$O(M^2 \log N)$	M DS $+ 2$ SS	$O(1)$
	This work	$O(MN)$	$O(M \log N)$	$3 \log N$ SS	$O(\log N)$

Table 1. Comparisons for costs of lifting semi-honest MPC protocols to malicious security, where N is the number of multiplication gates and M is the number of parties. We consider total computation and total communication costs. For dishonest majority, PCGs are used to compress correlated randomness storage. For honest majority, SS denotes a random Shamir sharing while DS denotes a random double sharing of Shamir sharing. Insignificant terms and the costs for emulating $\mathcal{F}_{\mathsf{Coin}}$ are omitted.

not familiar with Sumcheck protocols, we start with a warm-up construction of the distributed Sumcheck protocol in Section 4, which only proves trivial statements in MPC. Then in Section 5, we consider verification of multiplications in the simpler honest majority case, where we rely on a multiplicative secret sharing scheme, i.e., Shamir's scheme. Finally, in Section 6 we show a distributed Sumcheck protocol for checking multiplications in the dishonest majority, which is more challenging as the underlying secret sharing scheme is no longer multiplicative.

2 Technical Overview

Informally, the goal is to find efficient approaches that verify statements being shared among the parties. In the literature, typically there are two approaches that employ proof systems in different flavors.

GMW-style vs. Distributed proof. The first is of the celebrated "GMW"-style [32], where each party proves that he sent the correct message in the semi-honest protocol. The state-of-the-art proof system suitable for this framework is the FLIOP [7]. At a high level, FLIOP-based approach first reduce the statement to sub-statements that "capture" each party P_i's honest behaviors, where P_i (as an FLIOP prover) knows the sub-statement i, while the remaining parties (jointly emulate an FLIOP verifier) holds secret sharings of it. Then it suffices to apply FLIOPs in a black-box way, yielding sublinear communication in the honest majority setting [7,10,20], and in the dishonest majority setting [12] (assuming a semi-honest dealer). In terms of computational complexity, the underlying FLIOPs have linear prover time and verification time.

Regardless of the efficiency of FLIOPs, we find the above approach contains "redundancy", in the sense that the "proof-of-behavior" is only essential to the

corrupted parties, since honest parties will never deviate the protocol[3]! However, this redundancy appears to be unavoidable, as there is no magic method to identify corrupted parties for free. Consequently, each party's computation also scales linearly with the number of parties in [7,10,12,20], which limits the scalability.

The second approach allows the parties to jointly emulate a virtual prover, who knows the secret-shared statement, and proves the statement to themselves. In other words, the proof is generated distributively and then verified by the parties. This paradigm is intuitively more efficient, and a concrete, informal example is the global MACs technique used in SPDZ [23], which improves upon the pairwise MACs used in BDOZ [4]. On the one hand, if corrupted parties cheat in the proof generation, soundness ensures rejection with high probability. On the other hand, every party contributes to the proof, and no one's computation is meaningless. Despite potential efficiency benefits, there remains a major question: how to generate a proof without even knowing the statement. Existing works answer with FLIOP-based solutions by utilizing the multiplicative property of Shamir secret sharing in the honest majority [7,11,36,37], and introducing homomorphic encryption schemes in the dishonest majority [13]. Note that [36] simultaneously achieves additional linear computation and sublinear communication via Shamir secret sharing. However, the inherent robustness required by FLIOPs poses a barrier in the dishonest majority setting.

Towards distributed interactive proof. The huge gap between interactive proofs and MPC raises numerous challenges that have not been elaborately explored before. Our first key observation comes from the fact that for most applications of Sumcheck, the multi-variate polynomial to be summed up is of low-degree and moreover highly structured. For instance, the GKR [33] protocol (for proving circuit satisfiability) runs Sumcheck on polynomials of the form

$$f_z(\mathbf{X}, \mathbf{Y}) := \widetilde{\mathsf{mult}}(\mathbf{X}, \mathbf{Y}, z)\widetilde{W}(\mathbf{X})\widetilde{W}(\mathbf{Y}) + \widetilde{\mathsf{add}}(\mathbf{X}, \mathbf{Y}, z)\big(\widetilde{W}(\mathbf{X}) + \widetilde{W}(\mathbf{Y})\big),$$

where $\widetilde{W}(\cdot), \widetilde{\mathsf{mult}}(\cdot, \cdot, z)$ and $\widetilde{\mathsf{add}}(\cdot, \cdot, z)$ are multilinear polynomials. In addition, $\widetilde{\mathsf{mult}}$ and $\widetilde{\mathsf{add}}$ are called *wiring predicates* and only depend on the circuit typology, and $\widetilde{W}(\cdot)$ is the multilinear extension (MLE) of wire values. Hence, in MPC scenarios, if the underlying secret sharing scheme is *multiplicative*, then parties can locally compute the shares of $f_z(\mathbf{X}, \mathbf{Y})$ for *any* evaluation point without additional interaction. This suggests that GKR might be suitable for the distributed proof approach in MPC. Additionally, GKR is *doubly efficient* and has been optimized to linear prover time [56]. These together make it possible to achieve distributed interactive proof systems with linear computation and sublinear communication.

Despite intuitively feasible, there remain a lot of issues to address since interactive proofs and MPC are totally different. First, the zero-knowledge property (resp. privacy) is not necessarily required for interactive proofs, while in contrast, MPC becomes trivial if privacy is not required. While many interactive proofs can be made zero-knowledge almost for free [51, Chapter 13], it is unclear

[3] We only consider static corruptions in this work. One can think of that interactive proofs become trivial if the prover always tells the truth.

how to apply such tricks in MPC. Secondly, security analyses become significantly more complicated. In (zero-knowledge) interactive proofs, the security analysis is modular and it suffices to consider two cases: a corrupted prover interacting with an honest verifier (for soundness), or an honest prover with a corrupted verifier (for zero-knowledge). When diving into MPC settings, it will be the case that corrupted provers may collude with corrupted verifiers to learn honest parties' inputs (as all parties jointly emulate the prover as well as the verifier). Hence, a simulation-based proof that guarantees privacy and soundness simultaneously is required. Thirdly, it is unclear how to adapt the techniques for achieving linear prover time in [56] to MPC settings, since the statement is secretly shared among the parties. Finally, the above three issues become much more challenging in the dishonest majority case, where multiplicative secret sharing is no longer viable. Basically, we need to minimize the multiplications during distributed proof generation. Otherwise, our Sumcheck-based approach offers no advantage over prior methods.

Distributed Sumcheck for verifying multiplications. In a bird's eye view, there are four key steps in building our compiler:

1. Reduce verification of multiplications to sumcheck problems.
2. Reformulate the sumcheck problem to guarantee privacy.
3. Enable joint emulation of the Sumcheck prover and verifier by all parties.
4. Optimize computational and communication complexity.

For a better readability, we first briefly introduce related notations.

Notations. Assume $N = 2^n$. The goal is to verify $\mathbf{W}_o \stackrel{?}{=} \mathbf{W}_\ell * \mathbf{W}_r$, where $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$ are secret-shared inputs and $*$ denotes the entry-wise multiplication. As in the literature, we view vectors as functions, for instance, $\mathbf{W}_o$ defines $W_o : \{0,1\}^n \to \mathbb{F}$, such that $W_o(i)$ equals to the i-th entry of $\mathbf{W}_o$. Denote the multilinear extension(MLE) of W_o by $\widetilde{W}_o$. Namely,

$$\widetilde{W}_o(y_1, \ldots, y_n) = \sum_{\omega \in \{0,1\}^n} W_o(\omega) \cdot \boldsymbol{\chi}_\omega(y_1, \ldots, y_n),$$

where $\boldsymbol{\chi}_\omega(x_1, \ldots, x_n) := \prod_{i=1}^n (x_i\omega_i + (1-x_i)(1-\omega_i))$ is referred as the multilinear Lagrange basis. We refer to Definition 3.1 for a formal definition of MLEs.

The reduction. A direct approach is to follow the insightful idea proposed by Goldwasser et al. [33,34], by viewing the verification as evaluating a one-layer circuit consisting of N multiplication gates. Indeed, this is our starting point, and we find that carefully generalizing sophisticated techniques suffices for honest-majority MPC. However, we fail in obtaining concrete benefits in dishonest majority. In fact, such characterizations contain redundancy, since GKR applies to general layered circuits with both addition gates and multiplication gates, while in our case, there are only multiplications.

We take a step further on utilizing MLEs and propose a new tighter reduction that yields better efficiency. Inspired by the approach of [53] for improving GKR

efficiency in the *single instruction multiple date* (SIMD) setting, we introduce the point predicate: $P_z(y) : \{0,1\}^n \to \{0,1\}$, which evaluates to 1 if and only if $y = z$ and otherwise 0. Then for $y, z \in \{0,1\}^n$, we have

$$W_o(z) = \sum_{y \in \{0,1\}^n} P_z(y) \cdot W_\ell(y) \cdot W_r(y).$$

The key observation is that the Lagrange basis $\boldsymbol{\chi}_y(z) = \boldsymbol{\chi}_z(y) = \prod_{i=1}^n (z_i y_i + (1-z_i)(1-y_i))$ is essentially a point predicate. Then, we have that

$$\begin{aligned}\widetilde{W}_o(z) &= \sum_{\omega \in \{0,1\}^n} \boldsymbol{\chi}_\omega(z) \cdot W_o(\omega) = \sum_{\omega \in \{0,1\}^n} \boldsymbol{\chi}_\omega(z) \cdot \Big(\sum_{y \in \{0,1\}^n} P_\omega(y) \cdot W_\ell(y) \cdot W_r(y) \Big) \\ &= \sum_{y \in \{0,1\}^n} \boldsymbol{\chi}_y(z) \cdot W_\ell(y) \cdot W_r(y) = \sum_{\mathbf{Y} \in \{0,1\}^n} \underbrace{\boldsymbol{\chi}_{\mathbf{Y}}(z) \cdot \widetilde{W}_\ell(\mathbf{Y}) \cdot \widetilde{W}_r(\mathbf{Y})}_{f_z(\mathbf{Y})},\end{aligned} \tag{1}$$

which holds for any $z \in \mathbb{F}^n$. Note that the last two equalities follow from that $P_\omega(y)$ is a point predicate (which equals to 1 only if $y = \omega$, so that we can merge y and ω), and the definition of multilinear extensions (i.e., for $\mathbf{Y} \in \{0,1\}^n$, $\widetilde{W}_\ell(\mathbf{Y}) = W_\ell(\mathbf{Y})$, and $\widetilde{W}_r(\mathbf{Y}) = W_r(\mathbf{Y})$). In the interactive proofs setting, it suffices to run the Sumcheck protocol on the above relation with a verifier-sampled random $z \xleftarrow{\$} \mathbb{F}$. Recall that in each round j of the Sumcheck protocol, the verifier receives a polynomial $f_z^j(Y_j) := \sum_{b_{j+1} \ldots b_n \in \{0,1\}^n} f_z(r_1 \ldots r_{j-1}, Y_j, b_{j+1} \ldots b_n)$, and responds with $r_j \xleftarrow{\$} \mathbb{F}$ to the prover.

Adding zero-knowledge. If parties jointly emulate a Sumcheck prover on proving the above Eq.(1), the proof messages (e.g., $f_z^j(Y_j)$) would leak information about the inputs. To keep privacy, we employ methods inspired from Libra [56]. At a high level, we mask the equation with a random *sparse* low degree polynomial and the inputs with random low degree vanishing polynomials, similar to Libra. However, there are subtle differences in detail. Jumping ahead, we basically run the Sumcheck protocol on a new equation of the following form:

$$\begin{aligned}\widetilde{W}_o(z) + G &= \sum_{\mathbf{Y} \in \{0,1\}^n} \underbrace{\big(\widetilde{W}_\ell(\mathbf{Y}) + aY_n(1-Y_n)\big)}_{\widetilde{W}'_\ell(\mathbf{Y})} \underbrace{\big(\widetilde{W}_r(\mathbf{Y}) + bY_n(1-Y_n)\big)}_{\widetilde{W}'_r(\mathbf{Y})} \boldsymbol{\chi}_z(\mathbf{Y}) + g(\mathbf{Y}) \\ &= \sum_{\mathbf{Y} \in \{0,1\}^n} \big(f'_z(\mathbf{Y}) + g(\mathbf{Y})\big),\end{aligned} \tag{2}$$

where $G = \sum_{\mathbf{Y} \in \{0,1\}^n} g(\mathbf{Y})$. Let us briefly explain the intuition. First, a, b are uniformly random and $g(\mathbf{Y})$ is a random n-variate sparse polynomial of low degrees. Informally, $g(\mathbf{Y})$ as a masking polynomial, prevents the verifier from learning information about $f'_z(\mathbf{Y})$. We note that in the zero-knowledge interactive proof setting of Libra, $g(\mathbf{Y})$ is determined by both prover and verifier. Loosely speaking, the prover first commits to a polynomial $g'(\mathbf{Y})$ and the verifier samples a random $\rho \xleftarrow{\$} \mathbb{F}$, defining $g(\mathbf{Y}) := \rho \cdot g'(\mathbf{Y})$. This allows to build an extractor that

extracts the witness and implies the resulting Sumcheck protocol is honest-verifier zero-knowledge. However, things will be different in the MPC setting, due to the existence of *at least one* honest party. One the one hand, we do not have to prove the stronger notion of knowledge soundness as in zero-knowledge Sumcheck. Intuitively, the protocol will be run only when honest parties have authenticated shares (that offer binding) of the statement and witness, which guarantees that if the generated proof is accepted, the emulated Sumcheck prover must have the knowledge of H as well as the polynomial f. One the other hand, one can naturally assume an ideal functionality that distributes shares of coefficients of $g(\mathbf{Y})$, ensuring privacy without relying on an honest verifier.

Finally, $aY_n(1-Y_n)$ and $bY_n(1-Y_n)$ are vanishing polynomials, evaluating to 0 for all $\mathbf{Y} \in \{0,1\}^n$. In contrast to interactive proofs, where vanishing polynomials can be arbitrary, here we use such tailored forms to minimize multiplications, which is particularly crucial in the dishonest majority setting. Essentially, $\widetilde{W}'_\ell(\mathbf{Y}), \widetilde{W}'_r(\mathbf{Y})$ are low-degree extensions of W_ℓ, W_r that agree to the multilinear extensions (except for variable Y_n), respectively. Running the Sumcheck protocol on such low-degree extensions yields efficiency benefits in computing $f'_z(r_1, \ldots, r_n)$. Now, the prover can reveal $\widetilde{W}'_\ell(r_1, \ldots, r_n)$ and $\widetilde{W}'_r(r_1, \ldots, r_n)$ to the verifier while keeping the privacy of $\mathbf{W}_\ell$ and $\mathbf{W}_r$.

Joint emulation. The key observation is that shares of $\mathbf{W}_\ell, \mathbf{W}_r$ imply shares of their MLEs $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$, while $\boldsymbol{\chi}_z(\mathbf{Y})$ is publicly known. Hence, if the secret sharing is multiplicative, the parties can *locally* compute shares of the Sumcheck proof

$$(f'^j_z + g^j)(Y_j) := \sum_{b_{j+1},\ldots,b_n \in \{0,1\}^{n-j}} (f'_z + g)(r_1, \ldots, r_{n-1}, Y_j, b_{j+1}, \ldots, b_n)$$

for each round j. By the soundness of Sumcheck, the proof need not be robustly shared and parties can directly reconstruct it in each round. Moreover, this property enables using standard Beaver triples to jointly compute the proof when the secret-sharing scheme is not multiplicative, e.g., SPDZ-sharing in the dishonest majority. As for jointly emulating the verifier, we leverage the public-coin nature of Sumcheck: security holds as long as the verifier tosses random coins, which will be visible to the prover as soon as tossed. Since at least one party is honest, an ideal functionality $\mathcal{F}_{\mathsf{Coin}}$ that samples $r_j \xleftarrow{\$} \mathbb{F}$ in each round suffices. We remark that in the last round n, the parties need to check

$$(f'^n_z + g^n)(r_n) \stackrel{?}{=} (f'_z + g)(r_1, \ldots, r_n) = (\boldsymbol{\chi}_z \widetilde{W}'_\ell \widetilde{W}'_r + g)(r_1, \ldots, r_n).$$

The right-hands values should be reconstructed in an authenticated manner, since it is computed after sampling r_n. This is guaranteed by the fact that $\widetilde{W}'_\ell(\mathbf{Y}), \widetilde{W}'_r(\mathbf{Y})$ and $g(\mathbf{Y})$ are essentially shared by a secret-sharing scheme with authentication, allowing parties to reveal one evaluation.

Linear time optimization. Thanks to the fact that MLEs are highly structured, the Sumcheck prover on proving Eq.(2) can pay only $O(N)$ computations and

$O(N)$ memory usage, by employing the well-known bookkeeping table algorithm. In a bare-bone sketch, in each round j, the prover maintains three bookkeeping tables (containing 2^{n-j+1} evaluations of $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y}), \boldsymbol{\chi}_z(\mathbf{Y})$, respectively) that allow to compute $f'^j_z(Y_j)$ efficiently with only $O(2^{n-j})$ operations. Additionally, updating the bookkeeping tables for the next round requires only $O(2^{n-j})$ operations. However, in the MPC setting, no one holds plaintext tables for $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$ (except for $\boldsymbol{\chi}_z(\mathbf{Y})$), since they depend on the secret-shared inputs. Instead, each party can locally compute shares of the bookkeeping tables, enabling an efficient distributed algorithm for computing secret-shared $f'^j_z(Y_j)$. Recall that $f'^j_z(Y_j)$ has not to be robustly shared, the shares can be locally computed without additional interaction if the secret-sharing scheme is multiplicative. While without multiplicative property, the algorithm is much more complicated. We resort to using standard Beaver triples and pay great efforts to minimize the number of consumed triples.

Minimize multiplication triples. Concretely, for round j, it suffices for the Sumcheck prover to compute 4 evaluations, e.g.,$\{0, \pm 1, 2\}$, of

$$f'^j_z(Y_j) := \sum_{b_{j+1},\ldots,b_n \in \{0,1\}} f'_z(r_1, \ldots, r_{j-1}, Y_j, b_{j+1}, \ldots, b_n).$$

Define $h(\mathbf{Y}) := (\widetilde{W}'_\ell \cdot \widetilde{W}'_r)(\mathbf{Y})$, and thus $f'_z(\mathbf{Y}) := (\boldsymbol{\chi}_z \cdot h)(\mathbf{Y})$. Our idea is to additionally maintain bookkeeping tables for the polynomial $h(\mathbf{Y})$, which, in round j, store $3 \cdot 2^{n-j}$ evaluations of $h(\mathbf{Y})$ at points of the form

$$(r_1 \ldots r_{j-1}, \{0, \pm 1\}, b_{j+1} \ldots b_n),$$

for all $(b_{j+1} \ldots b_n) \in \{0,1\}^{n-j}$. A direct observation is that, given the secret sharings of the table for $h(\mathbf{Y})$, parties can locally compute secret sharings of $f'^j_z(Y_j)$, since they already know $\boldsymbol{\chi}_z(\mathbf{Y})$. Thus, the challenge reduces to maintaining distributed bookkeeping tables for $h(\mathbf{Y})$.

The key observation is that the first two columns (respect to $b_{j+1} \in \{0,1\}$) of the table for round $j+1$ are linear combinations of the table for round j.

$$\begin{aligned}
\mathsf{Table}_j: \quad & h(r_1, \ldots, r_{j-1}, \quad \{0, 1, -1\}, \quad b_{j+1}, \quad b_{j+2}, \ldots, b_n), \\
& \qquad\qquad\qquad\qquad\qquad\qquad \Downarrow \textit{ interpolate and evaluate at } r_j \\
\mathsf{Table}_{j+1}: \quad & h(r_1, \ldots, r_{j-1}, \qquad\qquad r_j, \quad b_{j+1}, \quad b_{j+2}, \ldots, b_n),
\end{aligned}$$

This implies that sharings of the first two columns can be locally computed from those of the previous table. As for sharings of the last column (respect to $b_{j+1} = -1$), parties compute its entries from sharings of $\widetilde{W}'_\ell(\mathbf{Y})$ and $\widetilde{W}'_r(\mathbf{Y})$ evaluated at $(r_1, \ldots, r_j, -1, b_{j+2}, \ldots, b_n)$, consuming 2^{n-j-1} standard Beaver triples for round $j+1$. As for the initial table for round 1, the parties already hold sharings of the first two columns (respect to $b_1 \in \{0,1\}$) eliminating the need to recompute them using Beaver triples. It follows from the fact that for all $\mathbf{Y} \in \{0,1\}^n$, $h(\mathbf{Y}) = \widetilde{W}'_\ell \widetilde{W}'_r(\mathbf{Y}) = \widetilde{W}_o(\mathbf{Y})$. In the final round, two additional

points should be evaluated as $h(\mathbf{Y})$ has degree-4 in Y_n. This is precisely why we select vanishing polynomials of the form $aY_n(1-Y_n)$ and $bY_n(1-Y_n)$. The total number of required Beaver triples is $2+\sum_{j=1}^{n} 2^{n-j} = N+1$.

3 Preliminaries

Security Model and Functionalities. In the work, we consider static malicious security with abort, where the adversary $\mathcal{A}$ corrupts t parties at the beginning, and can cheat arbitrarily during the protocol execution. The security is guaranteed in the sense that if $\mathcal{A}$ cheats, then $\mathcal{A}$ will be caught by honest parties with a high probability, and moreover, $\mathcal{A}$ learns nothing about honest parties' input beyond the output. We formalize related standard functionalities in Appendix A.3. Specifically, these include $\mathcal{F}_{\mathsf{MPC}}$, $\mathcal{F}_{\mathsf{Coin}}$ for coin-tossing, $\mathcal{F}_{\mathsf{Commit}}$ for commitment, $\mathcal{F}_{\mathsf{Shamir}}$ for distributing random Shamir secret sharings, and etc.

Multilinear Extension. Multilinear extensions (MLEs) play a crucial role in the study of interactive proofs. We give a formal definition as follows:

Definition 3.1 (Multi-Linear Extension). *Let $f : \{0,1\}^n \to \mathbb{F}$ be a function that maps the n-dimensional binary hypercube to a commutative ring $\mathbb{F}$. The multilinear extension of f is the unique polynomial $\widetilde{f} : \mathbb{F}^n \to \mathbb{F}$ such that $\widetilde{f}(x_1,\ldots,x_n) = f(x_1,\ldots,x_n)$ for all $x_1,\ldots,x_n \in \{0,1\}$, where the degree of $\widetilde{f}$ in each variable is 1. Moreover, $\widetilde{f}$ has the form*

$$\widetilde{f}(x_1,\ldots,x_n) = \sum_{\omega\in\{0,1\}^n} f(\omega)\cdot \boldsymbol{\chi}_\omega(x_1,\ldots,x_n),$$

where, for any $\omega = (\omega_1,\ldots,\omega_n)$,

$$\boldsymbol{\chi}_\omega(x_1,\ldots,x_n) := \prod_{i=1}^{n} \big(x_i\omega_i + (1-x_i)(1-\omega_i)\big).$$

The set $\{\boldsymbol{\chi}_\omega : \mathbb{F}^n \to \mathbb{F}\}_{\omega\in\{0,1\}^n}$ is referred to as the set of multilinear Lagrange basis polynomials with interpolating set $\{0,1\}^n$.

Assume $P_\omega(x) : \{0,1\}^n \to \{0,1\}$ is a function that maps x to 1 if x equals ω and 0 otherwise, and we call it a point predicate. Then $\boldsymbol{\chi}_\omega$ can be viewed as the MLE of $P_\omega(\cdot)$, i.e., $\widetilde{P}_\omega := \boldsymbol{\chi}_\omega$. Formally, given P_ω,

$$\widetilde{P}_\omega(x) := \sum_{\nu\in\{0,1\}^n} P_\omega(\nu)\cdot \boldsymbol{\chi}_\nu(x) = \boldsymbol{\chi}_\omega(x).$$

Note the above computation is similar to executing private information retrieval (PIR) via function secret sharings (FSS) for point functions. The above extension function also holds for point functions i.e., $P_\omega(x) = z$ for some $z \in \mathbb{F}$.

With MLEs of the point predicate, actually we can transform any extension (maybe not multilinear) of a function to a multilinear extension. Given $g : \mathbb{F}^n \to \mathbb{F}$

as an extension of $f : \{0,1\}^n \to \mathbb{F}$, the MLE of f can be computed as

$$\widetilde{f}(x) := \sum_{\omega \in \{0,1\}^n} \widetilde{P}_\omega(x) \cdot g(\omega) = \sum_{\omega \in \{0,1\}^n} \boldsymbol{\chi}_\omega(x) \cdot g(\omega).$$

It is easy to verify that for arbitrary $y \in \{0,1\}$, $\widetilde{f}(y) = g(y) = f(y)$ as g is an extension of f. Moreover, $\widetilde{f}$ is multilinear as $\boldsymbol{\chi}_\omega$ is multilinear and $g(\cdot)$ is independent of x.

