Malicious Security in Collaborative zk-SNARKs: More than Meets the Eye

Sanjam Garg[*] Aarushi Goel[†] Abhishek Jain[‡] Bhaskar Roberts[§] Sruthi Sekar[¶]

Abstract

Collaborative zk-SNARKs (Ozdemir and Boneh, USENIX'22) are a multiparty variant of zk-SNARKs where multiple, mutually distrustful provers, each holding a private input, jointly compute a zk-SNARK using their combined inputs. A sequence of works has proposed efficient constructions of collaborative zk-SNARKs using a common template that involves designing secure multiparty computation (MPC) protocols to emulate a zk-SNARK prover without making non-black-box use of cryptography. To achieve security against malicious adversaries, these works adopt compilers from the MPC literature that transform semi-honest MPC into malicious-secure MPC.

In this work, we revisit this design template.

- **Pitfalls:** We demonstrate two pitfalls in the template, which can lead to a loss of input privacy. We first show that it is possible to compute collaborative proofs on *invalid* witnesses, which in turn can leak the inputs of honest provers. Next, we show that using state-of-the-art malicious security compilers *as-is* for proof computation is insecure, in general. Finally, we discuss mitigation strategies.
- **Malicious Security Essentially for Free:** As our main technical result, we show that in the honest-majority setting, one can forego malicious security checks performed by state-of-the-art malicious security compilers during collaborative proof generation of several widely used zk-SNARKs. In other words, we can avoid the overheads of malicious security compilers, enabling faster proof generation.

To the best of our knowledge, this is the first example of non-trivial computations where semi-honest MPC protocols achieve malicious security. The observations underlying our positive results are general and may have applications beyond collaborative zkSNARKs.

[*]UC Berkeley. Email: sanjamg@berkeley.edu
[†]Purdue University. Email: aarushi@purdue.edu
[‡]NTT Research and Johns Hopkins University. Email: abhishek@cs.jhu.edu
[§]UC Berkeley. Email: bhaskarr@eecs.berkeley.edu
[¶]IIT Bombay. Email: sruthi@cse.iitb.ac.in

Contents

1 Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [Kil92, Mic94, BCC+17] enable a prover to demonstrate the correctness of a large computation without revealing any secrets used in the computation. Recently, Ozdemir and Boneh [OB22] introduced the notion of *collaborative* zk-SNARKs – a multi-party variant of zk-SNARKs where multiple participants, each holding private inputs, collaborate to jointly compute a zk-SNARK that proves the correctness of a computation over their inputs. Collaborative zk-SNARKs enable many applications such as auditable secure multiparty computation (MPC) [BDO14] and verifiable distributed computation of private statistics (e.g., credit scores and healthcare statistics) [OB22].

General Design Template for Collaborative-zk-SNARKs. Ozdemir and Boneh [OB22] presented an efficient approach for constructing collaborative zk-SNARKs based on several widely-used zk-SNARKs, such as Groth16 [Gro16] and Plonk [GWC19], in which the proof generation time is nearly the same as when the proof is computed by a single prover. Building on this approach, subsequent works have presented new constructions either with the aim of achieving even faster proof generation times [GGJ+23, LZW+24b, LZW+24a], or adding support for other zk-SNARKs used in practice [BKb, LZW+24b, LZW+24a].

We now outline the general template for constructing collaborative zk-SNARKs [OB22]. The high level idea is to implement the prover algorithm of the zk-SNARK using MPC [Yao86, CCD88, GMW87, BGW88]. The design proceeds in three steps:

- **Step I (Generate the Extended Witness):** Suppose we want to prove the correctness of some computation described by an arithmetic circuit C. In the first step, the parties run a standard MPC protocol using their private inputs to compute secret shares of the "extended witness", namely, the wire values of the arithmetic circuit evaluated over the inputs.
- **Step II (Compute The Proof)[1]:** Next, the parties run a special-purpose MPC that takes as input the shares of the extended witness and computes a zk-SNARK proof by emulating the zk-SNARK prover algorithm. This MPC is designed to avoid *non-black-box* use of cryptography. Typically, we start by designing an MPC that is only secure against semi-honest adversaries.
- **Step III (Upgrade to Malicious Security)**: Finally, we compile the semi-honest protocol into a protocol that is secure against fully malicious adversaries. For this step, prior works employ state-of-the-art compilers from the MPC literature [DPSZ12, CGH+18, GIP+14, LN17, NV18, FL19, GSZ20, BBC+19, BGIN19, BGIN20, BGIN21, DEN24]. Importantly, these compilers preserve the black-box nature of the underlying custom semi-honest protocol.

1.1 Our Results

We revisit this design template and present the following results:

- **Pitfalls.** We demonstrate several pitfalls of the design template by presenting concrete attacks that violate input privacy of the honest parties. We also provide mitigation strategies.
- **Malicous Security Essentially for Free.** We show that step III of the design template can be omitted for several widely-used zk-SNARKs. More specifically, we show that with minor changes, existing semi-honest MPC protocols for proof computation of Groth16, Bulletproofs, and Plonk are already secure against malicious adversaries; therefore, the malicious security compiler is not needed. Our results hold in the honest majority setting. To the best of our

[1] [OB22] refers to Step II here as collaborative zk-SNARKs. We refer to collaborative zk-SNARKs as the complete process involving the extended witness generation and the proof generation.

knowledge, these are the first examples of non-trivial functionalities where semi-honest MPC protocols achieve malicious security.

We now elaborate on these results.

1.1.1 Results I: Pitfalls

We demonstrate two pitfalls of the existing design template via attacks on input privacy that exploit proofs computed on invalid witnesses, or the reactive nature of proof computation.

Attack #1. The key idea of our first attack is that standard zk-SNARKs do not guarantee input privacy for invalid witnesses, whereas collaborative zk-SNARKs must provide input privacy for *both* valid and invalid witnesses. Due to this mismatch in security, it is possible for an attacker to violate the input privacy of a collaborative zk-SNARK if the proof is computed using an invalid witness.

Let us elaborate. Collaborative zk-SNARKs must satisfy the *t-zero-knowledge* property, which guarantees that a corrupted subset of at most t provers learn nothing about the honest parties' inputs except for a bit indicating whether the combined inputs of all parties constitute a valid witness. Unlike standard zero knowledge that only guarantees security for true statements, t-zero-knowledge guarantees security even when the combined witness is invalid. This stronger guarantee is necessary, since unlike standard zero knowledge where the validity of a witness can be determined locally by an honest prover, the parties in collaborative zk-SNARKs can only determine this jointly.

As part of our attack, we design a distinguisher that receives as input a Groth16 [Gro16] zk-SNARK proof computed on one of two *invalid* witnesses, and correctly identifies which witness was used (thus breaking t-zero-knowledge). We next show how to use this distinguisher to launch two attacks on collaborative zk-SNARK computation of Groth16 proofs:

- *Outsider Attack:* Consider a two-prover collaborative zkSNARK for Groth16 proofs. Suppose that the joint input held by the provers corresponds to one of two possible invalid witnesses. If the provers compute a proof using this witness, then anyone can run the distinguisher algorithm on the final proof to determine the (invalid) witness used by the provers.

This attack can be mitigated by checking whether the witness is invalid during Step I, and aborting if this is the case.

- *Insider Attack:* Now consider the case where one of the provers is dishonest. Let us suppose that the provers hold a valid witness, and the honest prover's input has one of two possible values. We show that the dishonest prover can learn the honest prover's input by modifying the witness to be invalid and then inspecting the resulting proof as follows: The dishonest prover behaves honestly during Step I, and at the end of this step, the provers obtain secret shares of the (valid) extended witness. However, before the start of Step II, the dishonest prover locally modifies its share so that the reconstructed value corresponds to an *invalid* extended witness. This is possible, for example, in the case of *additive* secret sharing.[2] Next, the corrupted prover behaves honestly during step II. Then given the final proof, it runs the distinguisher algorithm to determine the input of the honest prover.