W.l.o.g., assume N is a power of two, then a vector $\mathbf{W} := (w_0, \ldots, w_{N-1})$ over $\mathbb{F}$ can be naturally viewed as a function $W : \{0,1\}^{\log N} \to \mathbb{F}$ such that $W(i)$ equals to the i-th entry of $\mathbf{W}$ for all $i \in [0, N)$. Hence, we define the multilinear extension of a vector $\mathbf{W}$ in this way, similarly denoted by $\widetilde{W}$. To evaluate the multilinear extension $\widetilde{W}$ of $\mathbf{W}$ efficiently, we employ the algorithm proposed in [52], which takes $O(N)$ time and $O(N)$ memory usage.

Lemma 3.2 ([52]). *Assume $N = 2^n$ and given $\mathbf{W} \in \mathbb{F}^N$ and $\boldsymbol{r} \in \mathbb{F}^n$, one can compute $\widetilde{W}(\boldsymbol{r})$ in $O(N)$ time and $O(N)$ space.*

Recap: Sumcheck [46]. The celebrated Sumcheck protocol allows to delegate in a verifiable way the task of summing up an n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$ on a binary cube $\{0,1\}^n$. In sketch, the Sumcheck protocol proceeds inductively as follows: in each round i, $\mathcal{P}$ sends to $\mathcal{V}$ a univariate polynomial

$$f_i(X_i) := \sum_{b_{i+1}, \ldots, b_n \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_n),$$

that is the summation of f on a binary cube $\{0,1\}^{n-i}$ with the first $i-1$ variables being fixed to $r_1, \ldots, r_{i-1}$ (received from $\mathcal{V}$ in previous rounds), then $\mathcal{V}$ returns to $\mathcal{P}$ a random $r_i \xleftarrow{\$} \mathbb{F}$. By definition, one can simply verify that $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$. Formal descriptions of the Sumcheck protocol are given in Π_{Sumcheck}. Assuming the maximum degree of f in each variable is d, the Sumcheck protocol has communication complexity $O(dn)$, round complexity n, and soundness error $O(dn/|\mathbb{F}|)$ by the Schwartz-Zippel Lemma. Moreover, the verifier is lightweight, as he only needs to evaluate f at one point rather than 2^n points, and n univariate polynomials of degree at most d.

Protocol 1: Π_{Sumcheck}

Given an n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$. Let $\deg_i(f)$ denote the degree of $f(X_1, \ldots, X_i, \ldots, X_n)$ in variable X_i. The protocol proceeds as follows.

- At the beginning, the prover $\mathcal{P}$ sends to the verifier $\mathcal{V}$ a value H claimed to equal the summation of f on the binary hypercube.
- In the first round, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_1(X_1)$ of degree at most $\deg_1(f)$ claimed to equal

$$\sum_{b_2, \ldots, b_n \in \{0,1\}} f(X_1, b_2, \ldots, b_n).$$

$\mathcal{V}$ checks that $H = f_1(0) + f_1(1)$. $\mathcal{V}$ sends a random $r_1 \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

– In the i_{th} round, where $1 < i < n$, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_i(X_i)$ of degree at most $\deg_i(f)$, claimed to equal

$$\sum_{b_{i+1},\ldots,b_n \in \{0,1\}} f(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_n).$$

$\mathcal{V}$ checks that $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$. $\mathcal{V}$ sends a random $r_i \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.

– In the n_{th} round, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial $f_n(X_n)$ of degree at most $\deg_n(f)$, claimed to equal

$$f(r_1, \ldots, r_{n-1}, X_n).$$

$\mathcal{V}$ checks that $f_{n-1}(r_{n-1}) = f_n(0) + f_n(1)$. Finally, $\mathcal{V}$ selects $r_n \xleftarrow{\$} \mathbb{F}$, and checks that $f(r_1, \ldots, r_n) = f_n(r_n)$. $\mathcal{V}$ will accept if and only if all the above checks pass. Otherwise, $\mathcal{V}$ rejects and aborts.

Lemma 3.3 (Schwartz-Zippel Lemma). *For any nonzero n-variate polynomial $f(x_1, \ldots, x_n)$ over $\mathbb{F}$ with each variable's degree at most d, we have that*

$$\mathsf{Pr}\left[f(\alpha) = 0 \mid \alpha \xleftarrow{\$} \mathbb{F}^n\right] \leq \frac{\sum_{i=1}^{n} \deg_i(f)}{|\mathbb{F}|} \leq \frac{dn}{|\mathbb{F}|}.$$

4 Distributed Sumcheck

In this section, we study the sumcheck problem in the distributed setting, where the polynomial $f(X_1, \ldots, X_n)$ together with H is shared across M parties, and the parties wish to verify whether f correctly sums to H. For illustrating the ideas, we first present a general *distributed sumcheck* protocol $\Pi_{\mathsf{DSumcheck}}$ in Section 4.1 for the trivial case that the coefficients of $f(X_1, \ldots, X_n)$ are all secret-shared. We then introduce a non-trivial Sumcheck problem in the distributive setting, well-motivated by MPC scenarios: verifying multiplication relations, where inputs and outputs are secret-shared.

4.1 Distributed Sumcheck

We assume an authenticated linear secret sharing scheme, denoted by $[\![\cdot]\!]$. This means that if the parties jointly hold shares of a sufficient number of evaluations of $f(X_1, \ldots, X_n)$, or all its coefficients, then they can locally compute shares of any evaluation $f(r_1, \ldots, r_n)$. In both cases, we say f is shared among the M parties, denoted by $[\![f]\!]$. Moreover, f is authenticated as each of its coefficients is authenticated. Following the structure of the classical Sumcheck protocol, we now give an informal description of a distributed sumcheck protocol.

– All parties jointly play the role of a Sumcheck prover. Specifically, for each message (i.e., a polynomial $f_i(X_i)$) that a Sumcheck prover would send in round j, the parties hold a sharing of it and collectively open the message.

– All parties jointly emulate a Sumcheck verifier. Note that a Sumcheck verifier samples a random $r_i \xleftarrow{\$} \mathbb{F}$ and sends it to the Sumcheck prover. This procedure can be emulated by an ideal functionality $\mathcal{F}_{\mathsf{Coin}}$ which tosses random coins for all parties. Additionally, each party also acts as an individual verifier to check the reconstructed proof.

For completeness, assuming each party follows the above protocol honestly, it is readily to see that all parties accept if and only if $H = \sum_{b_1,\ldots,b_n \in \{0,1\}^n} f(b_1, \ldots, b_n)$ by completeness of the Sumcheck protocol. Before discussing security, we first address subtle differences between proof systems and MPC settings. For proof systems, the prover is considered to be malicious and could deviate the protocol arbitrarily whereas the verifier is usually assumed to be honest. Moreover, zero-knowledge is not necessarily required. In contrast, in MPC, the minimal assumption is that at least one party is honest and the remaining parties may collude. However, the privacy of honest parties' inputs should be always preserved.

In fact, the above protocol is not secure, because the proof messages $f_i(X_i)$ may leak information about the secret-shared polynomial $f(\mathbf{X})$. To mitigate this, we mask f with a random sparse polynomial g, which is also shared among the parties. This is essentially a *distributed* analogue of the masking polynomial technique used for zero-knowledge Sumcheck in [56]. As long as g has sufficient entropy, the Sumcheck proof reveals nothing beyond the fact that $H \overset{?}{=} \sum_{b_1,\ldots,b_n \in \{0,1\}^n} f(b_1, \ldots, b_n)$. We now present the distributed Sumcheck protocol $\Pi_{\mathsf{DSumcheck}}$ for the case that all f's coefficients are secret-shared, assuming that $[\![\cdot]\!]$ has t-privacy[4], and at most t parties are corrupted.

Protocol 2: $\Pi_{\mathsf{DSumcheck}}$

Let $[\![\cdot]\!]$ denote an M-party authenticated linear secret sharing scheme over $\mathbb{F}$ with t-privacy. Consider an n-variate polynomial $f : \mathbb{F}^n \to \mathbb{F}$, where the degree of f in variable X_i is denoted by $\deg_i(f)$ for $i \in [1, n]$. We assume access to the ideal functionality $\mathcal{F}_{\mathsf{Coin}}$.

Input: $[\![H]\!]$, $[\![f(X_1, \ldots, X_n)]\!] := \sum_I [\![f_I]\!] \cdot \boldsymbol{X}^I$.[a]

Correlated randomness: $[\![g(X_1, \ldots, X_n)]\!] := [\![g_0]\!] + \sum_{i=1}^{n} \sum_{j=1}^{\deg_i(f)} [\![g_{i,j}]\!] \cdot X_i^j$, where $g_0, g_{i,j}$ are uniformly random in $\mathbb{F}$.

Goal: Verify that $H = \sum_{b_1,\ldots,b_n \in \{0,1\}} f(b_1, \ldots, b_n)$.

1. Parties compute $[\![G]\!] := \sum_{b_1,\ldots,b_n \in \{0,1\}} [\![g(b_1, \ldots, b_n)]\!]$, and open $[\![H + G]\!]$.
2. For each round $i \in [1, n]$,
   (a) Parties $\mathrm{P}_1, \ldots, \mathrm{P}_M$ locally compute

$$[\![f_i(X_i)]\!] := \sum_{b_{i+1},\ldots,b_n \in \{0,1\}^n} [\![(f + g)(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_n)]\!],$$

   whose degree is at most $\deg_i(f)$. Then they open the polynomial $f_i(X_i)$.
   (b) If $i = 1$, all parties check $H + G = f_1(0) + f_1(1)$. Otherwise, they check $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$.

[4] The shared value remains uniformly random conditioned on any $\leq t$ shares.

(c) Parties invoke $\mathcal{F}_{\mathsf{Coin}}$ to sample $r_i \xleftarrow{\$} \mathbb{F}$.

3. Parties $\mathrm{P}_1, \ldots, \mathrm{P}_M$ compute $[\![(f+g)(r_1, \ldots, r_n)]\!]$ and open it. All parties check $f_n(r_n) = (f+g)(r_1, \ldots, r_n)$.
4. All parties accept if and only if all the above checks pass. Otherwise, they reject and abort.

[a] $\boldsymbol{X}^I := X_1^{i_1} \cdots X_n^{i_n}$, where $0 \le i_j \le \deg_j(f)$, for $j \in [1, n]$.

Remark 4.1. Actually, there exists a much simpler and better solution for the case that all f's coefficients are secret-shared. Since $[\![\cdot]\!]$ is both authenticated and linear, one can let each party locally compute $[\![h]\!] := \sum_{b_1, \ldots b_n \in \{0,1\}} [\![f(b_1, \ldots, b_n)]\!]$ and open $[\![h]\!] + [\![0]\!]$, where $[\![0]\!]$ denotes a random sharing of zero. However, such a method is not applicable to the general case fitting MPC scenarios, where $f = f_1 \cdots f_\ell$, and each f_i's coefficients is secret-shared.

We briefly argue the security of $\Pi_{\mathsf{DSumcheck}}$ in this trivial case. The main observation is that essentially the parties jointly emulate an honest prover. Note that the underlying secret sharing scheme $[\![\cdot]\!]$ is both linear and authenticated. Hence, the proof message $f_i(X_i)$ in round i is actually shared in an authenticated way, meaning that honest parties can detect whether these proof messages are opened correctly. In addition, the verifier is honestly emulated by an ideal functionality $\mathcal{F}_{\mathsf{Coin}}$, leaving no room for adversarial manipulation. It is natural to assume the existence of $\mathcal{F}_{\mathsf{Coin}}$, since there is at least one honest party and $\mathcal{F}_{\mathsf{Coin}}$ can be instantiated efficiently. Similarly, the correlated randomness $[\![g(\mathbf{X})]\!]$ can also be assumed to be generated by an ideal functionality. Informally, the soundness of $\Pi_{\mathsf{DSumcheck}}$ reduces to the soundness of interactive proofs where both the prover and the verifier are honest, which trivially holds.

Interestingly, the soundness of $\Pi_{\mathsf{DSumcheck}}$ does not rely on the soundness of the Sumcheck protocol, which suggests that full authentication may not be strictly necessary for achieving malicious security. Specifically, it may not be essential to prevent the emulated prover from cheating, since the Sumcheck protocol itself is secure against a malicious prover. To explore this, we examine the notion of "minimal authentication". At first glance, authentication is necessary for the inputs, i.e., the statement to be checked. For instance, the adversary $\mathcal{A}$ should be prevented from changing $[\![H]\!]$ arbitrarily, otherwise $\mathcal{A}$ can falsely "correct" and prove incorrect sumcheck relations $H \neq \sum f(\mathbf{X})$. However, since all messages in $\Pi_{\mathsf{DSumcheck}}$ are essentially linear combinations of the inputs (and the public randomness), then they are inherently authenticated as long as the inputs are authenticated. We will see soon for many MPC scenarios it is not often the case that all coefficients of $f(X_1, \ldots, X_n)$ are explicitly shared among the parties. Instead, the function f is more likely of the form:

$$f(X_1, \ldots, X_n) = \prod_{i \in [1,\ell]} f_i(X_1, \ldots, X_n)$$

and coefficients of $f_1, \ldots, f_\ell$ are shared. Such cases are much more challenging, in particular, if we do not further assume a multiplicative secret sharing scheme. On the other hand, it in turn leaves us room to relax the full authenticated property. Loosely speaking, the polynomial $f = f_1 \cdots f_\ell$ is not necessary to be shared with authentication, since authenticated shares of $f_1, \ldots, f_\ell$ fully determine f and $\mathcal{A}$ can not modify the statement. For instance, the well-known Shamir secret sharing is robust (hence authenticated) against $t < M/2$ corruptions, denoted by $[\cdot]_t$, and $([a]_t, [b]_t)$ determines $c = ab$. The parties could obtain a non-authenticated share of $c = ab$ through local multiplications, denoted by $[c]_{2t}$.

4.2 Sumcheck for Multiplication Verification in MPC

For semi-honest MPC protocols based on robust linear secret sharing schemes that are secure up to additive attacks, it is well-known that to achieve malicious security it suffices to verify multiplications [30,7,12]. In this section, we aim at characterizing such multiplication verification problems using sumcheck relations. Concretely, given $[\![\mathbf{W}_o]\!], [\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$ corresponding to the outputs and inputs of multiplication gates in a circuit, the goal is to verify $\mathbf{W}_o \stackrel{?}{=} \mathbf{W}_\ell * \mathbf{W}_r$, where $*$ denotes the entry-wise multiplication.

Notations. Let $n \in \mathbb{N}$, $N = 2^n$ and define $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$. View $\mathbf{W}_o$ as a function $W_o : \{0,1\}^n \to \mathbb{F}$, such that $W_o(i)$ equals to the i-th entry of $\mathbf{W}_o$, for $i \in \{0,1\}^n = [0, N-1]$, and define W_ℓ, W_r similarly. Denote multilinear extensions(MLEs) of W_o, W_ℓ, W_r by $\widetilde{W}_o, \widetilde{W}_\ell, \widetilde{W}_r$, respectively.

We show two sumcheck relations induced by $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$. The first follows the idea of GKR [33] by defining a "wiring predicate" $\mathsf{mult}(z, x, y) : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, where $\mathsf{mult}(z, x, y)$ evaluates to 1 if and only if inputs (z, x, y) correspond to the three wire labels of a multiplication gate of the circuit. In this way, we have $W_o(z) = \sum_{x,y \in \{0,1\}^n} \mathsf{mult}(z, x, y) \cdot W_\ell(x) \cdot W_r(y)$. By the uniqueness of multilinear extensions, the above implies that

$$\widetilde{W}_o(z) = \sum_{\mathbf{X}, \mathbf{Y} \in \{0,1\}^n} \widetilde{\mathsf{mult}}(z, \mathbf{X}, \mathbf{Y}) \cdot \widetilde{W}_\ell(\mathbf{X}) \cdot \widetilde{W}_r(\mathbf{Y}).$$

The above equation holds for any $z \in \mathbb{F}^n$. Note that the Schwartz-Zippel Lemma guarantees that for any $W_o^* \neq W_o$, $z \xleftarrow{\$} \mathbb{F}^n$, $\widetilde{W}_o^*(z)$ equals to $\widetilde{W}_o(z)$ with probability $\leq n/|\mathbb{F}|$. Hence, verifying a single random evaluation of $\widetilde{W}_o$ suffices to establish correctness, leading to a sumcheck relation. Applying the Sumcheck protocol to this relation incurs $2n$ rounds, $O(n)$ communication and $O(N)$ prover computation by using the optimization technique of [56]. However, such a sumcheck relation contains redundancy. Intuitively, the wiring predicate was originally introduced for a general circuit consisting of addition gates and multiplication gates. In our setting, where only multiplication gates are involved, an output wire label alone is sufficient to identify the gate, leading to further optimizations.

As indicated in Section 2, inspired by the approach of [53], we introduce the point predicate $P_z(y) : \{0,1\}^n \to \{0,1\}$, which evaluates to 1 if and only if $y = z$

and otherwise. We obtain the following relations as in Eq.(1):

$$\widetilde{W}_o(z) = \sum_{\mathbf{Y}\in\{0,1\}^n} \underbrace{\boldsymbol{\chi}_{\mathbf{Y}}(z)\cdot\widetilde{W}_\ell(\mathbf{Y})\cdot\widetilde{W}_r(\mathbf{Y})}_{f_z(\mathbf{Y})},$$

which holds for any $z \in \mathbb{F}^n$. Applying the Sumcheck protocol on the above relation would be more efficient, since there are only n variables to sum-up. However, the parties do not hold explicit secret shares of the polynomial $f_z(\mathbf{Y})$, but only secret shares of the coefficients of $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$. Hence, $\Pi_{\mathsf{DSumcheck}}$ does not directly apply, and we need to develop techniques that enable efficient sumcheck verification while working with shared polynomial coefficients of $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$. We distinguish two typical settings and give the respective solutions:

- Honest majority MPC based on Shamir secret sharing, where we enjoy the multiplicative property of Shamir secret sharing. Details refer to Section 5.
- Dishonest majority MPC based on SPDZ-style authenticated secret sharing, where instead we have to use multiplication triples. Details refer to Section 6.

5 MPC with an Honest Majority

In this section, we tackle the distributed Sumcheck problem induced by Eq.(1) in the honest majority MPC setting, where we additionally assume *multiplicative* secret sharing schemes. Without lose of generality, we take Shamir secret sharing (refer to Appendix A.1) as a concrete and representative instantiation, denoted by $[\cdot]_t$. Formally, we assume M-party, $t < M/2$ corruptions, and a sufficiently large field $\mathbb{F}$, so that $[\cdot]_t$ is robust (hence authenticated), linear, and multiplicative.

Formally, the goal is to verify that secret-shared values $[\mathbf{W}_o]_t, [\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t$ satisfy $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$. At a high-level, we first translate it into a distributed Sumcheck problem, then it suffices to employ a Sumcheck protocol in the distributed way. Related notations are defined as follows.

Notations. Assume $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$ with $N = 2^n$ for some $n \in \mathbb{N}$. Denote their multilinear extensions (MLEs) by $\widetilde{W}_o(\mathbf{Y}), \widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$, respectively. Let $g(\mathbf{Y}) := g_0 + \sum_{i=1}^n \sum_{j=1}^3 g_{i,j} Y_i^j$ be a random *sparse* masking polynomial with all coefficients shared among the parties. Define $G := \sum_{\mathbf{Y}\in\{0,1\}^n} g(\mathbf{Y})$. According to Eq.(1), we define $f_z(\mathbf{Y}) := \boldsymbol{\chi}_z(\mathbf{Y}) \cdot \widetilde{W}_\ell(\mathbf{Y}) \cdot \widetilde{W}_r(\mathbf{Y})$. For simplicity, for $r_1 \dots r_{j-1}, v \in \mathbb{F}$ and $b_{j+1}, \dots, b_n \in \{0,1\}$ we define

$$V_v^j := (r_1, \dots, r_{j-1}, v, b_{j+1}, \dots, b_n), \quad \widehat{Y}_j := (r_1, \dots, r_{j-1}, Y_j, b_{j+1}, \dots, b_n).$$

For each $\omega := (b_{j+1}, \dots, b_n) \in \{0,1\}^{n-j}$, we abuse

$$f_{z,\omega}^j(Y_j) := f_z(\widehat{Y}_j) = f_z(r_1, \dots, r_{j-1}, Y_j, b_{j+1}, \dots, b_n),$$

and one can see that $f^j_{z,\omega}(v) = f_z(V^j_v)$. We also denote

$$f^j_z(Y_j) := \sum_{\omega\in\{0,1\}^{n-j}} \left(f_z(\widehat{Y}_j) + g(\widehat{Y}_j)\right) = \sum_{\omega\in\{0,1\}^{n-j}} \left(f^j_{z,\omega}(Y_j) + g(\widehat{Y}_j)\right).$$

Given above notations, verifying $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$ reduces to sampling $z \xleftarrow{\$} \mathbb{F}$ and then applying the Sumcheck protocol to the relation

$$\widetilde{W}_o(z) + G = \sum_{\mathbf{Y}\in\{0,1\}^n} (f_z + g)(\mathbf{Y}). \tag{3}$$

In a bare-bone sketch, in each round j, the Sumcheck verifier would receive a polynomial $f^j_z(Y_j)$, and verify that $f^{j-1}_z(r_{j-1}) = f^j_z(0) + f^j_z(1)$. In the final round, the Sumcheck verifier additionally checks $f^n_z(r_n) = (f_z + g)(r_1, \ldots, r_n)$.

Distributed Sumcheck. The goal is to jointly emulate a Sumcheck prover that runs the Sumcheck protocol on Eq.(3). Namely, the parties need to compute the (virtual) prover's message in each round.

In the MPC setting, inputs $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r$ and the random masking polynomial $g(\mathbf{Y})$ are shared among the parties. Note that the secret sharing scheme $[\cdot]_t$ is robust, linear, and multiplicative. A direct observation is that $\widetilde{W}_o(z) + G$ is authentically shared among parties. Moreover, for any $y \in \mathbb{F}^n$ parties can locally compute $[(f_z + g)(y)]_{2t} := \left[\widetilde{W}_\ell(y)\right]_t \cdot \left[\widetilde{W}_r(y)\right]_t \cdot \boldsymbol{\chi}_z(y) + [g(y)]_t$. Hence, for each $j \in [1, n]$, they can locally compute and reconstruct

$$\left[f^j_z(Y_j)\right]_{2t} := \sum_{\omega\in\{0,1\}^{n-j}} \left[f^j_{z,\omega}(Y_j)\right]_{2t} + \left[g(\widehat{Y}_j)\right]_t.$$

Remark 5.1. Note that $[\cdot]_{2t}$ is not authenticated, meaning that the adversary $\mathcal{A}$ can open $f^j_z(Y_j)$ to an arbitrary polynomial $f^{*j}_z(Y_j)$. We emphasis that this adversarial behavior does not compromise security because the round-by-round soundness of the Sumcheck protocol prevents any meaningful cheating. This is a critical difference compared to FLIOP-based approaches [7,36], allowing us to relax the requirement of robustness to authentication. Essentially, all parties are involved in emulating a (malicious) Sumcheck prover in the interactive proofs setting.

Last-round check. Emulating the final equation check $f^n_z(r_n) = (f_z + g)(r_1, \ldots, r_n)$ is more complicated, as in the non-distributed setting typically a polynomial commitment scheme would be required. The issue arises because $f^n_z(Y_n)$ is opened before r_n being sampled while $(f_z + g)(r_1, \ldots, r_n)$ is not. If parties simply reconstruct $[(f_z + g)(r_1, \ldots, r_n)]_{2t}$, $\mathcal{A}$ can always dishonestly pass the final check by opening it to the target $f^n_z(r_n)$. Basically,

$$(f_z + g)(r_1, \ldots, r_n) = (\widetilde{W}_\ell \widetilde{W}_r \boldsymbol{\chi}_z + g)(r_1, \ldots, r_n)$$

should be computed and reconstructed in an authenticated way. To this end, a naive approach would be to reconstruct Shamir sharings of $\left[\widetilde{W}_\ell(r_1,\ldots,r_n)\right]_t$, $\left[\widetilde{W}_r(r_1,\ldots,r_n)\right]_t$ and $[g(r_1,\ldots,r_n)]_t$. However, this leaks information about the inputs. We provide two solutions to the above issue. The first is to compute $\left[\widetilde{W}_\ell\widetilde{W}_r(r_1,\ldots,r_n)\right]_t$ with the help of one random Beaver triple $([a]_t,[b]_t,[c]_t)$, and it suffices to open $\left[\widetilde{W}_\ell(r_1,\ldots,r_n)-a\right]_t$, and $\left[\widetilde{W}_r(r_1,\ldots,r_n)-b\right]_t$. Using local computation, parties can then obtain $[(f_z+g)(r_1,\ldots,r_n)]_t$ and open it in an authenticated way. This approach is technically simple and only introduces a minor cost for preparing one triple.

Mask by vanishing polynomials. Inspired by the zero-knowledge GKR technique from [56], we show an alternative method that avoids using Beaver triples while requires only 4 additional random Shamir shares. At a high level, the idea is to mask $\widetilde{W}_\ell(\mathbf{Y}),\widetilde{W}_r(\mathbf{Y})$ with (low-degree) vanishing polynomials, such that it is allowed to open evaluations meanwhile the sumcheck relation still holds. Specifically, we choose the vanishing polynomials to be $aY_n(1-Y_n),bY_n(1-Y_n)$, where $a,b\xleftarrow{\$}\mathbb{F}$ and parties hold $[a]_t,[b]_t$. The underlying sumcheck relation is reformulated as follows:

$$\widetilde{W}_o(z)+G=\sum_{\mathbf{Y}\in\{0,1\}^n}\boldsymbol{\chi}_z(\mathbf{Y})\cdot\underbrace{\left(\widetilde{W}_\ell(\mathbf{Y})+aY_n(1-Y_n)\right)}_{\widetilde{W}'_\ell(\mathbf{Y})}\cdot\underbrace{\left(\widetilde{W}_r(\mathbf{Y})+bY_n(1-Y_n)\right)}_{\widetilde{W}'_r(\mathbf{Y})}+g(\mathbf{Y}). \tag{4}$$

The main observation is that for $j\in[1,n-1]$, it holds that

$$\begin{aligned}f_z^j(Y_j)&=\sum_{\omega\in\{0,1\}^{n-j}}\left(f_{z,\omega}^j(Y_j)+g(\widehat{Y}_j)\right)=\sum_{b_{j+1},\ldots,b_n\in\{0,1\}}(\widetilde{W}_\ell\widetilde{W}_r\boldsymbol{\chi}_z+g)(\widehat{Y}_j)\\&=\sum_{b_{j+1},\ldots,b_n\in\{0,1\}}(\widetilde{W}'_\ell\widetilde{W}'_r\boldsymbol{\chi}_z+g)(\widehat{Y}_j).\end{aligned}$$

Thus, for rounds $j\in[1,n-1]$, parties can still reconstruct $f_z^j(Y_j)$ as before. For round n, instead of opening $f_z^n(Y_n)$, parties reconstruct a degree-5 polynomial

$$f_z'^n(Y_n):=\boldsymbol{\chi}_z(\widehat{Y}_n)\cdot\left(\widetilde{W}_\ell(\widehat{Y}_n)+aY_n(1-Y_n)\right)\cdot\left(\widetilde{W}_\ell(\widehat{Y}_n)+bY_n(1-Y_n)\right)+g(\widehat{Y}_n).$$

Since $f_z'^n(Y_n)$ has degree 5, for privacy, the random masking polynomial $g(\mathbf{Y})$ should be of degree-5 on variable Y_n as well. It holds that $f_z^{n-1}(r_{n-1})=f_z'^n(0)+f_z'^n(1)$ and in the end they check $(f_z'^n+g)(r_n)=(\widetilde{W}'_\ell\widetilde{W}'_r\boldsymbol{\chi}_z+g)(r_1,\ldots,r_n)$ by opening Shamir shares $\left[\widetilde{W}'_\ell(r_1,\ldots,r_n)\right]_t:=\left[\widetilde{W}_\ell(r_1,\ldots,r_n)\right]_t+r_n(1-r_n)\cdot[a]_t$,

$$\left[\widetilde{W}'_r(r_1,\ldots,r_n)\right]_t:=\left[\widetilde{W}_r(r_1,\ldots,r_n)\right]_t+r_n(1-r_n)\cdot[b]_t,\text{ and }[g(r_1,\ldots,r_n)]_t.$$

By the randomness of $[a]_t,[b]_t,[g(\mathbf{Y})]_t$, this approach ensures that no additional information is leaked beyond verifying whether $\mathbf{W}_o=\mathbf{W}_\ell*\mathbf{W}_r$.

In summary, the above specifications admit a distributed sumcheck protocol for verifying $N = 2^n$ multiplications with $O(n)$ rounds, $O(n)$ communication, consuming $O(n)$ coins and $O(n)$ random Shamir shares. Specifically, the communication costs consist of, reconstructions of $4n+6$ elements, and $n+1$ invocations of $\mathcal{F}_{\mathsf{Coin}}$. The required randomness, beyond $\mathcal{F}_{\mathsf{Coin}}$, is $3n+5$ random Shamir shares.

Optimizing Computational Complexity. We also place special emphasis on computational efficiency. By introducing the bookkeeping table technique from [50] to the distributed setting, we achieve $O(N + M \log N)$ computation and $O(N)$ memory for each party. At a high level, all parties maintain three global bookkeeping tables for $\widetilde{W}_\ell, \widetilde{W}_r, \boldsymbol{\chi}_z$ during the protocol, where the first two tables are shared among parties via $[\cdot]_t$. We briefly outline the algorithms.

For each round j, the global table $\mathbf{A}_\ell^j$ (for $\widetilde{W}_\ell(\mathbf{Y})$) stores 2^{n-j+1} values

$$\widetilde{W}_\ell\left(r_1, \ldots, r_{j-1}, b_j, b_{j+1}, \ldots, b_n\right),$$

for all $b_j, b_{j+1} \ldots b_n \in \{0,1\}$. Given a fixed $\omega := (b_{j+1} \ldots b_n) \in \{0,1\}^{n-j}$, one can interpolate a degree-1 polynomial $\widetilde{W}_{\ell,\omega}^j(Y_j)$ such that $\widetilde{W}_{\ell,\omega}^j(v) = \mathbf{A}_\ell^j(v\|\omega)$, for $v \in \{0,1\}$. Since $\widetilde{W}_{\ell,\omega}^j(Y_j)$ corresponds to $\widetilde{W}_\ell(\widehat{Y}_j)$, where $\widehat{Y}_j := (r_1 \ldots r_{j-1}, Y_j, b_{j+1} \ldots b_n)$, evaluating $\widetilde{W}_{\ell,\omega}^j(Y_j)$ at r_j yields $\widetilde{W}_\ell(r_1 \ldots r_{j-1}, r_j, b_{j+1} \ldots b_n)$, i.e., the ω-entry of table $\mathbf{A}_\ell^{j+1}$. Formally, for each $\omega = (b_{j+1} \ldots b_n) \in \{0,1\}^{n-j}$, the table $\mathbf{A}_\ell^{j+1}$ updates as

$$\mathbf{A}_\ell^{j+1}(\omega) := (1 - r_j)\mathbf{A}_\ell^j(0\|\omega) + r_j \mathbf{A}_\ell^j(1\|\omega).$$

Hence, for each $j \in [1, n-1]$, $\left[\mathbf{A}_\ell^{j+1}\right]_t$ can be locally computed from $\left(\left[\mathbf{A}_\ell^j\right]_t, r_j\right)$ using $O(2^{n-j})$ operations in $\mathbb{F}$. Since $\mathbf{A}_\ell^1$ stores all N evaluations of $\widetilde{W}_\ell(\mathbf{Y})$ (i.e., $\mathbf{W}_\ell$), parties initialize $\left[\mathbf{A}_\ell^1\right]_t$ and iteratively compute subsequent tables by $\sum_{j=1}^n O(2^{n-j}) = O(N)$ operations in $\mathbb{F}$.

The same procedure applies to the tables $\left[\mathbf{A}_r^j\right]_t, \mathbf{A}_{\boldsymbol{\chi}}^j$ for $\widetilde{W}_r(\mathbf{Y}), \boldsymbol{\chi}_z(\mathbf{Y})$, respectively. There is a minor difference in obtaining $\mathbf{A}_{\boldsymbol{\chi}}^1$, which requires additional computations. Since $\boldsymbol{\chi}_z(\mathbf{Y}) = \prod_{i=1}^n (z_i Y_i + (1 - z_i)(1 - Y_i))$ is highly structured, the table $\mathbf{A}_{\boldsymbol{\chi}}^1$ can be easily prepared in $O(N)$ time. In particular, Appendix E.1 presents an efficient algorithm with only $N + O(\sqrt{N})$ multiplications.

Given the bookkeeping tables $\left[\mathbf{A}_\ell^j\right]_t$, $\left[\mathbf{A}_r^j\right]_t$ and $\mathbf{A}_{\boldsymbol{\chi}}^j$ for round j, the parties interpolate linear polynomials $\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]_t$, $\left[\widetilde{W}_r(\widehat{Y}_j)\right]_t$ and $\boldsymbol{\chi}_z(\widehat{Y}_j)$, and compute two additional evaluations at points $v \in \{-1, 2\}$, for each $b_{j+1}, \ldots, b_n \in \{0,1\}$. These evaluations determine $\left[f_{z,\omega}^j(Y_j)\right]_{2t} := \left[f_z(\widehat{Y}_j)\right]_{2t}$ and further

$$\left[f_z^j(Y_j)\right]_{2t} := \sum_{\omega \in \{0,1\}^{n-j}} \left(\left[f_{z,\omega}^j(Y_j)\right]_{2t} + \left[g(\widehat{Y}_j)\right]_t\right).$$

Since $[g(\mathbf{Y})]_t$ is sparse with only $3n+3$ terms, computing evaluations of $\left[g(\widehat{Y}_j)\right]_t$ is much cheaper.

We give a self-contained distributed Sumcheck protocol Π_{Vrfy} for verifying multiplications with a detailed computational cost analysis in Appendix E.1.

Protocol 3: Π_{Vrfy}

Let $n \in \mathbb{N}$ and $N = 2^n$. Let $[\cdot]_t$ denote an M-party Shamir secret sharing scheme over $\mathbb{F}$. Assume at most $t < M/2$ parties are corrupted and access to functionalities $\mathcal{F}_{\mathsf{Coin}}$ and $\mathcal{F}_{\mathsf{Shamir}}$.
Input: $[\mathbf{W}_o]_t, [\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t$, where $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$.
Randomness: $[a]_t, [b]_t$ and $[g(\mathbf{Y})]_t := [g_0]_t + [g_{n,4}]_t \cdot Y_n^4 + [g_{n,5}]_t \cdot Y_n^5 + \sum_{i=1}^{n} \sum_{j=1}^{3} [g_{i,j}]_t \cdot Y_i^j$, where $a, b, g_0, g_{i,j}$ are uniformly random in $\mathbb{F}$.
Goal: Verify that $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$.

1. Parties $\mathrm{P}_1, \ldots, \mathrm{P}_M$ invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $z \xleftarrow{\$} \mathbb{F}$.
2. Each party computes $\left[\widetilde{W}_o(z)\right]_t := \sum_{\omega \in \{0,1\}^n} [W_o(\omega)]_t \cdot \boldsymbol{\chi}_\omega(z)$ and $[G]_t := \sum_{b_1 \ldots b_n \in \{0,1\}} [g(b_1, \ldots, b_n)]_t$. Then they securely open $\left[\widetilde{W}_o(z) + G\right]_t$.
3. For round $j \in [1, n-1]$:
   (a) Compute the bookkeeping tables $\left[\mathbf{A}_\ell^j\right]_t, \left[\mathbf{A}_r^j\right]_t, \mathbf{A}_{\boldsymbol{\chi}}^j$.
   (b) For each $\omega := (b_{j+1}, \ldots, b_n) \in \{0,1\}^{n-j}$, Compute $\left[f_{z,\omega}^j(Y_j)\right]_{2t}$
      i. Obtain $\left[\mathbf{A}_\ell^j(0\|\omega)\right]_t, \left[\mathbf{A}_\ell^j(1\|\omega)\right]_t$ from table $\left[\mathbf{A}_\ell^j\right]_t$ and interpolate the polynomial $\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]_t$. (Obtain $\left[\widetilde{W}_r(\widehat{Y}_j)\right]_t, \boldsymbol{\chi}_z(\widehat{Y}_j)$ similarly.)
      ii. Evaluate $\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]_t$ at $Y_j \in \{-1, 2\}$ to obtain $\left[\widetilde{W}_\ell(V_{-1}^j)\right]_t$, $\left[\widetilde{W}_\ell(V_2^j)\right]_t$. (Also evaluate $\left[\widetilde{W}_r(\widehat{Y}_j)\right]_t, \boldsymbol{\chi}_z(\widehat{Y}_j)$ at $Y_j \in \{-1, 2\}$.)
      iii. Interpolate a degree-3 polynomial $\left[f_{z,\omega}^j(Y_j)\right]_{2t}$, such that $\left[f_{z,\omega}^j(v)\right]_{2t} := \left[\widetilde{W}_\ell(V_v^j)\right]_t \left[\widetilde{W}_r(V_v^j)\right]_t \cdot \boldsymbol{\chi}_z(V_v^j)$, for $v \in \{0, \pm 1, 2\}$.
   (c) Let $\left[f_z^j(Y_j)\right]_{2t} := \sum_{\omega \in \{0,1\}^{n-j}} \left(\left[f_{z,\omega}^j(Y_j)\right]_{2t} + \left[g(\widehat{Y}_j)\right]_t \right)$.
   (d) Open $f_z^j(Y_j)$ of degree 3. If $j = 1$, all parties verify $f_z^1(0) + f_z^1(1) = \widetilde{W}_0(z) + G$, otherwise $f_z^j(0) + f_z^j(1) = f_z^{j-1}(r_{j-1})$.
   (e) Invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $r_j \xleftarrow{\$} \mathbb{F}$.
4. For round $j = n$:
   (a) Compute the bookkeeping tables $[\mathbf{A}_\ell^n]_t, [\mathbf{A}_r^n]_t, \mathbf{A}_{\boldsymbol{\chi}}^n$.
   (b) Obtain $[\mathbf{A}_\ell^n(0)]_t, [\mathbf{A}_\ell^n(1)]_t$ from the table $[\mathbf{A}_\ell^n]_t$ and interpolate the polynomial $\left[\widetilde{W}_\ell(\widehat{Y}_n)\right]_t$. (Obtain $\left[\widetilde{W}_r(\widehat{Y}_n)\right]_t, \boldsymbol{\chi}_z(\widehat{Y}_n)$ similarly.)
   (c) Evaluate $\left[\widetilde{W}_\ell(\widehat{Y}_n)\right]_t$ at $Y_j \in \{-1, \pm 2, 3\}$, and obtain $\left[\widetilde{W}_\ell(V_v^n)\right]_t$, for $v \in \{-1, \pm 2, 3\}$. (Also evaluate $\left[\widetilde{W}_r(\widehat{Y}_n)\right]_t, \boldsymbol{\chi}_z(\widehat{Y}_n)$ at $\{-1, \pm 2, 3\}$.)
   (d) Interpolate a degree 5 polynomial $[f_z'^n(Y_n)]_{2t}$ such that $[f_z'^n(v)]_{2t} := \left(\left[\widetilde{W}_\ell(V_v^n)\right]_t + [a]_t\, v(1-v) \right) \left(\left[\widetilde{W}_r(V_v^n)\right]_t + [b]_t\, v(1-v) \right) \cdot \boldsymbol{\chi}_z(V_v^n) + g(V_v^n)$ for each $v \in \{0, \pm 1, \pm 2, 3\}$.
   (e) Open $f_z'^n(Y_n)$. All parties check $f_z'^n(0) + f_z'^n(1) = f_z^{n-1}(r_{n-1})$.
   (f) Invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $r_n \xleftarrow{\$} \mathbb{F}$. Additionally, define $\mathbf{r} := (r_1 \ldots r_n)$.

5. Parties compute and open $\left[\widetilde{W}'_\ell(\mathbf{r})\right]_t := \left[\widetilde{W}_\ell(\mathbf{r})\right]_t + r_n(1-r_n)\cdot[a]_t$, $\left[\widetilde{W}_r(\mathbf{r})\right]_t := \left[\widetilde{W}_r(\mathbf{r})\right]_t + r_n(1-r_n)\cdot[b]_t$, $[g(\mathbf{r})]_t$.
6. All parties verify $f'^n_z(r_n) = (\boldsymbol{\chi}_z \cdot \widetilde{W}'_\ell \cdot \widetilde{W}'_r + g)(\mathbf{r})$.
7. All parties accept if and only if all above checks pass. Otherwise, reject.

Theorem 5.2. *Π_{Vrfy} securely realizes the functionality $\mathcal{F}_{\mathsf{VrfyMult}}$ in the $(\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Shamir}})$-hybrid model against a static malicious adversary who corrupts at most t parties, where $t < M/2$. Specifically, Π_{Vrfy} has the following properties:*

- *Computational complexity $O(N + M\log N)$.*
- *Communication complexity $O(\log N)$. Concretely, parties reconstruct $4\log N + 6$ secret shared values, and consume $3\log N + 5$ random Shamir shares from $\mathcal{F}_{\mathsf{Shamir}}$, $\log N + 1$ random coins from $\mathcal{F}_{\mathsf{Coin}}$.*
- *Round complexity $O(\log N)$.*
- *The adversary $\mathcal{A}$ has an advantage of at most $\frac{4\log N+2}{|\mathbb{F}|}$.*

Proof. The detailed security proof is given in Appendix D.1. Here, we briefly discuss the computational complexity of Π_{Vrfy}, which mainly consists of three parts, emulations of the Sumcheck prover, reconstructions of the proof messages, and verification of the proof. Specifically in each round j, computing the secret-shared proof message $\left[f^j_z(Y_j)\right]_{2t}$ takes $O(2^{n-j})$ computations, opening Shamir secrets $f^j_z(Y_j)$ takes $O(M)$ computations[5], and verifying $f^{j-1}_z(r_{j-1}) = f^j_z(0) + f^j_z(1)$ takes $O(1)$ computations. Hence, the total computational cost per party is $O(N + M\log N)$ with an overall $O(\log N) = O(n)$ rounds. □

Remark 5.3. We then apply Π_{Vrfy} to the well-known semi-honest protocol by Damgård and Nielsen [22] with the optimizations from [36,37], which requires communication complexity of $5.5N$ field elements per multiplication gate per party, $O(NM)$ field operations for a designated king party, and $O(N)$ field operations for each remaining party. To integrate Π_{Vrfy} efficiently, we can also employ the king party approach, and let the king party to recover the proof message $f^j_z(Y_j)$[6]. The resulting malicious protocol retains the same efficiency in terms of both computation and communication. Specifically, the additional communication is $O(\log N)$ field elements per party, while for computation, the king party incurs only an additional $O(N + M\log N)$ operations, and every remaining party incurs an additional $O(N)$ operations.

6 MPC with a Dishonest Majority

In this section, we further tackle the dishonest majority setting, where we no longer assume multiplicative secret sharing schemes. By developing the distributed

[5] We assume the Lagrange coefficients to recover the secrets have been pre-computed.
[6] The king party has not to be fixed in different rounds. Moreover, Π_{Vrfy} can also be applied to the ATLAS protocol [35].

sumcheck techniques, we consequently obtain a concretely efficient MPC protocol with sublinear preprocessing, and online communication of $5 + o(1)$ field elements per multiplication gate per party. W.l.o.g, we take SPDZ-like [23] authenticated secret sharing (see Appendix A.2) as a concrete and representative instantiation.

Authenticated Beaver triples serve as an essential building block for designing modern MPC protocols, particularly in settings against a malicious majority. An *authenticated Beaver triple* over $\mathbb{F}$ is a tuple of random authenticated shares $(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket)$, satisfying $x, y \xleftarrow{\$} \mathbb{F}$ and $z = x \cdot y$. Given a sufficient amount of authenticated triples, the MPC online protocol can be executed efficiently, with communication of $2 + o(1)$ field elements per multiplication gate per party [23]. However, generating authenticated Beaver triples often constitutes a major bottleneck in overall complexity. Unlike other types of correlations, e.g. standard Beaver triples and authenticated shares that can be generated in sublinear communication by using practically efficient PCGs [8], to our best knowledge, existing efficient PCGs for authenticated Beaver triples [9,6,42] are currently restricted to the two-party case[7].

6.1 Checking Unverified Authenticated Beaver Triples

Before introducing our dishonest majority MPC protocols, we first study a fundamental problem: verification of the unverified authenticated triple relations. Basically, for SPDZ-like protocols, correctness of the random authenticated triples should be ensured before the online phase. To securely generate N valid authenticated triples, a typical method is to first generate $2N$ unverified authenticated triples via a semi-honest protocol, and then *sacrifice* half of them for verification, which incurs at least $2\times$ overhead. However, such $2\times$ overhead is not so satisfying, since the underlying semi-honest protocols are usually expensive.

We propose an alternative method for validating unverified authenticated Beaver triples by introducing the powerful sumcheck idea into the context. Concretely, for verifying N unverified authenticated Beaver triples, our sumcheck-based approach only requires additional correlated randomness of $N + 1$ *standard Beaver triples*, and $3 \log N + 5$ random authenticated sharings, which are significantly cheaper to generate than N authenticated Beaver triples.

Our approach follows the same high-level idea of Π_{Vrfy} from Section 5, where we first translate the verification task into a distributed sumcheck problem and employ a distributed Sumcheck protocol. The major difference and challenge is that the underlying secret sharing scheme is no longer multiplicative.

For convenience and readability, we restate the notations as follows.

Notations. Suppose $N = 2^n$ for some $n \in \mathbb{N}$. Let $[\cdot]$ denote an additive secret sharing scheme and $\llbracket \cdot \rrbracket$ denote the SPDZ authenticated secret sharing

[7] Although there exists a PCG [1] that generates authenticated triples with a seed size of $O(\sqrt{N})$, the construction is not yet practical because of the large hidden constant cubic to both the number of parties and the security parameter.

scheme. Let $[\![\mathbf{W}_o]\!], [\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$ be the inputs, where parties want to check $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$. Denote $\widetilde{W}_o, \widetilde{W}_\ell, \widetilde{W}_r : \mathbb{F}^n \to \mathbb{F}$ as the multilinear extensions (MLEs) of $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$, respectively. Let g be a random sparse polynomial of the form $g(\mathbf{Y}) := g_0 + g_{n,4}Y_n^4 + g_{n,5}Y_n^5 + \sum_{i=1}^n (g_{i,1}Y_i + g_{i,2}Y_i^2 + g_{i,3}Y_i^3)$, and $z \xleftarrow{\$} \mathbb{F}^n$.

Define polynomials $h(\mathbf{Y}) = \widetilde{W}_\ell(\mathbf{Y}) \cdot \widetilde{W}_r(\mathbf{Y})$, $h'(\mathbf{Y}) = \widetilde{W}'_\ell(\mathbf{Y}) \cdot \widetilde{W}'_r(\mathbf{Y})$, $f_z(\mathbf{Y}) = \boldsymbol{\chi}_z \widetilde{W}_\ell \widetilde{W}_r(\mathbf{Y})$ and $f'_z(\mathbf{Y}) = \boldsymbol{\chi}_z \widetilde{W}'_\ell \widetilde{W}'_r(\mathbf{Y})$, where $\widetilde{W}'_\ell(\mathbf{Y}) = \widetilde{W}_\ell(\mathbf{Y}) + aY_n(1-Y_n)$, $\widetilde{W}'_r(\mathbf{Y}) = \widetilde{W}_r(\mathbf{Y}) + bY_n(1-Y_n)$, and $a, b \xleftarrow{\$} \mathbb{F}$. Let $G = \sum_{\mathbf{Y}\in\{0,1\}^n} g(\mathbf{Y})$. The parties essentially check the same sumcheck relation as in Π_{Vrfy}:

$$\widetilde{W}_o(z) + G = \sum_{\mathbf{Y}\in\{0,1\}^n} (f'_z(\mathbf{Y}) + g(\mathbf{Y})) = \sum_{\mathbf{Y}\in\{0,1\}^n} (\boldsymbol{\chi}_z \widetilde{W}'_\ell \widetilde{W}'_r + g)(\mathbf{Y}). \quad (5)$$

For each $j \in [1, n-1]$ and $\omega := (b_{j+1}, \ldots, b_n) \in \{0,1\}^{n-j}$, we simply denote $\widehat{Y}_j = (r_1 \ldots r_{j-1}, Y_j, b_{j+1} \ldots b_n)$ and define polynomials

$$h^j_\omega(Y_j) = h(\widehat{Y}_j), \quad h'^j_\omega(Y_j) = h'(\widehat{Y}_j), \quad \boldsymbol{\chi}^j_{z,\omega}(Y_j) = \boldsymbol{\chi}_z(\widehat{Y}_j)$$
$$f^j_{z,\omega}(Y_j) = f_z(\widehat{Y}_j), \quad f'^j_{z,\omega}(Y_j) = f'_z(\widehat{Y}_j).$$

Note that for $j \in [1, n-1]$, $h(\widehat{Y}_j) = h'(\widehat{Y}_j)$ and therefore $f_z(\widehat{Y}_j) = f'_z(\widehat{Y}_j)$.