To prevent the second attack, we must ensure *input consistency* between Steps I and II. For instance, this could be achieved using robust secret sharing based on AMD codes [CDF+08]. To the

[2] We note that this does not work in the honest majority setting if the parties obtain a *threshold* secret sharing of the the extended witness. This is because, in threshold secret sharing, if the adversary locally modifies their shares, that will only result in the parties collectively holding an invalid sharing, which in turn will result in the protocol aborting during reconstruction.

best of our understanding, prior work treats the implementation of Steps I and II as independent tasks. The above attack demonstrates that this methodology is not sound.

Attack #2. The state-of-the-art compilers for malicious security in MPC (see section 1.2) are designed for semi-honest protocols that achieve privacy against malicious adversaries up to the output reconstruction round. Most secret-sharing based MPC protocols enjoy this privacy property [GIP+14]. To achieve full malicious security, these compilers perform an aggregate "MAC check" on the computation prior to the output reconstruction round. This prevents an adversary from learning the output if it behaved maliciously earlier in the protocol.

To the best of our understanding, prior work on collaborative zk-SNARKS [OB22, LZW+24b, LZW+24a, BKb, BKa, OB] uses these compilers *as-is* to achieve malicious security. We observe that this approach is not secure, in general. This is because many zk-SNARKs constitute a stateful, *reactive* functionality that produces several outputs. In this setting, a single aggregate MAC check before the release of the final output is insufficient. Specifically, we show a simple reactive functionality where this leads to loss of input privacy. Finally, we describe a simple, fail-safe compiler for achieving malicious security.

1.1.2 Results II: Malicious Security for Free

Surprisingly, we show that for some of the most widely-used zk-SNARKs, the semi-honest MPC for proof generation is already maliciously secure, so step III of the design template (compiling the semi-honest protocol into a maliciously secure protocol) is not required. We now give more details.

- **Groth16 and Bulletproofs:** With a minor change, we can use the existing semi-honest MPC protocols for Groth16 [Gro16] (as described in [OB22]), and Bulletproofs [BBB+18] (as presented in [BKb]). Our change only concerns with the way secret shares of random values are generated in these protocols; see section 2 for details.
- **Plonk:** For the Plonk proof system [GWC19], we can use a specific instantiation of the semi-honest MPC protocol presented in [OB22]. Our instantiation crucially uses Bar-Ilan and Beaver's constant-round protocol [BIB89] for iterated multiplication. We also use the modified randomness generation procedure mentioned above and add a constant-time check to the final round. For more details, see section 2.

Discussion. To the best of our knowledge, our positive results provide the first example of a non-trivial functionality — specifically, zk-SNARK proof generation — that can be securely computed without requiring any additional consistency checks during the computation. Prior to our work, it was unclear whether this was even possible. Since the underlying properties that we leverage are common to many zk-SNARKs, we believe our results could extend to other zk-SNARK constructions as well, presenting an interesting avenue for future research. More broadly, many of our observations about the types of functionalities for which semi-honest protocols remain secure against malicious adversaries are general and not specific to zk-SNARKs, and may therefore be of independent interest with potential for wider applications.

As an example, the primary motivation behind the popular Bar-Ilan–Beaver [BIB89] protocol for iterated multiplication is *efficiency improvement*; namely, achieving a constant-round protocol. We show, surprisingly, that the use of randomness in this protocol not only helps in reducing the multiplication depth, but also in achieving *security against malicious adversaries*. In fact, if one were to instead use a standard multiplication protocol composed iteratively for iterated multiplications, not only would it be less efficient, it would render our Plonk-based collaborative zk-SNARK insecure in the absence of a consistency check. We refer the reader to section 2 for details.