For $j \in [1, n-1]$, define $f^j_z(Y_j) = \sum_{\omega\in\{0,1\}^{n-j}} \left(f^j_{z,\omega}(Y_j) + g(\widehat{Y}_j)\right)$ and for $j = n$, simply denote $f'^n_{z,\omega}(Y_n)$ as $f'^n_z(Y_n)$. Recall that if applying the Sumcheck protocol to Eq.(5), the verifier will receive a degree-3 polynomial $f^j_z(Y_j)$ in each round $j \in [1, n-1]$ and a degree-5 polynomial $f'^n_z(Y_n)$ in the final round n. These polynomials should satisfy $f^1_z(0) + f^1_z(1) = \widetilde{W}_o(z) + G$, $f^j_z(0) + f^j_z(1) = f^{j-1}_z(r_{j-1})$, for $j \in [2, n-1]$, and $f'^n_z(0) + f'^n_z(1) = f^{n-1}_z(r_{n-1})$ by definition.

Apply distributed Sumcheck to Eq.(5). The technical difficulty comes from $[\![\cdot]\!]$ is not multiplicative, hence the parties can no longer locally compute sharings of $f_z(\widehat{Y}_j)$ and further $f^j_z(Y_j)$ from $\left[\!\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]\!\right], \left[\!\left[\widetilde{W}_r(\widehat{Y}_j)\right]\!\right]$. As shown in Section 5 that $f^j_z(Y_j)$ has not to be authentically shared, it suffices to use standard Beaver triples to compute $\left[f^j_z(Y_j)\right]$. We remark that the techniques used for achieving privacy (i.e., zero-knowledge) in Section 5 can naturally apply to this setting. Therefore, we concentrate on minimizing the number of consumed Beaver triples.

A direct approach. At a first glance, for each round j, parties need to reconstruct a polynomial $f^j_z(Y_j)$ as $f^j_z(Y_j) := \sum_{\omega\in\{0,1\}^{n-j}} \left(f^j_{z,\omega}(Y_j) + g(\widehat{Y}_j)\right)$. Note that each $f^j_{z,\omega}(Y_j)$ has degree 3 and is defined as $f^j_{z,\omega}(Y_j) = \boldsymbol{\chi}_z(\widehat{Y}_j) \cdot \widetilde{W}_\ell(\widehat{Y}_j) \cdot \widetilde{W}_r(\widehat{Y}_j)$. Since $\boldsymbol{\chi}_z(\widehat{Y}_j)$ is publicly known to all parties, and the parties can locally compute shares $\widetilde{W}_\ell(\widehat{Y}_j), \widetilde{W}_r(\widehat{Y}_j)$ from $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$, it suffices to compute shares of 4 evaluations of $f^j_{z,\omega}(Y_j)$ via 4 Beaver triples. These amount to total $6 + \sum_{j=1}^{n-1} 4 \cdot 2^{n-j} = 4N - 2$ Beaver triples over n rounds (the term "6" is due to that $f'^n_z(Y_n)$ has degree 5).

An optimization via bookkeeping tables. We show the above $4N-2$ triples can be reduced to $2N+1$ via bookkeeping tables. Note that given $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$, for each round $j \in [1,n]$, parties can locally compute the sharings of bookkeeping tables $\mathbf{A}_\ell^j, \mathbf{A}_r^j$ for $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y})$, respectively. Based on $\left[\!\left[\mathbf{A}_\ell^j\right]\!\right], [\![\mathbf{A}_r^j]\!]$, we let parties also maintain a global bookkeeping table $\mathbf{A}_h$ for $h(\mathbf{Y}) = (\widetilde{W}_\ell \widetilde{W}_r)(\mathbf{Y})$. Specifically, for each $j \in [1,n]$, the global table $\mathbf{A}_h^j$ contains $3 \cdot 2^{n-j}$ entries

$$h(r_1, \ldots, r_{j-1}, v, b_{j+1}, \ldots, b_n),$$

where $v \in \{0, \pm 1\}$, $b_{j+1} \ldots b_n \in \{0,1\}$. It is not hard to see that given $\left[\mathbf{A}_h^j\right]$, parties are able to locally compute $\left[f_z^j(Y_j)\right]$.

Now we examine the costs of maintaining bookkeeping tables for $h(\mathbf{Y})$. Assume parties hold $\left[\mathbf{A}_h^j\right]$. For each $\omega := (b_{j+1} \ldots b_n) \in \{0,1\}^{n-j}$, $\left[h_w^j(Y_j)\right]$ can be locally computed from $\left[\mathbf{A}_h^j(0\|\omega)\right]$, $\left[\mathbf{A}_h^j(1\|\omega)\right]$, $\left[\mathbf{A}_h^j(-1\|\omega)\right]$, and thus one can evaluate $\left[h_w^j(Y_j)\right]$ at r_j to obtain $\left[\mathbf{A}_h^{j+1}(\omega)\right]$. To complete the next table $\left[\mathbf{A}_h^{j+1}\right]$, we rely on the two tables $\left[\!\left[\mathbf{A}_\ell^{j+1}\right]\!\right], [\![\mathbf{A}_r^{j+1}]\!]$. Recall that $V_v^{j+1} := (r_1 \ldots r_j, v, b_{j+2} \ldots b_n)$ and $\mathbf{A}_h^{j+1}(-1\|b_{j+2} \cdots b_n) = h(V_{-1}^{j+1}) = (\widetilde{W}_\ell \widetilde{W}_r)(V_{-1}^{j+1})$, where $\left[\!\left[\widetilde{W}_\ell(V_{-1}^{j+1})\right]\!\right]$ (similarly for $\left[\!\left[\widetilde{W}_r(V_{-1}^{j+1})\right]\!\right]$) can be locally computed from $\left[\!\left[\mathbf{A}_\ell^{j+1}(0\|b_{j+2} \cdots b_n)\right]\!\right], \left[\!\left[\mathbf{A}_\ell^{j+1}(1\|b_{j+2} \cdots b_n)\right]\!\right]$. Hence, one standard Beaver triple suffices to compute $\left[\mathbf{A}_h^{j+1}(-1\|b_{j+2} \cdots b_n)\right]$ for each $b_{j+2} \ldots b_n \in \{0,1\}$. Similarly, the first table $\left[\mathbf{A}_h^1\right]$ can be computed from $[\![\mathbf{A}_\ell^1]\!], [\![\mathbf{A}_r^1]\!]$ by consuming $3 \cdot 2^{n-1}$ Beaver triples. These amount to total $3 \cdot 2^{n-1} + (\sum_{j=2}^{n-1} 2^{n-j}) + 3 = 2N+1$ Beaver triples in n rounds (the last term of "3" is due to $(\widetilde{W}'_\ell \widetilde{W}'_r)(\widehat{Y}_n)$ has degree 4 rather than 2, and two more evaluations are required).

The final optimization. Our final optimization comes from observing that $\left[\mathbf{A}_h^1\right]$ can be obtained by using only 2^{n-1} Beaver triples. In more detail, recall that for each $\omega := (b_2, \ldots, b_n) \in \{0,1\}^{n-1}$ and for $v \in \{0,1\}$,

$$\mathbf{A}_h^1(v\|\omega) = \mathbf{A}_\ell^1(v\|\omega)\mathbf{A}_r^1(v\|\omega) = \widetilde{W}_\ell(V_v^1)\widetilde{W}_r(V_v^1) = \widetilde{W}_o(V_v^1)$$

where $\widetilde{W}_o(V_v^1)$ corresponds to the $(v\|\omega)$-th entry of $\mathbf{W}_o$ and has already been shared. Hence, there is no need to compute these 2^n shares from $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$, and we can reuse $[\![\mathbf{W}_o]\!]$.

Missing pieces. A missing piece is that as in Π_{Vrfy}, authenticated openings are required, i.e., $\widetilde{W}_o(z) + G, \widetilde{W}'_\ell(r_1, \ldots, r_n), \widetilde{W}'_r(r_1, \ldots, r_n), g(r_1, \ldots, r_n)$, which are shared by $[\![\cdot]\!]$. To this end, we can apply the standard SPDZ MAC check mechanism [21], presented in π_{VrfyMAC}. We also give formal descriptions of required

functionalities, $\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Auth}}$, and $\mathcal{F}_{\mathsf{Triple}}$. Specifically, $\mathcal{F}_{\mathsf{Coin}}$ provides random consistent values to the parties, and $\mathcal{F}_{\mathsf{Commit}}$ is used in π_{VrfyMAC}. Besides, $\mathcal{F}_{\mathsf{Auth}}$, $\mathcal{F}_{\mathsf{Triple}}$ provide correlated randomness of random authenticated sharings, and random Beaver triples, respectively. Note that $\mathcal{A}$ is allowed to introduce additive errors to triples produced by $\mathcal{F}_{\mathsf{Triple}}$. Putting all pieces together, we present $\Pi_{\mathsf{VrfyAuTriple}}$ that employs the sumcheck idea to check unverified authenticated multiplication triples. The security proof is presented in Appendix D.2.

Protocol 4: π_{VrfyMAC}

Input: $[\![x_1]\!], \ldots, [\![x_\ell]\!]$, where $x_1, \ldots, x_\ell \in \mathbb{F}$ are opened to all parties.
Goal: All parties verify the MACs of $[\![x_1]\!], \ldots, [\![x_\ell]\!]$.

1. Parties invoke $\mathcal{F}_{\mathsf{Coin}}$ and obtain $\alpha_1, \ldots, \alpha_\ell \in \mathbb{F}$.
2. Parties compute $[\sigma] := \sum_{j=1}^{\ell} \alpha_j([\Delta]\, x_j - [M_{x_j}])$. Each party calls $\mathcal{F}_{\mathsf{Commit}}$ with input $[\sigma]$.
3. With $[\sigma]$ being committed, parties open their commitments and check that $\sum_{i=1}^{M} \sigma_i \stackrel{?}{=} 0$. If not, output reject, else output accept.

Protocol 5: $\Pi_{\mathsf{VrfyAuTriple}}$

The goal is to verify $N = 2^n$ (unverified) authenticated Beaver triples (produced by $\mathcal{F}_{\mathsf{AuTripleE}}$). Assume access to $\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Auth}}, \mathcal{F}_{\mathsf{Triple}}$ in an M-party setting.
Input: $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!], [\![\mathbf{W}_o]\!]$ with $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o \in \mathbb{F}^N$.
Correlated Randomness: $N+1$ random Beaver triples. In addition, $3n+5$ random authenticated shares $[\![a]\!], [\![b]\!]$ with $[\![\boldsymbol{g}]\!]$ specifying an n-variable polynomial $g(Y_1, \ldots, Y_n) = g_0 + g_{n,4} Y_n^4 + g_{n,5} Y_n^5 + \sum_{i=1}^{n} (g_{i,1} Y_i + g_{i,2} Y_i^2 + g_{i,3} Y_i^3)$.
Goal: All parties verify that $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$.

1. Parties $\mathrm{P}_1, \ldots, \mathrm{P}_M$ invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $z \xleftarrow{\$} \mathbb{F}^n$.
2. Each party locally computes $\left[\!\left[\widetilde{W}_o(z)\right]\!\right] := \sum_{\omega \in \{0,1\}^n} [\![W_o(\omega)]\!] \cdot \boldsymbol{\chi}_\omega(z)$ and $[\![G]\!] := \sum_{b \in \{0,1\}^n} [\![g(b_1, \ldots, b_n)]\!]$. Then they open $\widetilde{W}_o(x) + G$ and check using π_{VrfyMAC}. If incorrect, all parties reject and abort.
3. For round $j \in [1, n-1]$: consume 2^{n-j} Beaver triples to obtain $[f_z^j(Y_j)]$.
   (a) Locally compute the bookkeeping tables $[\![\mathbf{A}_\ell^j]\!], [\![\mathbf{A}_r^j]\!], \mathbf{A}_{\boldsymbol{\chi}}^j$, and part of $[\mathbf{A}_h^j]$ (i.e., $[\mathbf{A}_\ell^j(v \| \omega)]$, for $v \in \{0,1\}, \omega \in \{0,1\}^{n-j}$).
   (b) For each $\omega := (b_{j+1}, \ldots, b_n) \in \{0,1\}^{n-j}$,
      i. Obtain $\left[\!\left[\widetilde{W}_\ell(V_0^j)\right]\!\right] := [\![\mathbf{A}_\ell^j(0 \| \omega)]\!]$, $\left[\!\left[\widetilde{W}_\ell(V_0^j)\right]\!\right] := [\![\mathbf{A}_\ell^j(1 \| \omega)]\!]$ from the table $[\![\mathbf{A}_\ell^j]\!]$ and interpolate the polynomial $\left[\!\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]\!\right]$. (Interpolate $\left[\!\left[\widetilde{W}_r(\widehat{Y}_j)\right]\!\right], \boldsymbol{\chi}_z(\widehat{Y}_j)$ similarly.)
      ii. Evaluate $\left[\!\left[\widetilde{W}_\ell(\widehat{Y}_j)\right]\!\right], \left[\!\left[\widetilde{W}_r(\widehat{Y}_j)\right]\!\right]$ at point -1 to obtain $\left[\!\left[\widetilde{W}_\ell(V_{-1}^j)\right]\!\right]$, $\left[\!\left[\widetilde{W}_r(V_{-1}^j)\right]\!\right]$. Then, evaluate $\boldsymbol{\chi}_z(\widehat{Y}_j)$ at points $-1, 2$ and obtain $\boldsymbol{\chi}_z(V_{-1}^j), \boldsymbol{\chi}_z(V_2^j)$.

iii. Compute $\left[\mathbf{A}_h^j(-1\|\omega)\right] := \left[\widetilde{W}_\ell(V_{-1}^j)\widetilde{W}_r(V_{-1}^j)\right]$ using one Beaver triple. Fill in the table $\left[\mathbf{A}_h^j\right]$.

iv. Interpolate the polynomial $\left[h_\omega^j(Y_j)\right]$ of degree 2 satisfying $\left[h_\omega^j(v)\right] := \left[\mathbf{A}_h^j(v\|\omega)\right]$, for $v \in \{0, \pm 1\}$. Then evaluate $\left[h_\omega^j(Y_j)\right]$ at point 2, i.e., obtain $\left[h(V_2^j)\right]$.

v. Interpolate the polynomial $\left[f_{z,\omega}^j(Y_j)\right]$ of degree 3 such that $\left[f_{z,\omega}^j(v)\right] := \left[h_\omega^j(v)\right] \cdot \boldsymbol{\chi}_z(V_v^j)$, for $v \in \{0, \pm 1, 2\}$.

(c) Compute $\left[f_z^j(Y_j)\right] := \sum_{\omega\in\{0,1\}^{n-j}} \left(\left[f_{z,\omega}^j(Y_j)\right] + [\![g(r_1, \ldots, r_{j-1}, Y_j, \omega)]\!]\right)$.

(d) Open $f_z^j(Y_j)$ of degree 3. If $j = 1$, all parties check $f_z^1(0) + f_z^1(1) = \widetilde{W}_0(z) + G$, otherwise, they check $f_z^j(0) + f_z^j(1) = f_z^{j-1}(r_{j-1})$.

(e) Invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $r_j \xleftarrow{\$} \mathbb{F}$.

4. For round $j = n$: consume 3 Beaver triples to obtain $[f_z'^n(Y_n)]$.

(a) Compute the bookkeeping tables $[\![\mathbf{A}_\ell^n]\!], [\![\mathbf{A}_r^n]\!], \mathbf{A}_{\boldsymbol{\chi}}^n$, and part of $[\mathbf{A}_h^n]$.

(b) Read $[\![\mathbf{A}_\ell^n(0)]\!], [\![\mathbf{A}_\ell^n(1)]\!]$ from the table $[\![\mathbf{A}_\ell^n]\!]$ and interpolate the polynomial $\left[\!\left[\widetilde{W}_\ell(\widehat{Y}_n)\right]\!\right]$. (Obtain $\left[\!\left[\widetilde{W}_r(\widehat{Y}_n)\right]\!\right], \boldsymbol{\chi}_z(\widehat{Y}_n)$ similarly.)

(c) Evaluate $\left[\!\left[\widetilde{W}_\ell(\widehat{Y}_n)\right]\!\right], \left[\!\left[\widetilde{W}_r(\widehat{Y}_n)\right]\!\right]$ at points $-1, \pm 2$, and obtain $\left[\!\left[\widetilde{W}'_\ell(V_v^n)\right]\!\right] := \left[\!\left[\widetilde{W}_\ell(V_v^n)\right]\!\right] + [\![a]\!]\, v(1-v)$, $\left[\!\left[\widetilde{W}'_r(V_v^n)\right]\!\right] := \left[\!\left[\widetilde{W}_r(V_v^n)\right]\!\right] + [\![b]\!]\, v(1-v)$ for $v \in \{-1, \pm 2\}$. Evaluate $\boldsymbol{\chi}_z(\widehat{Y}_n)$ at points $-1, \pm 2, 3$.

(d) Use 3 Beaver triples to compute $\left[\widetilde{W}'_\ell(V_v^n)\widetilde{W}'_r(V_v^n)\right]$, for $v \in \{-1, \pm 2\}$.

(e) Interpolate the polynomial $[h'(Y_n)]$ of degree 4 satisfying $[h'(0)] := [\![\mathbf{A}_h^n(0)]\!]$, $[h'(1)] := [\![\mathbf{A}_h^n(1)]\!]$, and $[h'(v)] := \left[\widetilde{W}'_\ell(V_v^n)\widetilde{W}'_r(V_v^n)\right]$, for $v \in \{-1, \pm 2\}$. Then evaluate $[h'(Y_n)]$ at point 3.

(f) Interpolate the polynomial $[f_z'^n(Y_n)]$ of degree 5 such that $[f_z'^n(v)] := [h'(v)] \cdot \boldsymbol{\chi}(V_v^n) + [\![g(V_v^n)]\!]$, for $v \in \{-1, \pm 2, 3\}$.

(g) Open $f_z'^n(Y_n)$. All parties check $f_z'^n(0) + f_z'^n(1) = f_z^{n-1}(r_{n-1})$.

(h) Invoke $\mathcal{F}_{\mathsf{Coin}}$ to obtain $r_n \xleftarrow{\$} \mathbb{F}$. Additionally, define $\mathbf{r} := (r_1 \ldots r_n)$.

5. Parties locally compute and open $\left[\!\left[\widetilde{W}'_\ell(\mathbf{r})\right]\!\right] := \left[\!\left[\widetilde{W}_\ell(\mathbf{r})\right]\!\right] + r_n(1-r_n) \cdot [\![a]\!]$, $\left[\!\left[\widetilde{W}'_r(\mathbf{r})\right]\!\right] := \left[\!\left[\widetilde{W}_r(\mathbf{r})\right]\!\right] + r_n(1-r_n) \cdot [\![b]\!]$, $[\![g(\mathbf{r})]\!]$. They check the openings by invoking π_{VrfyMAC}. If the openings are incorrect, all parties reject.

6. All parties verify $f_z'^n(r_n) = (\boldsymbol{\chi}_z \cdot \widetilde{W}'_\ell \cdot \widetilde{W}'_r + g)(\mathbf{r})$.

7. All parties accept if and only if all above checks pass. Otherwise, reject.

Theorem 6.1. *$\Pi_{\mathsf{VrfyAuTriple}}$ securely realizes the functionality $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ in the $(\mathcal{F}_{\mathsf{AuTripleE}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Triple}}, \mathcal{F}_{\mathsf{Auth}})$-hybrid model against a static malicious adversary corrupting up to $M-1$ parties. Moreover, $\Pi_{\mathsf{VrfyAuTriple}}$ has the followings:*

- *Computational complexity: each party performs $O(N)$ operations over $\mathbb{F}$.*
- *Communication complexity: each party sends $4\log N + 6$ elements (for reconstructions), and $2(N+1)$ elements (for Beaver triple openings), calls $\mathcal{F}_{\mathsf{Commit}}$ (for checking MACs) twice.*

– *Correlated randomness: the parties consume* $N+1$ *random Beaver triples from* $\mathcal{F}_{\mathsf{Triple}}$, $3\log N+5$ *random authenticated shares from* $\mathcal{F}_{\mathsf{Auth}}$, *and* $\log N+4$ *random coins from* $\mathcal{F}_{\mathsf{Coin}}$.
– *Round complexity* $O(\log N)$.
– *The adversary* $\mathcal{A}$ *has an advantage of at most* $\frac{4\log N+4}{|\mathbb{F}|}$.

Remark 6.2. Let $\Pi_{\mathsf{AuTriple}} := (\mathcal{F}_{\mathsf{AuTripleE}}, \Pi_{\mathsf{VrfyAuTriple}})$ be a protocol that first invokes $\mathcal{F}_{\mathsf{AuTripleE}}$ to generate unverified authenticated triples and then applies $\Pi_{\mathsf{VrfyAuTriple}}$ to verify the validity of these triples. It is readily to see that Π_{AuTriple} securely realizes $\mathcal{F}_{\mathsf{AuTriple}}$ that produces fully verified authenticated Beaver triples. Furthermore, such triples can directly fit SPDZ online protocol while LeMans [47] cannot produce SDPZ-type authenticated triples.

6.2 Malicious MPC with a Dishonest Majority

In this section, we design a malicious MPC protocol by leveraging $\Pi_{\mathsf{VrfyAuTriple}}$ to check multiplication gates, where the preprocessing phase can be realized in sublinear communication and the online phase only sends $5+o(1)$ field elements per multiplication gate per party.

We follow a similar high-level strategy to Le Mans [47]. The main observation is that partially authenticated multiplication triples $(\llbracket a \rrbracket, \llbracket b \rrbracket, [c])$ also suffice for online circuit evaluations, provided that additional procedures of authenticating c and checking multiplication gates are applied. In particular, Le Mans adopts a checking method inspired by "dual execution" that requires $4+o(1)$ additional field elements per multiplication gate per party. Moreover, as shown in Le Mans, partially authenticated multiplication triples can be generated in sublinear communication by utilizing the programmability of PCGs for OLE and PCGs for VOLE[8]. Therefore, the overall asymptotic communication complexity by Le Mans' approach is $6+o(1)$ field elements per multiplication gate per party.

We present an efficient MPC protocol Π_{MPC}, where multiplication gates are instead verified by $\Pi_{\mathsf{VrfyAuTriple}}$. The functionality for preprocessing is presented in $\mathcal{F}_{\mathsf{Prep}}$, which can be instantiated with programmable PCGs as shown in [47].

Remark 6.3. Note that $\Pi_{\mathsf{VrfyAuTriple}}$ simply assumes errors are only introduced in the output wires, however in fact errors can also be introduced in the input wires due to the use of partially authenticated multiplication triples. We observe that $\Pi_{\mathsf{VrfyAuTriple}}$ still suffices for ensuring security. The reason is that any additive error introduced to some input wire of multiplication gates can be reduced to deviations in computing certain output wire of previous multiplication gates, which then will be caught by $\Pi_{\mathsf{VrfyAuTriple}}$ with high probability.

[8] In fact, the state-of-art-art PCGs for OLEs and VOLEs are based on different paradigms and different variants of LPN assumptions. Hence, their seeds cannot be directly reused and instead a tailored PCG for VOLE is used.

Protocol 6: Π_{MPC}

Let $\mathcal{C}$ be a circuit consisting of $N = 2^n$ multiplication gates and ℓ input gates involving M parties.
Initialize: The parties prepare correlated randomness as follows:

1. Invoke $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{Initialize})$ to get a MAC key share $\Delta_i \in \mathbb{F}$.
2. Invoke $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{Input}, \mathrm{P}_i)$ for all parties to obtain random authenticated shares $[\![r]\!]$ where P_i learns r, for every input that P_i will provide.
3. Invoke $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{AuShare}, N+3n+5)$ to obtain $N+O(n)$ authenticated shares together with their MAC key.
4. Invoke $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{Triple}, N+1)$ to obtain $N+1$ standard Beaver triples (where the adversary may inject additive errors).
5. Invoke $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{AuTriple}, N)$ to obtain N unverified authenticated triples (where the adversary may inject additive errors).

Input: To share an input x held by party P_i:

1. Let $[\![r]\!]$ be an unused random authenticated share. P_i sends $x - r$ to all other parties.
2. The parties compute $[\![x]\!] := [\![r]\!] + (x - r)$.

Add: To add two values $[\![x]\!]$ and $[\![y]\!]$, the parties locally compute $[\![z]\!] = [\![x]\!] + [\![y]\!]$.
Mult: To multiply two values $[\![x]\!]$ and $[\![y]\!]$:

1. Let $([\![a]\!], [\![b]\!], [c])$ be an unused partially authenticated triple and $[\![r]\!]$ a random authenticated share. The parties open $[\![x]\!] - [\![a]\!]$ as α, $[\![y]\!] - [\![b]\!]$ as β and $[c] - [\![r]\!]$ as γ.
2. They locally compute $[\![z]\!] = [\![r]\!] + \gamma + \alpha\,[\![b]\!] + \beta\,[\![a]\!] + \alpha\beta$.

Output: After the evaluation, to output a value $[\![z]\!]$:

1. Call the procedure π_{VrfyMAC} to check the MACs on the values that have been opened in multiplications.
2. Let $\mathbf{W}_o, \mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$ be the outputs and inputs corresponding to N multiplication gates in $\mathcal{C}$. The parties invoke $\Pi_{\mathsf{VrfyAuTriple}}$ on inputs $[\![\mathbf{W}_o]\!], [\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!]$.
3. If both π_{VrfyMAC} and $\Pi_{\mathsf{VrfyAuTriple}}$ output accept, then the parties securely open $[\![z]\!]$. Otherwise, the parties abort.

Corollary 6.4. *Π_{MPC} securely realizes the $\mathcal{F}_{\mathsf{MPC}}$ in the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid model against a static malicious adversary corrupting at most $M-1$ parties. Moreover, Π_{MPC} has the following properties:*

- *Computational complexity: $O(N)$.*
- *Communication: $5N+2$ openings ($3N$ in multiplications, $2N+2$ in $\Pi_{\mathsf{VrfyAuTriple}}$); $4\log N + 6$ reconstructions in $\Pi_{\mathsf{VrfyAuTriple}}$; ℓ broadcasts in inputs; $O(1)$ invocations of π_{VrfyMAC};*
- *Round complexity: $O(\mathsf{depth}(\mathcal{C}) + \log N)$.*

Remark 6.5. Corollary 6.4 directly implies Theorem 1.2. Note that for dishonest majority, we also benefit from a king party optimization, since the Sumcheck proof

messages have not to be robustly reconstructed. Then the total communication is dominated by $10MN$ field elements, less than $12MN$ of Le Mans [47] with the king party optimization. In addition, we can support arbitrary finite fields and rings $\mathbb{Z}_{p^k}$ via using Galois extensions, incurring a $\lambda\times$ overhead, where λ is the security parameter.

Acknowledgments. The authors would like to thank Yuval Ishai and Hongqing Liu for their many helpful discussions and valuable suggestions on this work. This work was supported in part by the National Key Research and Development (R&D) Program of China under Grants 2022YFA1004900, the National Natural Science Foundation of China under Grants 12361141818, 12426302, 12031011, and 12271084, and the Natural Science Foundation of Shanghai under the 2024 Shanghai Action Plan for Science, Technology, and Innovation Grant 24BC3200700.

References

1. Abram, D., Scholl, P.: Low-communication multiparty triple generation for SPDZ from ring-LPN. In: Hanaoka, G., Shikata, J., Watanabe, Y. (eds.) Public-Key Cryptography - PKC 2022. LNCS, vol. 13177, pp. 221–251. Springer (2022). https://doi.org/10.1007/978-3-030-97121-2_9 27
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer (1991) 44
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) STOC 1988. pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213 2
4. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer (2011) 10, 44
5. Bombar, M., Bui, D., Couteau, G., Couvreur, A., Ducros, C., Servan-Schreiber, S.: FOLEAGE: $\mathbb{F}_4$OLE-Based Multi-Party Computation for Boolean Circuits. In: Advances in Cryptology – ASIACRYPT 2024. Springer (2025), ASIACRYPT 2024 44
6. Bombar, M., Couteau, G., Couvreur, A., Ducros, C.: Correlated pseudorandomness from the hardness of quasi-abelian decoding. In: Advances in Cryptology - CRYPTO 2023. LNCS, vol. 14084, pp. 567–601. Springer (2023). https://doi.org/10.1007/978-3-031-38551-3_18 27, 44
7. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology - CRYPTO 2019. LNCS, vol. 11694, pp. 67–97. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_3 3, 4, 6, 7, 9, 10, 20, 22, 43
8. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: Silent OT extension and more. In: CRYPTO 2019. LNCS, vol. 11694, pp. 489–518. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_16 27, 44

9. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators from ring-lpn. In: CRYPTO 2020. LNCS, vol. 12171, pp. 387–416. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_14 27, 44
10. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) CCS 2019. pp. 869–886. ACM (2019). https://doi.org/10.1145/3319535.3363227 9, 10, 44
11. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020. LNCS, vol. 12493, pp. 244–276. Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_9 3, 4, 9, 10, 44, 55
12. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Sublinear GMW-style compiler for MPC with preprocessing. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology - CRYPTO 2021. LNCS, vol. 12826, pp. 457–485. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_16 3, 4, 8, 9, 10, 20, 44, 55
13. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Secure multiparty computation with sublinear preprocessing. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022. LNCS, vol. 13275, pp. 427–457. Springer (2022). https://doi.org/10.1007/978-3-031-06944-4_15 3, 5, 8, 9, 10, 44
14. Boyle, E., Hamilis, M., Gilboa, N., Ishai, Y., Nof, A.: Secure computation with $o(\sqrt{|C|})$ preprocessing. Personal Communication. (2025) 3, 8, 44
15. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) STOC1988. pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214 2
16. Chen, H., Cramer, R.: Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In: CRYPTO 2006. LNCS, vol. 4117, pp. 521–536. Springer (2006) 43
17. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: CRYPTO 2018. LNCS, vol. 10993, pp. 34–64. Springer (2018) 3, 43
18. Corrigan-Gibbs, H.: Protecting privacy by splitting trust. Ph.D. thesis, Stanford University, USA (2019), https://searchworks.stanford.edu/view/13422814 3
19. Dalskov, A.P.K., Escudero, D., Nof, A.: Fast fully secure multi-party computation over any ring with two-thirds honest majority. In: CCS 2022. pp. 653–666. ACM (2022) 43
20. Dalskov, A.P.K., Escudero, D., Nof, A.: Fully secure MPC and zk-FLIOP over rings: New constructions, improvements and extensions. In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology - CRYPTO 2024. LNCS, vol. 14927, pp. 136–169. Springer (2024). https://doi.org/10.1007/978-3-031-68397-8_5 4, 9, 10, 44
21. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) Computer Security - ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer (2013). https://doi.org/10.1007/978-3-642-40203-6_1 29
22. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) Advances in Cryptology - CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer (2007). https://doi.org/10.1007/978-3-540-74143-5_32 6, 26, 39, 43

23. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology - CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer (2012). https://doi.org/10.1007/978-3-642-32009-5_38 2, 10, 27, 39, 44
24. Ding, C., Huang, Y.: Dubhe: Succinct zero-knowledge proofs for standard AES and related applications. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023. pp. 4373–4390. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/ding-changchang 3
25. Ding, C., Huang, Y.: Phecda: Post-quantum transparent zkSNARKs from improved polynomial commitment and VOLE-in-the-head with application in publicly verifiable aes. In: 2025 IEEE Symposium on Security and Privacy (SP). pp. 55–55. IEEE Computer Society (2024) 3
26. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: Turbopack: Honest majority MPC with constant online communication. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) CCS 2022. pp. 951–964. ACM (2022). https://doi.org/10.1145/3548606.3560633 6
27. Furukawa, J., Lindell, Y.: Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. In: CCS 2019. pp. 1557–1571. ACM (2019) 43
28. Garg, S., Goel, A., Jain, A., Policharla, G., Sekar, S.: zkSaaS: Zero-knowledge SNARKs as a service. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023. pp. 4427–4444. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/garg 43
29. Genkin, D., Ishai, Y., Polychroniadou, A.: Efficient multi-party computation: From passive to active security via secure SIMD circuits. In: CRYPTO 2015. LNCS, vol. 9216, pp. 721–741. Springer (2015) 3
30. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Shmoys, D.B. (ed.) STOC 2014. pp. 495–504. ACM (2014). https://doi.org/10.1145/2591796.2591861 3, 4, 5, 20, 44
31. Genkin, D., Ishai, Y., Weiss, M.: Binary AMD circuits from secure multiparty computation. In: TCC 2016. LNCS, vol. 9985, pp. 336–366 (2016) 43, 44
32. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) STOC 1987. pp. 218–229. ACM (1987). https://doi.org/10.1145/28395.28420 2, 3, 4, 9, 44
33. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Dwork, C. (ed.) STOC 2008. pp. 113–122. ACM (2008). https://doi.org/10.1145/1374376.1374396 3, 10, 11, 20, 46
34. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. J. ACM **62**(4), 27:1–27:64 (2015). https://doi.org/10.1145/2699436 3, 11
35. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: efficient and scalable MPC in the honest majority setting. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 244–274. Springer (2021). https://doi.org/10.1007/978-3-030-84245-1_9 26
36. Goyal, V., Song, Y.: Malicious security comes free in honest-majority MPC. IACR Cryptol. ePrint Arch. p. 134 (2020) 2, 3, 5, 6, 7, 8, 9, 10, 22, 26, 55, 57, 58
37. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority MPC. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryp-

tology - CRYPTO 2020. LNCS, vol. 12171, pp. 618–646. Springer (2020). https://doi.org/10.1007/978-3-030-56880-1_22 6, 10, 26, 44
38. Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Actively secure garbled circuits with constant communication overhead in the plain model. In: TCC 2017. LNCS, vol. 10678, pp. 3–39. Springer (2017) 44
39. Hazay, C., Venkitasubramaniam, M., Weiss, M.: The price of active security in cryptographic protocols. In: EUROCRYPT 2020. LNCS, vol. 12106, pp. 184–215. Springer (2020) 43, 44
40. Ishai, Y., Kushilevitz, E., Prabhakaran, M., Sahai, A., Yu, C.: Secure protocol transformations. In: CRYPTO 2016. LNCS, vol. 9815, pp. 430–458. Springer (2016) 43
41. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer (2008) 3, 43, 44
42. Li, Z., Xing, C., Yao, Y., Yuan, C.: Efficient pseudorandom correlation generators for any finite field. In: Fehr, S., Fouque, P. (eds.) EUROCRYPT 2025. LNCS, vol. 15605, pp. 145–175. Springer (2025). https://doi.org/10.1007/978-3-031-91092-0_6 27, 44
43. Lin, F., Xing, C., Yao, Y.: Interactive line-point zero-knowledge with sublinear communication and linear computation. In: ASIACRYPT 2024. pp. 337–366. Springer (2025) 3
44. Liu, T., Xie, T., Zhang, J., Song, D., Zhang, Y.: Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In: IEEE Symposium on Security and Privacy, SP 2024. pp. 1777–1793. IEEE (2024). https://doi.org/10.1109/SP54263.2024.00035, https://doi.org/10.1109/SP54263.2024.00035 43
45. Liu, X., Zhou, Z., Wang, Y., He, J., Zhang, B., Yang, X., Zhang, J.: Scalable collaborative zk-SNARK and its application to efficient proof outsourcing. IACR Cryptol. ePrint Arch. p. 940 (2024), https://eprint.iacr.org/2024/940 43
46. Lund, C., Fortnow, L., Karloff, H.J., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992). https://doi.org/10.1145/146585.146605 3, 16
47. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid MPC for dishonest majority. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022. LNCS, vol. 13507, pp. 719–749. Springer (2022). https://doi.org/10.1007/978-3-031-15802-5_25 3, 5, 9, 32, 34, 44
48. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979) 39
49. Shamir, A.: IP = PSPACE. J. ACM **39**(4), 869–877 (1992). https://doi.org/10.1145/146585.146609 3
50. Thaler, J.: Time-optimal interactive proofs for circuit evaluation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. vol. 8043, pp. 71–89. Springer (2013). https://doi.org/10.1007/978-3-642-40084-1_5 24
51. Thaler, J.: Proofs, arguments, and zero-knowledge. Found. Trends Priv. Secur. **4**(2-4), 117–660 (2022). https://doi.org/10.1561/3300000030 10
52. Vu, V., Setty, S.T.V., Blumberg, A.J., Walfish, M.: A hybrid architecture for interactive verifiable computation. In: SP 2013. pp. 223–237. IEEE Computer Society (2013). https://doi.org/10.1109/SP.2013.48 16
53. Wahby, R.S., Ji, Y., Blumberg, A.J., Shelat, A., Thaler, J., Walfish, M., Wies, T.: Full accounting for verifiable outsourcing. In: Thuraisingham, B., Evans, D., Malkin, T., Xu, D. (eds.) CCS 2017. pp. 2071–2086. ACM (2017). https://doi.org/10.1145/3133956.3133984 11, 20

54. Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: SP 2018. pp. 926–943. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00060 3
55. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) CCS 2022. pp. 3003–3017. ACM (2022). https://doi.org/10.1145/3548606.3560652 43
56. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 733–764. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_24 3, 10, 11, 12, 18, 20, 23, 48, 50
57. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: FOCS. pp. 162–167. IEEE Computer Society (1986) 2, 44
58. Zhang, J., Liu, T., Wang, W., Zhang, Y., Song, D., Xie, X., Zhang, Y.: Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In: Kim, Y., Kim, J., Vigna, G., Shi, E. (eds.) CCS 2021. pp. 159–177. ACM (2021). https://doi.org/10.1145/3460120.3484767 3
59. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: SP 2020. pp. 859–876. IEEE (2020). https://doi.org/10.1109/SP40000.2020.00052 3

Supplementary Material

A More Preliminaries & Functionalities

A.1 Shamir Secret Sharing Scheme

Recall the Shamir secret sharing scheme [48]. Consider an M-party setting over a finite field $\mathbb{F}$. A degree t Shamir sharing of $s \in \mathbb{F}$ is denoted as $[s]_t := (s_1 \dots s_M)$ such that there exists a degree t polynomial $f(\cdot) \in \mathbb{F}[\mathbf{X}]$ satisfying:

$$f(0) = s \text{ and } \forall i \in [1, M], f(i) = s_i.$$

Definition A.1 (Multplicative Secret Sharing Scheme). *If* $\forall x, y \in \mathbb{F}$, $[x] := (x_1 \dots x_M), [y] := (y_1 \dots y_M)$, *there exists a recombination vector* $\boldsymbol{z} := (z_1 \dots z_M)$ *such that*

$$\sum_{j \in [1,M]} \boldsymbol{z}_j \cdot x_j y_j = x \cdot y.$$

The linear secret sharing scheme $[\cdot]$ *is called multiplicative.*

Note that Shamir secret sharing scheme is multiplicative as the entry-wise product can be viewed as a degree $2t$ polynomial evaluation and thus there always exists a recombination vector $\boldsymbol{z}$ from Lagrange interpolation.

Definition A.2 (Double Sharings). *A double sharing for Shamir secret sharing scheme is a pair of the form* $([r]_t, [r]_{2t})$, *where* r *is uniformly random and unknown to the adversary.*

The double sharings can be used to reduce the degree from $2t$ to t during multiplications. [22] proprosed an efficient approach to produce double sharings, with linear amortized communication cost per double-sharing.

A.2 SPDZ Sharing [23]

Consider an M-party setting over a field $\mathbb{F}$. Formally, the authenticated secret-sharing of $x \in \mathbb{F}$ is defined as:

$$[\![x]\!] = (\Delta_i, x_i, \mathrm{M}_{x,i})_{i=1}^{M} \text{ such that } \sum_i x_i = x, \text{ and } \sum_i \mathrm{M}_{x,i} = x \cdot \sum_i \Delta_i,$$

where $x_i, \Delta_i, \mathrm{M}_{x,i} \in \mathbb{F}$ are held by party P_i. We call $\Delta := \sum_i \Delta_i$ the global key and $\mathrm{M}_x := \sum_i \mathrm{M}_{x,i}$ the MAC key of x, where the global key shares Δ_i are fixed for every shared x. This structure ensures authentication, in the sense that $[\![x]\!]$ is *binding* to x.

A.3 Functionalities

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

On receiving inputs $x_1, x_2, \ldots, x_\ell$ from the parties, send $\mathcal{C}(x_1, x_2, \ldots, x_\ell)$ to all parties.

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

$\mathcal{F}_{\mathsf{Coin}}$ samples $r \xleftarrow{\$} \mathbb{F}$ and sends it to all parties.

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

Commit: On input $(\mathsf{Commit}, \mathrm{P}_i, x, \tau_x)$ from P_i, where τ_x is a previously unused identifier, $\mathcal{F}_{\mathsf{Commit}}$ stores $(\mathrm{P}_i, x, \tau_x)$ and sends (P_i, τ_x) to all parties.
Open: On input $(\mathsf{Open}, \mathrm{P}_i, \tau_x)$ from P_i, $\mathcal{F}_{\mathsf{Commit}}$ retrieves x and sends $(x, \mathrm{P}_i, \tau_x)$ to all parties.

Functionality 4: $\mathcal{F}_{\mathsf{Shamir}}$

Suppose M-party, and $t < M/2$ corruptions. Denote by $[\cdot]_t$ the Shamir secret sharing scheme. Let ℓ be the length parameter.

1. $\mathcal{F}_{\mathsf{Shamir}}$ samples $\boldsymbol{g} \xleftarrow{\$} \mathbb{F}^\ell$ and constructs $[g]_t$.
2. $\mathcal{F}_{\mathsf{Shamir}}$ distributes $[g]_t$ to all parties.

Functionality 5: $\mathcal{F}_{\mathsf{Mult}}$

Let n be a positive integer and $N = 2^n$. Suppose M-party, and $t < M/2$ corruptions. Denote by $[\cdot]_t$ the Shamir secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{Mult}}$ receives from honest parties their shares of $[\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t$, which relate to inputs of N multiplications gates. Then $\mathcal{F}_{\mathsf{Mult}}$ reconstructs $\mathbf{W}_\ell, \mathbf{W}_r \in \mathbb{F}^N$.
2. $\mathcal{F}_{\mathsf{Mult}}$ computes the shares of $[\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t$ held by corrupted parties and sends these shares to $\mathcal{A}$.
3. $\mathcal{F}_{\mathsf{Mult}}$ receives from $\mathcal{A}$ a set of shares $\{\mathbf{W}'_{o,i}\}_{i \in T}$ and an additive error $\boldsymbol{\epsilon}$.
4. $\mathcal{F}_{\mathsf{Mult}}$ computes $\mathbf{W}'_o = \mathbf{W}_\ell * \mathbf{W}_r + \boldsymbol{\epsilon}$. Based on the secret $\mathbf{W}'_o$ and t shares $\{\mathbf{W}'_{o,i}\}_{i \in T}$, $\mathcal{F}_{\mathsf{Mult}}$ reconstructs the whole sharing $[\mathbf{W}'_o]_t$ and distributes the shares to honest parties.

Functionality 6: $\mathcal{F}_{\mathsf{VrfyMult}}$

Let n be a positive integer and $N = 2^n$. Suppose M-party, and $t < M/2$ corruptions. Denote by $[\cdot]_t$ the Shamir secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{VrfyMult}}$ receives from honest parties their shares of $[\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t, [\mathbf{W}_o]_t$, which relate to inputs of N multiplications gates. Then $\mathcal{F}_{\mathsf{VrfyMult}}$ reconstructs $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o \in \mathbb{F}^N$.
2. $\mathcal{F}_{\mathsf{VrfyMult}}$ computes the shares of $[\mathbf{W}_\ell]_t, [\mathbf{W}_r]_t, [\mathbf{W}_o]_t$ held by corrupted parties and sends these shares to $\mathcal{A}$.
3. $\mathcal{F}_{\mathsf{VrfyMult}}$ computes $\boldsymbol{\epsilon} := \mathbf{W}_o - \mathbf{W}_\ell * \mathbf{W}_r$.
4. If $\boldsymbol{\epsilon} \neq 0$, $\mathcal{F}_{\mathsf{VrfyMult}}$ sends abort to all parties. Otherwise, $\mathcal{F}_{\mathsf{VrfyMult}}$ sends accept to all parties.

Functionality 7: $\mathcal{F}_{\mathsf{Triple}}$

Suppose M-party, and $t \leq M-1$ corruptions. Denote by $[\cdot]$ the additive secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$. Let ℓ be the length parameter.

1. $\mathcal{F}_{\mathsf{Triple}}$ samples random triples $\mathbf{a}, \mathbf{b}, \mathbf{c} \xleftarrow{\$} \mathbb{F}^\ell$ with $\mathbf{c} = \mathbf{a} * \mathbf{b}$.
2. $\mathcal{F}_{\mathsf{Triple}}$ receives shares $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ from $\mathcal{A}$ for $i \in T$ and $\boldsymbol{\epsilon}$.
3. $\mathcal{F}_{\mathsf{Triple}}$ computes $\mathbf{c}' := \mathbf{c} + \boldsymbol{\epsilon}$. Based on shares from corrupted parties, $\mathcal{F}_{\mathsf{Triple}}$ constructs $[\mathbf{a}], [\mathbf{b}], [\mathbf{c}']$ and distributes the shares to honest parties.

Functionality 8: $\mathcal{F}_{\mathsf{Auth}}$

Suppose M-party, and $t \leq M-1$ corruptions. Denote by $[\![\cdot]\!]$ the authenticated secret sharing scheme. Let ℓ be the length parameter. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{Auth}}$ samples global keys to each honest parties, and receives global keys $\Delta_i \in \mathbb{F}$ from $\mathcal{A}$ for $i \in T$. $\mathcal{F}_{\mathsf{Auth}}$ sets $\Delta := \sum_{i=1}^M \Delta_i$.
2. $\mathcal{F}_{\mathsf{Auth}}$ samples $\boldsymbol{g} \xleftarrow{\$} \mathbb{F}^\ell$ and constructs $[\![\boldsymbol{g}]\!]$.
3. $\mathcal{F}_{\mathsf{Auth}}$ distributes $[\![\boldsymbol{g}]\!]$ to all parties.

Functionality 9: $\mathcal{F}_{\mathsf{AuTripleE}}$

Let n be a positive integer and $N = 2^n$. Suppose M-party, and $t \leq M-1$ corruptions. Denote by $[\![\cdot]\!]$ the authenticated secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{AuTripleE}}$ samples global keys to each honest parties, and receives global keys $\Delta_i \in \mathbb{F}$ from $\mathcal{A}$ for $i \in T$. $\mathcal{F}_{\mathsf{AuTripleE}}$ sets $\Delta := \sum_{i=1}^M \Delta_i$.
2. $\mathcal{F}_{\mathsf{AuTripleE}}$ samples random triples $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o \xleftarrow{\$} \mathbb{F}^N$ with $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$.
3. $\mathcal{F}_{\mathsf{AuTripleE}}$ receives shares $\mathbf{W}_{\ell,i}, \mathbf{W}_{r,i}, \mathbf{W}_{o,i}$ and their MAC keys $\mathbf{M}_{\ell,i}, \mathbf{M}_{r,i}, \mathbf{M}_{o,i}$ from $\mathcal{A}$ for $i \in T$ and $\boldsymbol{\epsilon}$.
4. $\mathcal{F}_{\mathsf{AuTripleE}}$ computes $\mathbf{W}'_o := \mathbf{W}_o + \boldsymbol{\epsilon}$. Based on shares from corrupted parties and the global key Δ, $\mathcal{F}_{\mathsf{AuTripleE}}$ constructs $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!], [\![\mathbf{W}'_o]\!]$ and distributes the shares to honest parties.

Functionality 10: $\mathcal{F}_{\mathsf{VrfyAuTriple}}$

Let n be a positive integer and $N = 2^n$. Suppose M-party, and $t \leq M-1$ corruptions. Denote by $[\![\cdot]\!]$ the authenticated secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ receives global keys Δ_i from all parties and compute $\Delta := \sum_{i=1}^{M} \Delta_i$.
2. $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ receives $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!], [\![\mathbf{W}_o]\!]$ from all parties. Then $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ reconstructs $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o, \mathbf{M}_\ell, \mathbf{M}_r, \mathbf{M}_o$, and checks $\Delta \cdot \mathbf{W}_\ell \stackrel{?}{=} \mathbf{M}_\ell$, $\Delta \cdot \mathbf{W}_r \stackrel{?}{=} \mathbf{M}_r$, $\Delta \cdot \mathbf{W}_o \stackrel{?}{=} \mathbf{M}_o$. If all these check pass, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ continues. Otherwise, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ sends abort to all parties.
3. $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ computes $\boldsymbol{\epsilon} := \mathbf{W}_o - \mathbf{W}_\ell * \mathbf{W}_r$. If $\boldsymbol{\epsilon} \neq 0$, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ sends abort to all parties. Otherwise, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ sends accept to all parties.

Functionality 11: $\mathcal{F}_{\mathsf{AuTriple}}$

Let n be a positive integer and $N = 2^n$. Suppose M-party, and $t \leq M-1$ corruptions. Denote by $[\![\cdot]\!]$ the authenticated secret sharing scheme. Let T with $|T| = t$ denote the set of corrupted parties controlled by the adversary $\mathcal{A}$.