1.2 Related Work

Collaborative zk-SNARKs. Following the introduction of collaborative zk-SNARKs [OB22], several follow-up works [GGJ+23, LZW+24b, LZW+24a] have focused on improving the efficiency of proof generation. In [GGJ+23], the authors introduce and instantiate the notion of zk-SNARKs-as-a-service (zkSaaS) for the same underlying zk-SNARKs as in [OB22] (i.e., Groth16 [Gro16], Plonk [GWC19], and Marlin [CHM+20]), which improves the communication and computation cost of the prover parties. In [LZW+24b, LZW+24a], the authors extend collaborative zk-SNARKs to two new zk-SNARKs, Libra [XZZ+19] and Hyperplonk [CBBZ23], and show that communication and computation costs can be fully distributed among the prover parties. Towards expanding the library of efficient collaborative zk-SNARKs, [BKb] builds collaborative zk-SNARKs based on Bulletproofs [BBB+18].

Malicious Security Compilers in MPC. A long series of work over the last decade has focused on designing efficient compilers that can transform a semi-honest secure MPC to withstand security against malicious adversaries. Several of these works [DPSZ12, CGH+18, GIP+14, LN17, NV18] involve running two (or more)[3] parallel executions of the semi-honest protocol, some recent works [FL19, GSZ20, BBC+19, BGIN19, BGIN20, BGIN21, DEN24] show that maliciously secure MPC can be achieved at nearly the same (amortized) communication cost as semi-honest MPC. [FL19] achieves this by relying on a two-thirds honest majority assumption, while [GSZ20, BGIN19, BGIN20] operate in the standard honest majority setting (i.e., tolerating up to half corruptions) and [BBC+19, BGIN21] in the dishonest majority setting. However, these techniques introduce additional rounds that grow logarithmically with the size of the function, slowing down the protocol. Dalskov et al. [DEN24] recently reduced this round overhead to a constant, but at the cost of increased computation and communication complexity in the preprocessing phase of the MPC protocol.

2 Technical Overview

In this section, we discuss the main ideas underlying the result that malicious security comes essentially for free. For details on the pitfalls and the general compiler for malicious security, we refer the reader to section 4.

2.1 Additive Attack Paradigm [GIP+14]

The starting point of our techniques is the influential additive attack paradigm of [GIP+14], which roughly says that when a malicious adversary participates in a semi-honest-secure MPC protocol, they can only introduce "additive" errors. This idea underlies the design of most state-of-the-art MPC compilers for transforming semi-honest MPC to maliciously-secure MPC.

Typical MPC protocols represent the target function as an arithmetic circuit and evaluate it gate-by-gate over secret-shared inputs. While addition gates can be computed non-interactively, multiplication gates require interaction among parties, providing an opportunity for a *malicious* adversary to influence the computation.

However, Genkin et al. [GIP+14] demonstrated that the scope for such attacks remains quite limited in most secret-sharing-based MPC protocols. For a typical *semi-honest-secure* MPC protocol based on secret sharing in the honest majority setting, if the protocol is executed in the presence of a *malicious adversary*, the following observations are true:

[3] Two repetitions suffice when working over a large field (exponential in the security parameter). This is indeed the case for zk-SNARK computations – the focus of this work. For small fields, these compilers can require running many parallel executions.

1. **Obs 1: Arbitrary Additive Errors:** When multiplying two secret shared values a and b, a malicious adversary can only introduce an arbitrary error term ϵ, such that the parties eventually obtain secret shares of $a \cdot b + \epsilon$ instead of $a \cdot b$. Crucially, while the adversary can choose ϵ freely, it cannot make ϵ dependent on a or b.
2. **Obs 2: Privacy Until Reconstruction:** Moreover, the MPC protocol maintains *privacy* against malicious adversaries, except during the final round when the output is reconstructed.