1. $\mathcal{F}_{\mathsf{AuTriple}}$ samples global keys to each honest parties, and receives global keys $\Delta_i \in \mathbb{F}$ from $\mathcal{A}$ for $i \in T$. $\mathcal{F}_{\mathsf{AuTriple}}$ sets $\Delta := \sum_{i=1}^{M} \Delta_i$.
2. $\mathcal{F}_{\mathsf{AuTriple}}$ samples random triples $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o \stackrel{\$}{\leftarrow} \mathbb{F}^N$ with $\mathbf{W}_o = \mathbf{W}_\ell * \mathbf{W}_r$.
3. $\mathcal{F}_{\mathsf{AuTriple}}$ receives shares $\mathbf{W}_{\ell,i}, \mathbf{W}_{r,i}, \mathbf{W}_{o,i}$ and their MAC keys $\mathbf{M}_{\ell,i}, \mathbf{M}_{r,i}, \mathbf{M}_{o,i}$ from $\mathcal{A}$ for $i \in T$ and $\boldsymbol{\epsilon}$.
4. $\mathcal{F}_{\mathsf{AuTriple}}$ computes $\mathbf{W}'_o := \mathbf{W}_o + \boldsymbol{\epsilon}$. Based on shares from corrupted parties and the global key Δ, $\mathcal{F}_{\mathsf{AuTriple}}$ constructs $[\![\mathbf{W}_\ell]\!], [\![\mathbf{W}_r]\!], [\![\mathbf{W}'_o]\!]$ and distributes the shares to honest parties.
5. If $\boldsymbol{\epsilon} \neq 0$, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ sends abort to all parties. Otherwise, $\mathcal{F}_{\mathsf{VrfyAuTriple}}$ sends accept to all parties.

Functionality 12: $\mathcal{F}_{\mathsf{Prep}}$

Initialize: On input (Init) from all parties, $\mathcal{F}_{\mathsf{Prep}}$ samples global keys to each honest parties, and receives global keys $\Delta_i \in \mathbb{F}$ from $\mathcal{A}$ for $i \in T$. $\mathcal{F}_{\mathsf{Prep}}$ sets $\Delta := \sum_{i=1}^{M} \Delta_i$.

Input: On input $(\mathsf{Input}, \mathrm{P}_i)$ from all parties, do the following:

1. $\mathcal{F}_{\mathsf{Prep}}$ samples $r \stackrel{\$}{\leftarrow} \mathbb{F}$ and constructs $[\![r]\!]$.
2. $\mathcal{F}_{\mathsf{Prep}}$ sends r to P_i and distributes $[\![r]\!]$ to all parties.

Authenticated Share: On input $(\mathsf{AuShare}, \ell)$ from all parties, do the following:

1. $\mathcal{F}_{\mathsf{Prep}}$ samples $\boldsymbol{r} \stackrel{\$}{\leftarrow} \mathbb{F}^\ell$ and constructs $[\![\boldsymbol{r}]\!]$.
2. $\mathcal{F}_{\mathsf{Prep}}$ distributes $[\![\boldsymbol{r}]\!]$ to all parties.

Triple: On input (Triple, ℓ) from all parties, $\mathcal{F}_{\mathsf{Prep}}$ performs as follows

1. $\mathcal{F}_{\mathsf{Prep}}$ samples random triples $\mathbf{a}, \mathbf{b}, \mathbf{c} \xleftarrow{\$} \mathbb{F}^{\ell}$ with $\mathbf{c} = \mathbf{a} * \mathbf{b}$.
2. $\mathcal{F}_{\mathsf{Prep}}$ receives shares $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ from $\mathcal{A}$ for $i \in T$ and $\boldsymbol{\epsilon}$.
3. $\mathcal{F}_{\mathsf{Prep}}$ computes $\mathbf{c}' := \mathbf{c} + \boldsymbol{\epsilon}$. Based on shares from corrupted parties, $\mathcal{F}_{\mathsf{Prep}}$ constructs $[\mathbf{a}], [\mathbf{b}], [\mathbf{c}']$ and distributes the shares to honest parties.

Partially Authenticated Triple: On input $(\mathsf{AuTriple}, \ell)$ from all parties, do:

1. $\mathcal{F}_{\mathsf{Prep}}$ samples random triples $\mathbf{a}, \mathbf{b}, \mathbf{c} \xleftarrow{\$} \mathbb{F}^{\ell}$ with $\mathbf{c} = \mathbf{a} * \mathbf{b}$.
2. $\mathcal{F}_{\mathsf{Prep}}$ receives shares $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i$ and their MAC keys $\mathbf{M}_{a,i}, \mathbf{M}_{b,i}$ from $\mathcal{A}$ for $i \in T$ and $\boldsymbol{\epsilon}$.
3. $\mathcal{F}_{\mathsf{Prep}}$ computes $\mathbf{c}' := \mathbf{c} + \boldsymbol{\epsilon}$. Based on shares from corrupted parties and the global key Δ, $\mathcal{F}_{\mathsf{Prep}}$ constructs $[\![\mathbf{a}]\!], [\![\mathbf{b}]\!], [\mathbf{c}']$ and distributes the shares to honest parties.

B Related Work

Very recently, there is a line of works [55,28,44,45] consider distributing the proof generation of sumcheck-based SNARKs to many workers for acceleration. Since their motivation is totally different from ours, typically, these works mainly focus on the concrete efficiency, with privacy and security being less concerned. We then list the-state-of-art results for achieving malicious MPC protocols in the following, classified by the fraction of honest parties.

Strong Honest Majority Setting. For less than $(1-\epsilon)/3$ fraction of corrupted parties and Boolean circuits, Genkin et al. [31] achieved $\mathsf{polylog}(|C|, \kappa)$ overhead, where $|C|$ is the circuit size and κ is the statistical security parameter. For at least two-thirds honest parties, [27] achieved $O(d)$ additive communication with fairness using Shamir's secret sharing scheme, where d is the depth of the circuit, while [19] achieved constant additive communication for full security. However, the scheme of [19] is efficient only for a small number of parties, as it heavily relies on replicated secret sharing. Moreover, the computational overhead of these works remain unclear.

Honest Majority Setting. By exploiting the "dual execution" idea, [17] demonstrated active security with abort for arithmetic circuits, achieving a multiplicative overhead of 2 compared to passive protocols for sufficiently large fields. Following the IPS paradigm [41], [40] achieved a constant multiplicative communication overhead for arbitrary fields but only for a constant number of parties, later [39] extended to support an arbitrary number of parties, building on a variant of [22] instantiated with algebraic geometric secret sharing [16]. However, the computational overhead is unclear due to the use of algebraic geometry codes. With the introduction of FLPCP/FLIOPs by Boneh et al. [7], follow-up works

[10,11,37,20] achieve malicious security with abort (even full security) with additive logarithmic communication in various settings. However, the additional computation is at least quasi-linear in the circuit size since the underlying FLIOP has a quasi-linear time prover.

Dishonest Majority Setting. In this setting, there are two ways of measuring the overhead of achieving malicious security.

Over the passive GMW protocol [32]. For Boolean circuits, [41] and [38] demonstrated constant communication overhead over GMW and garbled circuits [57]. For arithmetic computations, [30] established constant communication overhead in the OLE-hybrid model over GMW for sufficiently large fields, by introducing AMD circuits. Notably, [31] also works in the dishonest majority setting, and has $\mathsf{polylog}(|C|, \kappa)$ communication overhead for an arbitrary number of parties. In addition, Hazay, Venkitasubramaniam, and Weiss [39] based on IPS compiler [41] achieved constant communication overhead (2 in the best case), though their result remains mostly theoretical.

Over the preprocessing model. Many modern concretely efficient MPC protocols in the dishonest majority works in the preprocessing model. Specifically, during an offline phase, the parties execute a circuit/input independent procedure to generate correlated randomness. Once inputs are available, the parties run a lightweight, non-cryptographic online phase that consumes the correlated randomness. Hence, an efficient online protocol should have low communication as well as correlated randomness consumption. The correlated randomness typically comes in two forms: multiplication triples [2] and authenticated multiplication triples [4,23], for semi-honest and malicious security, respectively. Informally, authenticated multiplication triples enhance multiplication triples by using linearly homomorphic message authentication codes (MACs). Intuitively, secret-sharing with MACs offers authentication, so that malicious adversary's deviation could be detected. Recent advancements in pseudorandom correlation generators (PCGs) [8,9,6,5,42] have demonstrated concretely efficient methods for generating multi-party multiplication triples, or two-party authenticated multiplication triples with *sublinear* communication complexity even for arbitrary fields. However, it still remains open how to extend these techniques to efficiently generate authenticated multiplication triples for more than two parties.

Recent works [12,13,47] presented MPC with $O(|C|)$ online communication per party with sublinear preprocessing. Roughly speaking, [12,13] extend the FLIOP approach to dishonest majority setting, by introducing a star-sharing framework. Very concurrently, Boyle et al. [14] greatly improve over this routine and put forward MPC with $2|C| + o(1)$ communication and linear computation based on LPN-style assumptions. A minor drawback is that the online phase would no longer be information-theoretic secure. In contrast, Le Mans [47] proposed a simpler approach for producing unverified authenticated triples by leveraging the programmability of PCGs for multiplication triples and VOLE. However,

their approach allows the adversary to inject additive errors to the unverified authenticated triples. To solve this issue, they employ the "dual execution" idea and essentially move the check to the online phase.

C Sumcheck for Other Relations

In this section, we characterize typical relations appeared in MPC scenarios by Sumcheck relations, so that one can replace the original check by using our distributed Sumcheck techniques.

C.1 Multiplication Triple

We first recap multiplication triple relations. Formally, given $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{F}^N$, the goal is to verify that $w_i = u_i v_i$, for each $i \in [0, N-1]$.
Notations. W.l.o.g., assume $N = 2^n$. View $\boldsymbol{w}$ as a function $w : \{0,1\}^n \to \mathbb{F}$, such that $w(i) = w_i$, for $i \in \{0,1\}^n = [0, N-1]$. Let $P_z : \{0,1\}^n \to \{0,1\}$ denote the point predicate, such that $P_z(i) = 1$ if and only if $z = i$, and otherwise $P_z(i) = 0$, where $i \in \{0,1\}^n$. We have the following:

$$w(z) = \sum_{i \in \{0,1\}^n} P_z(i) \cdot u(i) \cdot v(i).$$

Hence, the multilinear extensions satisfy the following:

$$\widetilde{w}(z) = \sum_{y \in \{0,1\}^n} \widetilde{P}_z(y) \cdot \widetilde{u}(y) \cdot \widetilde{v}(y) = \sum_{y \in \{0,1\}^n} \boldsymbol{\chi}_x(z) \cdot \widetilde{u}(y) \cdot \widetilde{v}(y),$$

for every $z \in \mathbb{F}^n$. Thus, one can run Sumcheck with $z \xleftarrow{\$} \mathbb{F}^n$.

C.2 Circuit Dependent Preprocessing

Multiplication triples are used for circuit independent preprocessing. Nowadays, many MPC protocols employ circuit dependent preprocessing to further decrease online phase communication. Here, we consider relations for circuit dependent preprocessing. Specifically, given r_x, r_y, r_z as the masks of the three wires of a multiplication gate, and the wire values (x, y, z), the masked values are computed as $(\Lambda_x, \Lambda_y, \Lambda_z) \leftarrow (x + r_x, y + r_y, z + r_z)$. Generally, the protocol works as follows. Given (Λ_x, Λ_y) and $(r_x, r_y, r_x \cdot r_y + r_z)$ being additively shared among the parties, Λ_z is computed as

$$\begin{aligned} \Lambda_z \leftarrow z + r_z &= (\Lambda_x - r_x) \cdot (\Lambda_y - r_y) + r_z \\ &= \Lambda_x \cdot \Lambda_y + r_x \cdot r_y + r_z - r_x \cdot \Lambda_y - r_y \cdot \Lambda_x. \end{aligned}$$

After the protocol execution, the invariant becomes that

$$\Lambda_z - r_z = (\Lambda_x - r_x) \cdot (\Lambda_y - r_y) = \Lambda_x \cdot \Lambda_y + r_x \cdot r_y - r_x \cdot \Lambda_y - r_y \cdot \Lambda_x.$$

Let $W_x, W_y, W_z : \{0,1\}^n \to \mathbb{F}$ be the functions for (r_x, r_y, r_z) and $P_x, P_y, P_z : \{0,1\}^n \to \mathbb{F}$ the functions for $(\Lambda_x, \Lambda_y, \Lambda_z)$. Then the problem reduces to the following Sumcheck:

$$\widetilde{W}_z(u) = \sum_{\omega \in \{0,1\}^n} \boldsymbol{\chi}_u(\omega) \cdot \big(\widetilde{P}_z(\omega) - \widetilde{W}_x(\omega) \cdot \widetilde{W}_y(\omega) + \widetilde{P}_x(\omega) \cdot \widetilde{W}_y(\omega) + \widetilde{W}_x(\omega) \cdot \widetilde{P}_y(\omega) - \widetilde{P}_x(\omega) \cdot \widetilde{P}_y(\omega)\big),$$

for $u \xleftarrow{\$} \mathbb{F}^n$. Note that the MLEs $(\widetilde{P}_x, \widetilde{P}_y, \widetilde{P}_z)$ can be explicitly computed by all of the parties as $(\Lambda_x, \Lambda_y, \Lambda_z)$ will be explicitly recovered during the protocol execution.

C.3 Rank One Constraint System

R1CS is a popular NP language. We consider proving R1CS in the MPC setting. Specifically, given three public matrices $A, B, C \in \mathbb{F}^{k \times m}$ and a witness $\mathbf{z} \in \mathbb{F}^m$ robustly shared by the parties, the goal is to prove that $A\mathbf{z} * B\mathbf{z} = C\mathbf{z}$, where $*$ denotes entry-wise multiplication.

Define $\mathbf{a}, \mathbf{b}, \mathbf{c}$ as $\mathbf{a} := A \cdot \mathbf{z}, \mathbf{b} := B \cdot \mathbf{z}, \mathbf{c} := C \cdot \mathbf{z}$. The R1CS satisfiability problem reduces to verifying $\mathbf{a}[i] \cdot \mathbf{b}[i] = \mathbf{c}[i]$ for each $i \in [1, k]$. Note that the parties can locally compute secret sharings of $\mathbf{a}, \mathbf{b}, \mathbf{c}$, since we always assume a linear secret sharing scheme. Let $\widetilde{\mathbf{a}}, \widetilde{\mathbf{b}}, \widetilde{\mathbf{c}}$ be the multilinear extension of $\mathbf{a}, \mathbf{b}, \mathbf{c}$ viewed as multivariate functions. Then the R1CS problem reduces to the following:

$$\widetilde{\mathbf{c}}(x) = \sum_{\omega} \boldsymbol{\chi}_x(\omega) \cdot \widetilde{\mathbf{a}}(\omega) \cdot \widetilde{\mathbf{b}}(\omega),$$

which is equivalent to check multiplications.

C.4 Circuit Evaluation

The celebrated GKR [33] protocol employs the Sumcheck protocol to prove circuit satisfiability. The core idea is to capture the relations between two adjacent layers of the circuit by Sumcheck relations. Specifically, let $\mathbf{W}_{j+1} \in \mathbb{F}^{N_{j+1}}$ be the input to layer j, and $\mathbf{W}_j \in \mathbb{F}^{N_j}$ be the output. In the MPC setting, typically $\mathbf{W}_{j+1}, \mathbf{W}_j$ are shared by the parties. As in GKR, we define "wiring predicates" mult_j (add_j) which take as inputs three labels (two from W_{j+1} and one from W_j), and output 1 if and only if they correspond to a multiplication (addition) gate in layer j, otherwise output 0. With $W_j, \mathsf{mult}_j, \mathsf{add}_j$ defined as above, it holds that for any $z \in \mathbb{F}$,

$$\widetilde{W}_j(z) = \sum_{x,y \in \{0,1\}^{n_{j+1}}} \widetilde{\mathsf{mult}}_j(z,x,y)\widetilde{W}_{j+1}(x)\widetilde{W}_{j+1}(y) + \widetilde{\mathsf{add}}_j(z,x,y)\big(\widetilde{W}_{j+1}(x) + \widetilde{W}_{j+1}(y)\big),$$

In a bare-bone sketch, GKR proceeds from output layer to input layer sequentially, and employs the Sumcheck protocol for the layer-by-layer reduction. We find that semi-honest MPC protocols that are based on non-linear secret sharing schemes, or not secure up to additive attacks may benefit from a distributive analogue of GKR.

D Security Proofs

D.1 Proof of Theorem 5.2

Proof (Proof of security of Π_{Vrfy}). For completeness, we give a detailed proof, which is essentially a simplified version of the proof of $\Pi_{\mathsf{VrfyAuTriple}}$ for dishonest majority. We construct a simulator $\mathcal{S}$ for our protocol and show that the view it generates is statistically indistinguishable to the adversary $\mathcal{A}$'s view in a real execution. Let T be the set of corrupted parties controlled by $\mathcal{A}$. The simulator $\mathcal{S}$ receives $\{\boldsymbol{\epsilon}, (\boldsymbol{W}_{\ell,i}, \boldsymbol{W}_{r,i}, \boldsymbol{W}_{o,i})_{i\in T}\}$ as an input, and then interacts with $\mathcal{A}$ playing the role of the honest parties. $\mathcal{S}$ also emulates functionalities $\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Shamir}}$ that output random coins and random Shamir shares, respectively. In particular, $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ randomly picks $\boldsymbol{W}_\ell, \boldsymbol{W}_r, \boldsymbol{W}_o^* \xleftarrow{\$} \mathbb{F}^N$ with $\boldsymbol{W}_o^* = \boldsymbol{W}_\ell * \boldsymbol{W}_r + \boldsymbol{\epsilon}$ and constructs random Shamir secret sharings for honest parties conditioned on $\mathcal{A}$'s shares.
2. Distributing correlated randomness: $\mathcal{S}$ constructs Shamir secret sharings of $3n+5$ random elements of $\mathbb{F}$, and sends corresponding shares to corrupted parties controlled by $\mathcal{A}$.
3. Emulating $\mathcal{F}_{\mathsf{Coin}}$: $\mathcal{S}$ samples $z \xleftarrow{\$} \mathbb{F}^n$ and sends it to $\mathcal{A}$.
4. $\mathcal{S}$ receives $(\widetilde{W}^*_{o,i}(z) + G_i)$ from $\mathcal{A}$, and at the same time, sends shares to $\mathcal{A}$ as the role of each honest party. Then $\mathcal{S}$ checks $\widetilde{W}^*_{o,i}(z) + G_i \stackrel{?}{=} \sum_{\omega\in\{0,1\}^n} W_{o,i}(\omega)\cdot \boldsymbol{\chi}_\omega(z) + \sum_{\mathbf{Y}\in\{0,1\}^n} g_i(\mathbf{Y})$, for each $i \in T$. $\mathcal{S}$ accepts if and only if all these checks pass. This procedure corresponds to authentically open Shamir secrets in the real execution, where the honest parties will always reject an incorrect $\widetilde{W}^*_{o,i}(z) + G_i$. Moreover, if all $\widetilde{W}^*_{o,i}(z) + G_i$ sent by $\mathcal{A}$ are correct, then in the real execution, $(\widetilde{W}^*_o(z) + G) = (\widetilde{W}_o(z) + G)$ happens with probability at most $\frac{n}{|\mathbb{F}|}$ by the Schwartz-Zippel Lemma.
5. Emulating Round j, where $j \in [1, n]$:
   (i) $\mathcal{S}$ sends shares (of $f_z^j(Y_j)$) to $\mathcal{A}$ as the role of each honest party, and at the same time, receives those from $\mathcal{A}$, for party $i \in T$.
   (ii) Reconstruct the polynomial from the shares, where might exist additive errors introduced by $\mathcal{A}$, denoted by $f_z^{*j}(Y_j) := f_z^j(Y_j) + f_\delta^j(Y_j)$. $\mathcal{S}$ checks whether $f_\delta^j(0) + f_\delta^j(1) = f_z^{*(j-1)}(r_{j-1}) - f_z^{j-1}(r_{j-1})$. (If $j = 1$, $\mathcal{S}$ instead checks whether $f_\delta^j(0) + f_\delta^j(1) = \widetilde{W}^*_o(z) - \widetilde{W}_o(z)$.)
   (iii) Emulating $\mathcal{F}_{\mathsf{Coin}}$: $\mathcal{S}$ samples $r_j \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{A}$.
6. $\mathcal{S}$ sends shares to $\mathcal{A}$ as the role of each honest party, and at the same time, receives from $\mathcal{A}$ the party i's sharings of $\widetilde{W}'_\ell(\mathbf{Y}), \widetilde{W}'_r(\mathbf{Y}), g(\mathbf{Y})$ evaluated at $(r_1, \ldots, r_n)$, for each $i \in T$. $\mathcal{S}$ checks whether these openings are correct since $\mathcal{S}$ knows the actual shares of corrupted parties. $\mathcal{S}$ rejects if any of the above checks fails.
7. $\mathcal{S}$ rejects as long as $\boldsymbol{\epsilon} \neq 0$.

Here we rely on the induction hypothesis that $\widetilde{W}^*_o(z) \neq \widetilde{W}_o(z)$. Consider the reconstructed polynomial $f_z^{*1}(Y_1)$ in round 1, which should satisfy $\widetilde{W}^*_o(z) + G = f_z^{*1}(0) + f_z^{*1}(1)$. Otherwise, honest parties will reject eventually.

– If $f_z^{*1}(Y_1) = f_z^1(Y_1)$, then $\widetilde{W}_o^*(z) + G = f_z^{*1}(0) + f_z^{*1}(1) = f_z^1(0) + f_z^1(1) = \widetilde{W}_o(z) + G$, which implies a contradiction.
– If $f_z^{*1}(Y_1) \neq f_z^1(Y_1)$, then $f_z^{*1}(r_1) = f_z^1(r_1)$ happens with probability at most $\frac{3}{|\mathbb{F}|}$, for a random $r_1 \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma. Note that $f_z^{*1}(r_1) = f_z^1(r_1)$ is equivalent to $f_\delta^1(r_1) = 0$.

Thus we obtain the induction hypothesis for the next round, i.e., $f_z^{*1}(r_1) \neq f_z^1(r_1)$. For round $j \in [2, n-1]$, assume the induction hypothesis $f_z^{*(j-1)}(r_{j-1}) \neq f_z^{j-1}(r_{j-1})$, and consider the reconstructed polynomial $f_z^{*j}(Y_j)$. Similarly, $f_z^{*j}(Y_j)$ needs to satisfy $f_z^{*(j-1)}(r_{j-1}) = f_z^{*j}(0) + f_z^{*j}(1)$.

– If $f_z^{*j}(Y_j) = f_z^j(Y_j)$, then $f_z^{*(j-1)}(r_{j-1}) = f_z^{*j}(0) + f_z^{*j}(1) = f_z^j(0) + f_z^j(1) = f_z^{j-1}(r_{j-1})$, which implies a contradiction.
– If $f_z^{*j}(Y_j) \neq f_z^j(Y_j)$, then $f_z^{*j}(r_j) = f_z^j(r_j)$ happens with probability at most $\frac{3}{|\mathbb{F}|}$, for a random $r_j \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma. Note that $f_z^{*j}(r_j) = f_z^j(r_j)$ is equivalent to $f_\delta^j(r_j) = 0$.

For round n, assume the induction hypothesis $f_z^{*(n-1)}(r_{n-1}) \neq f_z^{n-1}(r_{n-1})$, and consider the reconstructed polynomial $f_z'^{*n}(Y_n)$, which should satisfy $f_z^{*(n-1)}(r_{n-1}) = f_z'^{*n}(0) + f_z'^{*n}(1)$. Recall that $f_z^{n-1}(Y_{n-1}) = f_z'^{(n-1)}(Y_{n-1})$.

– If $f_z'^{*n}(Y_n) = f_z'^n(Y_n)$, then $f_z^{*(n-1)}(r_{n-1}) = f_z'^{*n}(0) + f_z'^{*n}(1) = f_z'^n(0) + f_z'^n(1) = f_z'^{(n-1)}(r_{n-1}) = f_z^{n-1}(r_{n-1})$, which implies a contradiction.
– If $f_z'^{*n}(Y_n) \neq f_z'^n(Y_n)$, then $f_z'^{*n}(r_n) = f_z'^n(r_n) = (\widetilde{W}'_\ell \widetilde{W}'_r \boldsymbol{\chi} + g)(r_1, \ldots, r_n)$ happens with probability at most $\frac{5}{|\mathbb{F}|}$, for a random $r_n \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma.

Note that both $\mathcal{S}$ and honest parties will reject if for some round j, $f_z^{*(j-1)}(r_{j-1}) \neq f_z^{*j}(0) + f_z^{*j}(1)$, even if $\boldsymbol{\epsilon} = 0$. We claim that the reconstructed values as well as their sharings, i.e., the entire emulated Sumcheck proof, are independent of the inputs $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o$. Intuitively, this is because that the parties essentially apply a Sumcheck protocol on $(f'_z + g)(\mathbf{Y})$, which is zero-knowledge as shown in Libra [56].

Recall that the reconstructed values are, $\widetilde{W}_o(z) + G$, $f_z'^j(Y_j)$ for $j \in [1, n]$, and $\widetilde{W}'_\ell(r_1, \ldots, r_n)$,$\widetilde{W}'_r(r_1, \ldots, r_n)$,$g(r_1, \ldots, r_n)$. First, $\widetilde{W}'_\ell(r_1, \ldots, r_n) := \widetilde{W}_\ell(r_1, \ldots, r_n) + ar_n(1 - r_n)$ and $\widetilde{W}'_r(r_1, \ldots, r_n) := \widetilde{W}_r(r_1, \ldots, r_n) + br_n(1 - r_n)$ are uniformly random by the randomness of a, b. Hence it suffices to consider the leakage on the polynomial $f'_z(\mathbf{Y})$. In fact, each $f_z'^j(Y_j)$ ($j \in [1, n-1]$) reveals 4 linear combinations of coefficients of the polynomial $(f'_z + g)(\mathbf{Y})$ (6 for $f_z'^n(Y_n)$). In addition, there are $n+1$ linear constraints of these polynomials, as it should hold that $\widetilde{W}_o(z) + G = f_z'^1(0) + f_z'^1(1)$, $f_z'^{(j-1)}(r_{j-1}) = f_z'^j(0) + f_z'^j(1)$ for $j \in [2, n]$ and $f_z'^n(r_n) = (f'_z + g)(r_1, \ldots, r_n)$. Hence, there are total $\big(1{+}4(n{-}1){+}6{+}1\big) - (n{+}1) = 3n + 3$ independent linear constraints on $(f'_z + g)(\mathbf{Y})$ and a masking polynomial

g of the type $g := g_0 + g_{n,4} \cdot Y_n^4 + g_{n,5} \cdot Y_n^5 + \sum_{i=1}^{n} \sum_{j=1}^{3} g_{i,j} \cdot Y_i^j$ with $3n+3$ random coefficients suffices.

Moreover, the sharings of reconstructed values are also distributed uniformly at random because each party's shares are masked by their shares of the masking polynomial $g(\mathbf{Y})$.

Hence the adversary $\mathcal{A}$'s indistinguishability advantage is upper bounded by $\frac{4n+2}{|\mathbb{F}|}$ by a union bound. □

D.2 Proof of Theorem 6.1

Proof (Proof of security of $\Pi_{\mathsf{VrfyAuTriple}}$). We construct a simulator $\mathcal{S}$ for our protocol and show that the view it generates is statistically indistinguishable to the adversary $\mathcal{A}$'s view in a real execution. Let T be the set of corrupted parties controlled by $\mathcal{A}$. The simulator $\mathcal{S}$ receives $\{\boldsymbol{\epsilon}, (\boldsymbol{W}_{\ell,i}, \boldsymbol{W}_{r,i}, \boldsymbol{W}_{o,i}, \boldsymbol{M}_{\ell,i}, \boldsymbol{M}_{r,i}, \boldsymbol{M}_{o,i}, \Delta_i)_{i\in T}\}$ as an input (because $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{AuTripleE}}$), and then interacts with $\mathcal{A}$ playing the role of the honest parties. $\mathcal{S}$ also emulates functionalities $\mathcal{F}_{\mathsf{Coin}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{Auth}}, \mathcal{F}_{\mathsf{Triple}}$. In particular, $\mathcal{S}$ works as follows:

1. $\mathcal{S}$ randomly picks $\boldsymbol{W}_\ell, \boldsymbol{W}_r, \boldsymbol{W}_o^* \xleftarrow{\$} \mathbb{F}^N$ with $\boldsymbol{W}_o^* = \boldsymbol{W}_\ell * \boldsymbol{W}_r + \boldsymbol{\epsilon}$ and constructs random authenticated secret sharings for honest parties conditioned on $\mathcal{A}$'s shares.
2. Distributing correlated randomness: $\mathcal{S}$ samples n random Beaver triples over $\mathbb{F}$ and $3n+5$ random authenticated sharings, and sends the corresponding shares to corrupted parties controlled by $\mathcal{A}$[9].
3. Emulating $\mathcal{F}_{\mathsf{Coin}}$: $\mathcal{S}$ samples $z \xleftarrow{\$} \mathbb{F}^n$ and sends it to $\mathcal{A}$.
4. Emulating π_{VrfyMAC}: $\mathcal{S}$ receives $(\widetilde{W}_{o,i}^*(z) + G_i)$ from $\mathcal{A}$, and at the same time, sends corresponding shares to $\mathcal{A}$ as the role of each honest party. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Commit}}$, where $\mathcal{S}$ receives MAC key openings from $\mathcal{A}$ and at the same time, informs $\mathcal{A}$ that honest parties' MAC key openings are received. After that, $\mathcal{S}$ sends MAC key openings of honest parties to $\mathcal{A}$.
5. Then $\mathcal{S}$ checks $\widetilde{W}_{o,i}^*(z) + G_i \stackrel{?}{=} \sum_{\omega\in\{0,1\}^n} W_{o,i}(\omega) \cdot \boldsymbol{\chi}_\omega(z) + \sum_{\mathbf{Y}\in\{0,1\}^n} g_i(\mathbf{Y})$, for each $i \in T$. $\mathcal{S}$ accepts if and only if all these checks pass. This procedure corresponds to check validity of MAC openings in the real execution, where the honest parties would accept an incorrect $\widetilde{W}_{o,i}^*(z) + G_i$ with probability at most $\frac{1}{|\mathbb{F}|}$. Moreover, if all $\widetilde{W}_{o,i}^*(z) + G_i$ sent by $\mathcal{A}$ are correct, then in the real execution, $(\widetilde{W}_o^*(z) + G) = (\widetilde{W}_o(z) + G)$ happens with probability at most $\frac{n}{|\mathbb{F}|}$ by the Schwartz-Zippel Lemma.
6. Emulating Round j, where $j \in [1, n]$:
   (i) $\mathcal{S}$ acts as honest parties for $i \notin T$, except that all messages being sent to $\mathcal{A}$ are uniformly sampled from $\mathbb{F}$. In more detail, $\mathcal{S}$ sends Beaver triple openings to $\mathcal{A}$ as the role of each honest party, and at the same time,

[9] Here for simplicity, we assume $\mathcal{A}$ does not inject additive errors in distributing random Beaver triples, since such additive errors can be captured by the additive errors introduced in opening $\left[f_z^j(Y_j)\right]$.

receives Beaver triple openings from $\mathcal{A}$. After that, $\mathcal{S}$ sends shares (of $f_z^j(Y_j)$) to $\mathcal{A}$ as the role of each honest party, and at the same time, receives those from $\mathcal{A}$, for party $i \in T$.

(ii) Note that $\mathcal{S}$ can detect $\mathcal{A}$'s deviations in Beaver triple openings (however, $\mathcal{S}$ cannot compute the corresponding additive error polynomial $f_{\delta_1}^j(Y_j)$, which depends on the inputs). While, $\mathcal{S}$ can compute the additive error polynomial introduced by $\mathcal{A}$ in opening $f_z^j(Y_j)$, denoted by $f_{\delta_2}^j(Y_j)$. $\mathcal{S}$ checks $f_{\delta_2}^j(0) + f_{\delta_2}^j(1) = 0$.

(iii) Emulating $\mathcal{F}_{\mathsf{Coin}}$: $\mathcal{S}$ samples $r_j \xleftarrow{\$} \mathbb{F}$ and sends it to $\mathcal{A}$.

7. Emulating π_{VrfyMAC}: $\mathcal{S}$ sends shares to $\mathcal{A}$ as the role of each honest party, and at the same time, receives from $\mathcal{A}$ the party i's sharings of $\widetilde{W}'_\ell(\mathbf{Y}), \widetilde{W}'_r(\mathbf{Y}), g(\mathbf{Y})$ evaluated at $(r_1, \ldots, r_n)$, for each $i \in T$. Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Coin}}$ and sends 3 random coefficients to $\mathcal{A}$. After that, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Commit}}$, where $\mathcal{S}$ receives MAC key openings from $\mathcal{A}$ and at the same time, informs $\mathcal{A}$ that honest parties' MAC key openings are received. Finally, $\mathcal{S}$ sends MAC key openings of honest parties to $\mathcal{A}$. $\mathcal{S}$ checks whether these openings are correct since $\mathcal{S}$ knows the actual shares of corrupted parties. $\mathcal{S}$ rejects if any of the above checks fails.
8. $\mathcal{S}$ rejects as long as $\boldsymbol{\epsilon} \neq 0$.

We show an inductive analysis on privacy and soundness, which is essentially a distributed analogue of soundness analysis of the zero-knowledge Sumcheck protocol [56]. Recall that $[\![\boldsymbol{W}_\ell]\!], [\![\boldsymbol{W}_r]\!], [\![\boldsymbol{W}_o^*]\!], [\![\boldsymbol{g}]\!]$ satisfy $\boldsymbol{W}_o^* = \boldsymbol{W}_o + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is either zero or non-zero, $\boldsymbol{W}_\ell * \boldsymbol{W}_r = \boldsymbol{W}_o$, and $\boldsymbol{g}$ specifies an n-variate polynomial $g(\mathbf{Y})$ (of degree 3 on variables $Y_1, \ldots, Y_{n-1}$, and degree 5 on variable Y_n). We analyze $\mathcal{A}$'s view in the real execution as follows:

First, since $\boldsymbol{W}_o^*, g(\mathbf{Y})$ are authenticated, $\mathcal{A}$ can open $\widetilde{W}_o^*(z) + G$ to another value with advantage at most $\frac{1}{|\mathbb{F}|}$. As $\mathcal{S}$ will reject an incorrect opening, this leads to $\mathcal{A}$'s indistinguishability advantage at most $\frac{1}{|\mathbb{F}|}$. Moreover, G is a random mask and therefore $\widetilde{W}_o^*(z) + G$ is uniformly random.

Next, since if $\boldsymbol{W}_o^* \neq \boldsymbol{W}_o$, then $\widetilde{W}_o^*(z) \neq \widetilde{W}_o(z)$ with $z \xleftarrow{\$} \mathbb{F}^n$ happens with probability at most $\frac{n}{|\mathbb{F}|}$, by the Schwartz-Zippel Lemma. If $\widetilde{W}_o^*(z) = \widetilde{W}_o(z)$, then $\mathcal{A}$ can pass all the subsequent checks of honest parties. Note that $\widetilde{W}_o^*(z) = \widetilde{W}_o(z)$ is equivalent to $\widetilde{\epsilon}(z) = 0$, which can be detected by $\mathcal{A}$. Since $\mathcal{S}$ will finally reject as long as $\boldsymbol{\epsilon} \neq 0$, this leads to $\mathcal{A}$'s indistinguishability advantage at most $\frac{n}{|\mathbb{F}|}$.

Now, we can start with assuming $\widetilde{W}_o^*(z) \neq \widetilde{W}_o(z)$. Essentially, $\mathcal{A}$ wants to convince honest parties a wrong claim that $\widetilde{W}_o^*(z) = \sum_{\mathbf{Y} \in \{0,1\}^n} \boldsymbol{\chi}_z(\mathbf{Y}) \cdot \widetilde{W}_\ell(\mathbf{Y}) \widetilde{W}_r(\mathbf{Y})$. We briefly recall notations used here. Let $h(\mathbf{Y}) = \widetilde{W}_\ell(\mathbf{Y}) \widetilde{W}_r(\mathbf{Y})$, $f_z(\mathbf{Y}) = \boldsymbol{\chi}_z(\mathbf{Y}) \cdot \widetilde{W}_\ell(\mathbf{Y}) \widetilde{W}_r(\mathbf{Y})$ and $f_z^j(Y_j) = \sum_{b_{j+1}, \ldots, b_n \in \{0,1\}} (f_z + g)(\widehat{Y}_j)$, where $\widehat{Y}_j = (r_1 \ldots r_{j-1}, Y_j, b_{j+1} \ldots b_n)$. Besides, $V_v^j = (r_1 \ldots r_{j-1}, v, b_{j+1} \ldots b_n)$.

Round 1. Induction hypothesis: $\widetilde{W}_o^*(z) \neq \widetilde{W}_o(z)$.

Strategy of $\mathcal{A}$. Recall that all parties will reconstruct a polynomial $f_z^{*1}(Y_1)$, and honest parties will reject in the end if $f_z^{*1}(0) + f_z^{*1}(1) \neq \widetilde{W}_o^*(z) + G$. We

first examine $\mathcal{A}$'s strategies to satisfy this condition. Basically, $\mathcal{A}$ can open an arbitrary $f_z^{*1}(Y_1)$ such that $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o^*(z) + G$. In general, there are three cases.

Case 1. If $\mathcal{A}$ proceeds the protocol honestly. Then the parties would reconstruct a degree-3 polynomial $f_z^{*1}(Y_1)$ such that

$$f_z^{*1}(Y_1) = \sum_{\omega \in \{0,1\}^{n-1}} \boldsymbol{\chi}_z(\widehat{Y}_1)\big(h_\omega^1(Y_1) + h_{\epsilon,\omega}^1(Y_1)\big) + g(\widehat{Y}_1) = f_z^1(Y_1) + f_\epsilon^1(Y_1),$$

where $h_{\epsilon,\omega}^1(Y_1)$ is a degree-2 polynomial with $h_{\epsilon,\omega}^1(v) = \widetilde{\epsilon}(V_v^1)$ for $v \in \{0,1\}$ and $h_{\epsilon,\omega}^1(-1) = 0$. Note that

$$\sum_{v\in\{0,1\}} f_\epsilon^1(v) = \sum_{v\in\{0,1\}} \sum_{\omega\in\{0,1\}^{n-1}} \boldsymbol{\chi}_z(V_v^1) h_{\epsilon,\omega}^1(v) = \sum_{\mathbf{Y}\in\{0,1\}^n} \boldsymbol{\chi}_z(\mathbf{Y})\widetilde{\epsilon}(\mathbf{Y}) = \widetilde{\epsilon}(z).$$

Hence, $f_z^{*1}(0) + f_z^{*1}(1) = f_z^1(0) + f_\epsilon^1(0) + f_z^1(1) + f_\epsilon^1(1) = \widetilde{W}_o(x) + G + f_\epsilon^1(0) + f_\epsilon^1(1) = \widetilde{W}_o(x) + G + \widetilde{\epsilon}(z) = \widetilde{W}_o^*(z) + G$. This means that $\mathcal{A}$ can pass the check $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o^*(z) + G$ without tampering.

Case 2. If $\mathcal{A}$ sends incorrect Beaver triple openings. Recall that Beaver triples are used to compute additive shares of $h_\omega^1(-1) := \widetilde{W}_\ell(V_{-1}^1)\widetilde{W}_r(V_{-1}^1)$ in the honest execution, for each $\omega := (b_2, \ldots, b_n) \in \{0,1\}^{n-1}$. W.l.o.g., suppose for some $\omega := (b_2, \ldots, b_n) \in \{0,1\}^{n-1}$, there are additive errors $\delta_{\ell,\omega}, \delta_{r,\omega}$ introduced by $\mathcal{A}$, and $([a], [b], [c])$ is the Beaver triple used. Then

$$\begin{aligned}
\big[h_\omega^{*1}(-1)\big] &= [a]\,(\widetilde{W}_r(V_{-1}^1) - b + \delta_{r,\omega}) + [b]\,(\widetilde{W}_\ell(V_{-1}^1) - a + \delta_{\ell,\omega}) + [c] \\
&\quad + (\widetilde{W}_\ell(V_{-1}^1) - a + \delta_{\ell,\omega})(\widetilde{W}_r(V_{-1}^1) - b + \delta_{r,\omega}) \\
&= \big[h_\omega^1(V_{-1}^1)\big] + \underbrace{\delta_{\ell,\omega}\left[\widetilde{W}_r(V_{-1}^1)\right] + \delta_{r,\omega}\left[\widetilde{W}_\ell(V_{-1}^1)\right] + \delta_{\ell,\omega}\delta_{r,\omega}}_{\left[h_{\delta_{1,\omega}}^1(-1)\right]}
\end{aligned}$$

Since the resulting additive error $h_{\delta_1,\omega}^1(-1)$ depends on the shared value, it may leak information about $\mathbf{W}_\ell, \mathbf{W}_r$ via observing whether $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o^*(z) + G$. We show that the check $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o^*(z) + G$ always passes no matter how $\mathcal{A}$ cheats in opening Beaver triples. Intuitively, errors introduced at evaluation point -1 make no difference on evaluations at points $v \in \{0,1\}$.

$$\begin{aligned}
f_z^{*1}(Y_1) &= \sum_{\omega\in\{0,1\}^{n-1}} \boldsymbol{\chi}_z(\widehat{Y}_1)\big(h_\omega^1(Y_1) + h_{\epsilon,\omega}^1(Y_1) + h_{\delta_1,\omega}^1(Y_1)\big) + g(\widehat{Y}_1) \\
&= f_z^1(Y_1) + f_\epsilon^1(Y_1) + f_{\delta_1}^1(Y_1),
\end{aligned}$$

where $h_{\delta_1,\omega}^1(Y_1)$ is a degree-2 polynomial with $h_{\delta_1,\omega}^1(v) = 0$, for $v \in \{0,1\}$ and $h_{\delta_1,\omega}^1(-1)$ defined as above. This implies that $f_{\delta_1}^1(Y_1) = \sum_{\omega\in\{0,1\}^{n-1}} \boldsymbol{\chi}_z(\widehat{Y}_1) h_{\delta_1,\omega}^1(Y_1)$

evaluates to 0 at $\{0,1\}$. Finally they would reconstruct a degree-3 polynomial $f_z^{*1}(Y_1) = f_z^1(Y_1) + f_\epsilon^1(Y_1) + f_{\delta_1}^1(Y_1)$, such that $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o(z) + G + f_\epsilon^1(0) + f_\epsilon^1(1) + f_{\delta_1}^1(0) + f_{\delta_1}^1(1) = \widetilde{W}_o^*(z) + G$. Therefore, $\mathcal{A}$ can always pass the check $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o^*(z) + G$ in this case.

Case 3. If $\mathcal{A}$ sends incorrect Beaver triple openings and incorrect openings of $f_z^{*1}(Y_1)$. Then they would finally reconstruct a degree-3 polynomial $f_z^{*1}(Y_1) = \big(f_z^1(Y_1) + f_\epsilon^1(Y_1) + f_{\delta_1}^1(Y_1)\big) + f_{\delta_2}^1(Y_1)$, where $f_{\delta_2}^1(Y_1)$ is of degree ≤ 3 and only depends on additive errors introduced by $\mathcal{A}$. Now $f_z^{*1}(0) + f_z^{*1}(1) = \widetilde{W}_o(z) + G + f_\epsilon^1(0) + f_\epsilon^1(1) + f_{\delta_1}^1(0) + f_{\delta_1}^1(1) + f_{\delta_2}^1(0) + f_{\delta_2}^1(1) = \widetilde{W}_o^*(z) + G + f_{\delta_2}^1(0) + f_{\delta_2}^1(1)$. Therefore, $\mathcal{A}$ needs to select $f_{\delta_2}^1(Y_1)$ such that $f_{\delta_2}^1(0) + f_{\delta_2}^1(1) = 0$.

View of $\mathcal{A}$. We claim that $\mathcal{A}$ learns zero knowledge in round 1. Generally, $\mathcal{A}$ receives i) Beaver triple openings, ii) honest parties' shares of $f_z^{*1}(Y_1)$, iii) r_1. It is straightforward to see that Beaver triple openings and r_1 are uniformly random. Note that the final reconstructed polynomial $f_z^{*1}(Y_1)$ is masked by $g^1(Y_1) = \sum_{b_2,\ldots,b_n} g(\widehat{Y}_1)$, which is a random polynomial[10] of degree 3. Therefore, in all the above three cases, $f_z^{*1}(Y_1)$ as well as its shares of honest parties are uniformly random.

Advantage of $\mathcal{A}$ in round 1. Here we rely on the induction hypothesis that $\widetilde{W}_o^*(z) \neq \widetilde{W}_o(z)$. Consider the final reconstructed polynomial $f_z^{*1}(Y_1)$.

- If $f_z^{*1}(Y_1) = f_z^1(Y_1)$, then $\widetilde{W}_o^*(z) + G = f_z^{*1}(0) + f_z^{*1}(1) = f_z^1(0) + f_z^1(1) = \widetilde{W}_o(z) + G$, which implies a contradiction.
- If $f_z^{*1}(Y_1) \neq f_z^1(Y_1)$, then $f_z^{*1}(r_1) = f_z^1(r_1)$ happens with probability at most $\frac{3}{|\mathbb{F}|}$, for a random $r_1 \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma. Note that $f_z^{*1}(r_1) = f_z^1(r_1)$ is equivalent to $f_\epsilon^1(r_1) + f_{\delta_1}^1(r_1) + f_{\delta_2}^1(r_1) = 0$. There are two conditions depending on whether $\mathcal{A}$ cheats on Beaver triple openings.
  - If $\mathcal{A}$ sends Beaver triple openings honestly, then $f_{\delta_1}^1(Y_1) = 0$. Since $f_{\delta_2}^1(r_1)$ and $f_\epsilon^1(r_1)$ are determined by r_1 and additive errors introduced by $\mathcal{A}$, $\mathcal{A}$ can know whether $f_z^{*1}(r_1) = f_z^1(r_1)$ happens. Then in the following rounds $j \in [2,n]$, $\mathcal{A}$ can open $f_z^{*j}(Y_j)$ to the correct $f_z^j(Y_j)$ (jumping ahead, by setting $f_{\delta_1}^j(Y_j) = 0$ and $f_{\delta_2}^j(Y_j) := -f_\epsilon^j(Y_j)$), and will pass the checks of honest parties. This means $\mathcal{A}$ succeeds in fooling honest parties and leads to $\mathcal{A}'s$ indistinguishability advantage at most $\frac{3}{|\mathbb{F}|}$.
  - If $\mathcal{A}$ sends incorrect Beaver triple openings, then probably $f_{\delta_1}^1(Y_1) \neq 0$ (since it depends on $\mathbf{W}_\ell, \mathbf{W}_r$), and $\mathcal{A}$ cannot learn whether he succeeds during the protocol execution and determine the cheating strategy. This obviously is a worse strategy for $\mathcal{A}$ and there is no need to give a tight bound of $\mathcal{A}$'s advantage in this case, since any degree-3 polynomial vanishes at a random point with probability at most $\frac{3}{|\mathbb{F}|}$.

[10] Indeed, $g(\mathbf{Y})$ is referred as the masking polynomial in the literature. We argue the entropy of g is sufficient for all rounds at the very end of this proof.

Thus we obtain the induction hypothesis for the next round, i.e., $f_z^{*1}(r_1) \neq f_z^1(r_1)$.

Remark D.1. We argue that there is no leakage of information if $\mathcal{A}$ has sent incorrect Beaver triple openings and honest parties finally accepts. Jumping ahead, cheating on Beaver triple openings will not leads to $f_z^{*(j-1)}(r_{j-1}) \neq f_z^{*j}(0) + f_z^{*j}(1)$ in some round j (this claim will be soon argued, which is similar to the $j = 1$ case). Hence, such events only mean that the random polynomial $f_z'^{*(n)}(Y_n)$ reconstructed in the last round, evaluates to a random value $(\boldsymbol{\chi}_z \widetilde{W}'_\ell \widetilde{W}'_r + g)(r_1, \ldots, r_n)$ at a random point r_n, which happens with probability at most $\frac{3}{|\mathbb{F}|}$.

Round $j \in [2, n-1]$**.** Induction hypothesis: $f_z^{*(j-1)}(r_{j-1}) \neq f_z^{(j-1)}(r_{j-1})$.

Strategy of $\mathcal{A}$. We argue similarly to that of round 1. Here, all parties will reconstruct a polynomial $f_z^{*j}(Y_j)$, and honest parties will reject in the end if $f_z^{*j}(0) + f_z^{*j}(1) \neq f_z^{*(j-1)}(r_{j-1})$. There are three cases as well.

Case 1. If $\mathcal{A}$ proceeds the protocol honestly. Then they will reconstruct

$$f_z^{*j}(Y_j) = \sum_{\omega \in \{0,1\}^{n-j}} \boldsymbol{\chi}_z(\widehat{Y}_j)\big(h_\omega^j(Y_j) + h_{\epsilon,\omega}^j(Y_j)\big) + g(\widehat{Y}_j) = f_z^j(Y_j) + f_\epsilon^j(Y_j),$$

where $h_{\epsilon,\omega}^j(v) = h_{\epsilon,v\|\omega}^{j-1}(r_{j-1})$ for $v \in \{0,1\}$ and $h_{\epsilon,\omega}^j(-1) = 0$. Note that

$$f_\epsilon^j(Y_j) = \sum_{\omega \in \{0,1\}^{n-j}} \boldsymbol{\chi}_z(\widehat{Y}_j) h_{\epsilon,\omega}^j(Y_j)$$

is determine by the initial errors $\boldsymbol{\epsilon}$ and randomness $r_1, \ldots, r_{j-1}$ introduced in the previous rounds.

$$\begin{aligned}\sum_{v \in \{0,1\}} f_\epsilon^j(v) &= \sum_{v \in \{0,1\}} \sum_{\omega \in \{0,1\}^{n-j}} \boldsymbol{\chi}_z(V_v^j) h_{\epsilon,\omega}^j(v) \\ &= \sum_{\omega' \in \{0,1\}^{n-j+1}} \boldsymbol{\chi}_z(V_{r_{j-1}}^{j-1}) h_{\epsilon,\omega'}^{j-1}(r_{j-1}) = f_\epsilon^{j-1}(r_{j-1}).\end{aligned}$$

Hence,

$$\begin{aligned}f_z^{*j}(0) + f_z^{*j}(1) &= f_z^j(0) + f_\epsilon^j(0) + f_z^j(1) + f_\epsilon^j(1) \\ &= f_z^{j-1}(r_{j-1}) + f_\epsilon^{j-1}(r_{j-1}) = f_z^{*(j-1)}(r_{j-1}).\end{aligned}$$

This means that $\mathcal{A}$ can pass the check $f_z^{*j}(0) + f_z^{*j}(1) = f_z^{*(j-1)}(r_{j-1})$ without tampering.

Case 2. Similarly, if $\mathcal{A}$ sends incorrect Beaver triple openings. Then there would be an additive error $h_{\delta_1,\omega}^j(-1)$ for some $\omega := (b_{j+1}, \ldots, b_n) \in \{0,1\}^{n-j}$. The reconstructed polynomial $f_z^{*j}(Y_j)$ will have the form

$$\begin{aligned}f_z^{*j}(Y_j) &= \sum_{\omega \in \{0,1\}^{n-j}} \boldsymbol{\chi}_z(\widehat{Y}_j)\big(h_\omega^j(Y_j) + h_{\epsilon,\omega}^j(Y_j) + h_{\delta_1,\omega}^j(Y_j)\big) + g(\widehat{Y}_j) \\ &= f_z^j(Y_j) + f_\epsilon^j(Y_j) + f_{\delta_1}^j(Y_j),\end{aligned}$$

where $h^j_{\delta_1,\omega}(Y_j)$ is a degree-2 polynomial with $h^j_{\delta_1,\omega}(v) = 0$, for $v \in \{0,1\}$ and $h^j_{\delta_1,\omega}(-1)$ defined as above. This implies that $f^j_{\delta_1}(Y_j) = \sum_{\omega\in\{0,1\}^{n-j}} \boldsymbol{\chi}_z(\widehat{Y}_j) h^j_{\delta_1,\omega}(Y_j)$ evaluates to 0 at $\{0,1\}$. Hence, $\mathcal{A}$ can always pass the check $f^{*j}_z(0) + f^{*j}_z(1) = f^{*(j-1)}_z(r_{j-1})$ in this case.

Case 3. Similarly, if $\mathcal{A}$ sends incorrect Beaver triple openings and incorrect openings of $f^{*j}_z(Y_j)$. Then they would finally reconstruct a degree-3 polynomial $f^{*j}_z(Y_j) = \big(f^j_z(Y_j) + f^j_\epsilon(Y_j) + f^1_{\delta_1}(Y_j)\big) + f^j_{\delta_2}(Y_j)$, where $f^j_{\delta_2}(Y_j)$ is of degree ≤ 3 and only depends on additive errors introduced by $\mathcal{A}$. Now $f^{*j}_z(0) + f^{*j}_z(1) = f^j_z(0) + f^j_z(1) + f^j_\epsilon(0) + f^j_\epsilon(1) + f^j_{\delta_1}(0) + f^j_{\delta_1}(1) + f^j_{\delta_2}(0) + f^j_{\delta_2}(1) = f^{*(j-1)}_z(r_{j-1}) + f^j_{\delta_2}(0) + f^j_{\delta_2}(1)$. Therefore, $\mathcal{A}$ needs to select $f^j_{\delta_2}(Y_j)$ such that $f^j_{\delta_2}(0) + f^j_{\delta_2}(1) = 0$.

View of $\mathcal{A}$. By a very similar analyses to that of round 1, we claim that $\mathcal{A}$ learns zero knowledge in round j.

Advantage of $\mathcal{A}$ in round j. Here we rely on the induction hypothesis that $f^{*(j-1)}_z(r_{j-1}) \neq f^{(j-1)}_z(r_{j-1})$. Consider the final reconstructed polynomial $f^{*j}_z(Y_j)$.

- If $f^{*j}_z(Y_j) = f^j_z(Y_j)$, then $f^{*(j-1)}_z(r_{j-1}) = f^{*j}_z(0) + f^{*j}_z(1) = f^j_z(0) + f^j_z(1) = f^{(j-1)}_z(r_{j-1})$, which implies a contradiction.
- If $f^{*j}_z(Y_j) \neq f^j_z(Y_j)$, then $f^{*j}_z(r_j) = f^j_z(r_j)$ happens with probability at most $\frac{3}{|\mathbb{F}|}$, for a random $r_j \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma. Note that $f^{*j}_z(r_j) = f^j_z(r_j)$ is equivalent to $f^j_\epsilon(r_j) + f^j_{\delta_1}(r_j) + f^j_{\delta_2}(r_j) = 0$. Besides, $\mathcal{A}$ is suggested not to cheat on Beaver triple openings, so that $\mathcal{A}$ can learn whether $f^{*j}_z(r_j) = f^j_z(r_j)$ happens. Then in the following rounds, $j+1,\ldots,n$, $\mathcal{A}$ can open the polynomials $f^{*(j+1)}_z(Y_{j+1}),\ldots,f'^{*n}_z(Y_n)$ to the correct polynomials $f^{j+1}_z(Y_{j+1}),\ldots,f'^{n}_z(Y_n)$ and succeeds in fooling honest parties. This leads to $\mathcal{A}$'s indistinguishability advantage at most $\frac{3}{|\mathbb{F}|}$.

Round n. Induction hypothesis: $f^{*(n-1)}_z(r_{n-1}) \neq f^{(n-1)}_z(r_{n-1})$. Here we also take checks after round n into consideration. Actually the case of round n only slightly differs in analyzing $\mathcal{A}$'s advantage and view.

Advantage of $\mathcal{A}$ in round n. Let $f'^{*n}_z(Y_n)$ be the degree-5 polynomial reconstructed in the final round. Note that $f'^{n}_z(Y_n)$ satisfies $f^{(n-1)}_z(r_{n-1}) = f'^{(n-1)}_z(r_{n-1}) = f'^{n}_z(0) + f'^{n}_z(1)$.

- If $f'^{*n}_z(Y_n) = f'^{n}_z(Y_n)$, then $f^{*(n-1)}_z(r_{n-1}) = f'^{*n}_z(0) + f'^{*n}_z(1) = f'^{n}_z(0) + f'^{n}_z(1) = f^{(n-1)}_z(r_{n-1}) = f^{(n-1)}_z(r_{n-1})$, which implies a contradiction.
- If $f'^{*n}_z(Y_n) \neq f'^{n}_z(Y_n)$. In the end, the parties securely open $\Big[\!\Big[\widetilde{W}'_\ell(r_1,\ldots,r_n)\Big]\!\Big] := \Big[\!\Big[\widetilde{W}_\ell(r_1,\ldots,r_n)\Big]\!\Big] + [\![a]\!]\, r_n(1-r_n)$, $\Big[\!\Big[\widetilde{W}'_r(r_1,\ldots,r_n)\Big]\!\Big] := \Big[\!\Big[\widetilde{W}_r(r_1,\ldots,r_n)\Big]\!\Big] + [\![b]\!]\, r_n(1-r_n)$, and $[\![g(r_1,\ldots,r_n)]\!]$. Since $\widetilde{W}_\ell(\mathbf{Y}), \widetilde{W}_r(\mathbf{Y}), a, b, g(\mathbf{Y})$ are authenticated, $\mathcal{A}$ can open them into some other values with advantage at most

$\frac{2}{|\mathbb{F}|}$ (as we apply a batched MAC check). Assume correct openings, then all parties locally compute

$$(f'_z + g)(r_1, \ldots, r_n) := (\boldsymbol{\chi}_z \widetilde{W}'_\ell \cdot \widetilde{W}'_r + g)(r_1, \ldots, r_n),$$

and check

$$f'^{*n}_z(r_n) \stackrel{?}{=} (f'_z + g)(r_1, \ldots, r_n) = f'^{n}_z(r_n),$$

which happens with probability at most $\frac{5}{|\mathbb{F}|}$, for a random $r_n \xleftarrow{\$} \mathbb{F}$, by Schwartz-Zippel Lemma. This leads to $\mathcal{A}$'s indistinguishability advantage at most $\frac{5}{|\mathbb{F}|}$, since in the end $\mathcal{S}$ always rejects as long as $\boldsymbol{\epsilon} \neq 0$.

Total advantage of $\mathcal{A}$. With discussions as above, if $\boldsymbol{\epsilon} \neq 0$, then by a union bound, $\mathcal{A}$ is able to distinguish the view in the ideal world and that in the real execution with advantage at most $\frac{4n+4}{|\mathbb{F}|}$. If $\boldsymbol{\epsilon} = 0$, $\mathcal{A}$'s advantage is upper bounded by $\frac{2}{|\mathbb{F}|}$ (in this case, $\mathcal{S}$ differs from honest parties only in checking validity of MAC openings).

Total view of $\mathcal{A}$. We claim that in the end of the protocol, $\mathcal{A}$ learns no information about $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o$ beyond the fact that $\mathbf{W}_\ell * \mathbf{W}_r = \mathbf{W}_o$. Recall that $f'_z(\mathbf{Y}) = (\boldsymbol{\chi}_z \widetilde{W}'_\ell \widetilde{W}'_r + g)(\mathbf{Y})$, where g has n variables and $3n+3$ uniformly random coefficients. At the beginning, $\widetilde{W}_o(z) + g(z)$ is reconstructed. Then in each round $j \in [1, n-1]$, a degree-3 polynomial $f'^j_z(Y_j) = f'^j_z(Y_j)$ is reconstructed, which essentially reveals 4 independent linear combinations of coefficients of $(f'_z + g)(\mathbf{Y})$. In round $j = n$, a degree-5 polynomial $f'^n_z(Y_n)$ is reconstructed, and reveals 6 independent linear combinations of coefficients of $(f'_z + g)(\mathbf{Y})$. Finally $(f'_z + g)(r_1, \ldots, r_n)$ is computed from revealing $\widetilde{W}'_\ell(r_1, \ldots, r_n) := \widetilde{W}_\ell(r_1, \ldots, r_n) + ar_n(1-r_n)$, $\widetilde{W}'_r(r_1, \ldots, r_n) := \widetilde{W}_r(r_1, \ldots, r_n) + br_n(1-r_n)$, and $g(r_1, \ldots, r_n)$. Note that $\widetilde{W}'_\ell(r_1, \ldots, r_n), \widetilde{W}'_r(r_1, \ldots, r_n)$ are uniformly random by the randomness of a, b. Hence, it remains to consider the leakage on $(f'_z + g)(\mathbf{Y})$. Note that the above linear combinations are conditioned on $f'^{(j-1)}_z(r_{j-1}) = f'^j_z(0) + f'^j_z(1)$ between round $j-1$ and j (for $j = 1$, $\widetilde{W}_o(z) + G = f'^1_z(0) + f'^1_z(1)$, and $f'^n_z(r_n) = (f'_z + g)(r_1, \ldots, r_n)$). Hence, there are total $\big(1 + 4(n-1) + 6 + 1\big) - (n+1) = 3n+3$ independent linear constraints on $(f'_z + g)(\mathbf{Y})$, and a masking polynomial g with $3n+3$ random coefficients is sufficient. Similarly, shares of the reconstructed values are uniformly random due to the randomness of shares of $g(\mathbf{Y})$. □

E Concrete Complexity Analyses

In this section, we show the concrete procedure to compute the complexity results of our approach and the best known approach [36] in honest majority.

E.1 Concrete Computation Complexity of Distributed Sumcheck

Following the estimation of [11,12], we focus on counting the multiplication operations involved in our distributed Sumcheck protocol Π_{Vrfy}.

In a bird's eye view, for each round $j \in [1, n]$, each party's computational costs are dominated by two parts: (i) compute bookkeeping tables $\mathbf{A}^j_{\boldsymbol{\chi}}, \left[\mathbf{A}^j_\ell\right]_t, \left[\mathbf{A}^j_r\right]_t$, (ii) compute $\left[f^j_z(Y_j)\right]_{2t}$ given access to bookkeeping tables.

Part One. Consider the costs of computing $\left[\mathbf{A}^j_\ell\right]_t$ from $\left[\mathbf{A}^{j-1}_\ell\right]_t$. Note that $\left[\mathbf{A}^j_\ell\right]_t$ has 2^{n-j+1} entries and the value respect to entry $\omega \in \{0,1\}^{n-j+1}$ equals to

$$(\left[\mathbf{A}^{j-1}_\ell[1\|\omega]\right]_t - \left[\mathbf{A}^{j-1}_\ell[0\|\omega]\right]_t) \cdot r_{j-1} + \left[\mathbf{A}^{j-1}_\ell[0\|\omega]\right]_t,$$

which requires only one multiplication. Hence, computing the three table for round $j \in [2, n]$ requires $3 \cdot 2^{n-j+1}$ multiplications.

Note that $\left[\mathbf{A}^1_\ell\right]_t, \left[\mathbf{A}^1_r\right]_t$ are actually the inputs, and no computations are needed. We introduce a fast algorithm for computing $\mathbf{A}^1_{\boldsymbol{\chi}}$. Assume $n = 2^s$ and let $\otimes$ denote the tensor product. The main observation is that $\mathbf{A}^1_{\boldsymbol{\chi}}$ of 2^{2^s} entries equals to the tensor product of two sub-tables of size $2^{2^{s-1}}$. For instance, if $s = 1$, $\mathbf{A}^1_{\boldsymbol{\chi}}$ contains $4 = 2^{2^1}$ values,

$$(1 - z_0, z_0) \otimes (1 - z_1, z_1).$$

If $s = 2$, $\mathbf{A}^1_{\boldsymbol{\chi}}$ contains $16 = 2^{2^2}$ values,

$$\Big((1 - z_0, z_0) \otimes (1 - z_1, z_1)\Big) \otimes \Big((1 - z_2, z_2) \otimes (1 - z_3, z_3)\Big).$$

This implies that $\mathbf{A}^1_{\boldsymbol{\chi}}$ of 2^{2^s} entries can be inductively computed, following a binary-tree structure. The computational costs are $\sum_{i=1}^{s} 2^{2^i} \cdot 2^{s-i} = \sum_{i=1}^{s} N^{1/2^{i-1}} \cdot 2^{i-1} = N + O(\sqrt{N})$ multiplications. In addition, given access to $\mathbf{A}^1_{\boldsymbol{\chi}}$, $\left[\widetilde{W}_o(z)\right]_t = \sum_{\omega \in \{0,1\}^n} \boldsymbol{\chi}_z(\omega) \cdot [W_o(\omega)]_t$ can be computed in $N = 2^n$ multiplications.

Part Two. Recall that the parties compute sharings of four evaluations of

$$f^j_z(Y_j) = \sum_{b_{j+1},\ldots,b_n} (\boldsymbol{\chi}_z \cdot \widetilde{W}_\ell \cdot \widetilde{W}_r + g)(r_1, \ldots, r_{j-1}, Y_j, b_{j+1}, \ldots, b_n)$$

at $\{0, \pm 1, 2\}$. For simplicity, we omit the costs for evaluating $g(\mathbf{Y})$[11], since $g(\mathbf{Y})$ has only $O(\log N)$ monomials. First, as $\boldsymbol{\chi}_z, \widetilde{W}_\ell, \widetilde{W}_r$ are multilinear, computing their evaluations at $-1, 2$ from evaluations at $0, 1$ only involves *additions*. For instance, let V^j_v denote $(r_1, \ldots, r_{j-1}, v, b_{j+1}, \ldots, b_n)$,

$$\boldsymbol{\chi}_z(V^j_{-1}) = 2\boldsymbol{\chi}_z(V^j_0) - \boldsymbol{\chi}_z(V^j_1) = \boldsymbol{\chi}_z(V^j_0) + \boldsymbol{\chi}_z(V^j_0) - \boldsymbol{\chi}_z(V^j_1).$$

Since for each $(b_{j+1}, \ldots, b_n) \in \{0,1\}^{n-j}$, 4 evaluations need to be computed and each involves 2 multiplications, a direct calculation takes $8 \cdot 2^{n-j}$ multiplications

[11] In fact, we also omit the verification costs for each round and the additional costs for round n, since they are constant.

for round j. We show that the required multiplications can be reduced to $5 \cdot 2^{n-j}$ for round $j \in [2, n]$ and $3 \cdot 2^{n-1}$ for round 1.

The saving for round $j \in [2, n]$ is due to

$$(\widetilde{W}_\ell \cdot \widetilde{W}_r)(V_2^j) = 3(\widetilde{W}_\ell \cdot \widetilde{W}_r)(V_1^j) - 3(\widetilde{W}_\ell \cdot \widetilde{W}_r)(V_0^j) + (\widetilde{W}_\ell \cdot \widetilde{W}_r)(V_{-1}^j),$$

which only involves additions. The additional savings for round 1 is due to that $\left[(\boldsymbol{\chi}_z \cdot \widetilde{W}_\ell \cdot \widetilde{W}_r)(V_v^1)\right]_{2t}$ can be replaced with $\left[(\boldsymbol{\chi}_z \cdot \widetilde{W}_o)(V_v^1)\right]_t$, for $v \in \{0, 1\}$, which has already been computed in calculating $\left[\widetilde{W}_o(z)\right]_t$.

Putting all pieces together. By the above analyses, each party's computation is dominated by

$$2N + O(\sqrt{N}) + \big(\sum_{j=1}^{n}(3+7) \cdot 2^{n-j}\big) - 4 \cdot 2^{n-1} = 10N + O(\sqrt{N}).$$

Note that the verification costs are $O(\log N)$ computations.

Remark E.1. As for our distributed Sumcheck in dishonest majority, the hidden constant of each party's $O(N)$ computation is slightly higher. The additional computation mainly stems from the use of multiplication triples, more sophisticated calculation of evaluations of $\left[f_z^j(Y_j)\right]$ and the MACs of $\left[\!\left[\widetilde{W}_o(z)\right]\!\right], \left[\!\left[\widetilde{W}_\ell(\boldsymbol{r})\right]\!\right], \left[\!\left[\widetilde{W}_r(\boldsymbol{r})\right]\!\right]$.

E.2 Concrete Complexity of [36]

In this section, we calculate the concrete computation complexity of [36], which presents an verification protocol in the honest majority setting implicitly based on FLIOPs. In particular, we choose the compression factor k of [36] to be 2, so that the underlying FLIOP has optimized linear prover time. Along the way, we also show that [36] is essentially performing Sumcheck.

In an overview, [36]'s approach consists of the following three steps: DeLinearization, DimensionReduction, Randomization. Suppose the goal is to verify $N = 2^n$ multiplications on $\mathbf{W}_\ell, \mathbf{W}_r, \mathbf{W}_o$.

Costs of Step One. DeLinearization transforms the task to the verification of an inner-product of dimension N. Concretely, let $r \xleftarrow{\$} \mathcal{F}_{\mathsf{Coin}}$. Each party computes $r^1, r^2, \ldots, r^{N-1}$, and then $[\boldsymbol{x}]_t := ([W_\ell(0)]_t, r \cdot [W_\ell(1)]_t, \ldots, r^{N-1} \cdot [W_\ell(N-1)]_t)$, $[\boldsymbol{y}]_t := \mathbf{W}_r$, $[z]_t := \sum_{i=0}^{N-1} r^i \cdot [W_o(i)]_t$. This step requires $3N$ multiplications. //Note that [36] essentially applies Sumcheck on $z = \sum_{i=0}^{N-1} \boldsymbol{x}(i)\boldsymbol{y}(i)$.

Costs of Step Two. Informally, DimensionReduction inductively reduces the problem of checking inner-product of dimension N by half, until the resulting inner-product has a constant size, e.g., 2. Hence, we first calculate the costs for reducing dimension N to $N/2$. We denote $[\boldsymbol{x}]_t := (\left[\boldsymbol{x}^{(0)}\right]_t, \left[\boldsymbol{x}^{(1)}\right]_t)$,

$[\boldsymbol{y}]_t := (\left[\boldsymbol{y}^{(0)}\right]_t, \left[\boldsymbol{y}^{(1)}\right]_t)$. Let $\{0, \pm 1\}$ be the evaluation set. //Note that $[\boldsymbol{x}]_t$, $[\boldsymbol{y}]_t$ essentially correspond to the initial bookkeeping tables.

In DimensionReduction, the parties first compute $\left[z^{(j)}\right]_{2t} = \sum_{i=0}^{N/2-1} \left[\boldsymbol{x}^{(j)}(i)\right]_t \cdot \left[\boldsymbol{y}^{(j)}(i)\right]_t$ and then consume one double sharing pair to obtain $\left[z^{(j)}\right]_t$ for $j \in \{0, 1\}$. This procedure requires N multiplications. //Note that $z^{(j)} = \sum_{b_2,\ldots,b_n \in \{0,1\}} (\widetilde{x} \cdot \widetilde{y})(j, b_2, \ldots, b_n) = f_1(0)$, for $j \in \{0, 1\}$.

Then the parties compress $(\left[\boldsymbol{x}^{(0)}\right]_t, \left[\boldsymbol{x}^{(1)}\right]_t)$ into $\left[\boldsymbol{x}^{(r_1)}\right]_t$, where $\boldsymbol{x}^{(r_1)} \in \mathbb{F}^{N/2}$. Suppose $r_1 \xleftarrow{\$} \mathcal{F}_{\mathsf{Coin}}$. Concretely, $\left[\boldsymbol{x}^{(r_1)}(i)\right]_t := (\left[\boldsymbol{x}^{(1)}(i)\right]_t - \left[\boldsymbol{x}^{(0)}(i)\right]_t) \cdot r_1 + \left[\boldsymbol{x}^{(0)}(i)\right]_t$, for each $i \in [1, N/2]$. In addition, $\left[\boldsymbol{y}^{(r_1)}(i)\right]_t$ is computed in a similarly way. This introduces N multiplications per party. //This corresponds to computing bookkeeping tables for the next round.

It remains to compute $\left[z^{(r_1)}\right]_t$. The parties first compute

$$\left[z^{(-1)}\right]_{2t} := \sum_{i=0}^{N/2-1} (2\left[\boldsymbol{x}^{(0)}(i)\right]_t - \left[\boldsymbol{x}^{(1)}(i)\right]_t) \cdot (2\left[\boldsymbol{y}^{(0)}(i)\right]_t - \left[\boldsymbol{y}^{(1)}(i)\right]_t),$$

then they consume one double sharing pair to obtain $\left[z^{(-1)}\right]_t$. //Note that $z^{(-1)} = \sum_{b_2,\ldots,b_n \in \{0,1\}} (\widetilde{x} \cdot \widetilde{y})(-1, b_2, \ldots, b_n) = f_1(-1)$.

Finally, the parties compute

$$\left[z^{(r_1)}\right]_t := (\frac{\left[z^{(1)}\right]_t + \left[z^{(-1)}\right]_t}{2} - \left[z^{(0)}\right]_t) \cdot r_0^2 + \frac{\left[z^{(1)}\right]_t - \left[z^{(-1)}\right]_t}{2} \cdot r_0 + \left[z^{(0)}\right]_t.$$

This introduces $N/2$ multiplications per party. //Note that $z^{(r_1)} = \sum_{b_2,\ldots,b_n \in \{0,1\}} (\widetilde{x} \cdot \widetilde{y})(r_1, b_2, \ldots, b_n) = f_1(r_1)$.

Therefore, reducing dimension N to $N/2$ requires only $5N/2$ multiplications, yielding $5N$ multiplications for a constant dimension 2. //The inner-product relation $([z^{r_1}]_t, \left[\boldsymbol{x}^{(r_1)}\right]_t, \left[\boldsymbol{y}^{(r_1)}\right]_t)$ in [36] is equivalent to $f_2(X_2)$ such that $f_1(r_1) = f_2(0) + f_2(1)$ in Sumcheck.

Costs of Step Three. After $O(\log N)$ iterations of DimensionReduction, the inner-product has a constant dimension, and can be checked directly. The costs of this procedure can be omitted.

Putting all pieces together. Given discussion as above, each party's computation is dominated by $8N = 3N + 5N$ multiplications.

Remark E.2. Note that the $8N$ computation of the above instantiation is less than $10N$ of our distributed Sumcheck approach. The reason is that we use a more structured type of coefficients $(\boldsymbol{\chi}_z(0), \ldots, \boldsymbol{\chi}_z(N-1))$ rather than $(1, \lambda, \ldots, \lambda^{N-1})$ for compressing multiplications into sumcheck. The advantage of the multilinear coefficients is the resulting $\frac{\log N}{|\mathbb{F}|}$ soundness error rather than $\frac{N}{|\mathbb{F}|}$ from the later. Actually, we can employ the reduction of [36] and obtain the same $8N$ computation.