These observations are formalized by showing that the view of a malicious adversary in all-but-the-last round can be simulated in the ideal world. Moreover, the simulator can extract the value of ϵ.

The above observations hold either when the adversary corrupts *exactly* $n/2 - 1$ parties or if all secret shares of any *random values* used in the protocol are guaranteed to be honestly generated. Throughout this section, we will assume that one of these conditions is met and refer to such MPC protocols as **MPC secure up to additive attacks**.

Design of Maliciously Secure MPC. We can use the above observations to compile semi-honest MPC protocols into maliciously secure protocols. The compiler just has to detect additive errors, and if errors are detected, it aborts the protocol before reconstructing the final output. The way to detect additive errors involves runing two parallel instances of the semi-honest protocol (up until the last round) and verifying consistency between them. This roughly doubles the computational overhead compared to the semi-honest version. This design template was used in all prior works on maliciously secure collaborative zk-SNARKs [OB22, LZW+24b, LZW+24a, BKb, BKa, OB].

We seek to show that it is not necessary to run two parallel copies of the semi-honest protocol for the specific task of generating a zk-SNARK proof. This is because either (1) the additive errors do not allow the adversary to violate input privacy, so there is no need to detect them, or (2) additive errors can be detected much more efficiently by repurposing the proof's verifiability to check for additive errors.

2.2 Malicious Security for Free in Honest Majority: Starting Ideas

In this section, we explore some initial ideas for identifying scenarios where a semi-honest, honest majority MPC protocol (in particular, an MPC secure up to additive attacks) can also remain secure against malicious adversaries. Towards the end, in section 2.2.1 we demonstrate how these initial ideas are sufficient to show that a semi-honest collaborative Groth16 [Gro16] Proof generation is also maliciously secure.

As noted above, the two observations in [GIP+14] hold when either the adversary corrupts exactly $n/2 - 1$ parties or all random values in the protocol are honestly generated. Since we build on their observations, throughout this discussion, we assume one of these conditions are met when referring to a *semi-honest MPC* or an *MPC that is secure up to additive attacks*.

Recap: Malicious Security in MPC. Before delving into our ideas, let us review what it means for an MPC protocol computing function f to be secure against a malicious adversary. In the real-ideal world paradigm, this requires the existence of a polynomial-time simulator – with oracle access to an ideal functionality $\mathcal{F}$ computing the target function f– that can simulate the adversary's view in such a way that the following two distributions are indistinguishable: The joint distribution of the adversary's view and the output of the honest parties in the real world, and the joint distribution of the simulated view and the output of the honest parties computed by the ideal functionality. Here the "output of the honest parties" is included in the joint distribution to ensure correctness of the computed output.

However, since we are working with an MPC where the adversary can inject additive errors during the computation of multiplication gates, achieving correctness in the traditional sense is infeasible. This might suggest that the above definition is too strong to guarantee for any MPC secure

up to additive attacks. However, recall that our primary goal is to establish that collaborative proof generation for various zk-SNARKs using a semi-honest MPC also achieves security against malicious adversaries. Since zk-SNARKs are self-verifying, correctness of the computed proof does not need to be enforced by the MPC itself; instead, correctness can be verified by checking whether the proof successfully verifies.

Thus, we relax the definition of malicious security to allow the adversary to maul the output. Due to the self-verification property of proof systems, an MPC computing a zk-SNARK while achieving this weaker notion of security effectively implies an MPC achieving strong malicious security (i.e., the original definition).

L-Malicious Security. We introduce a relaxed notion of malicious security, which we call L-malicious security. Here, the ideal functionality $\mathcal{F}'$ takes as input the inputs of all corrupt and honest parties (say $\vec{x}$). Additionally, it receives a linear function L from the adversary and sends as output $L(f(\vec{x}))$ to all parties. Note that this relaxation only affects the correctness of output, but not privacy. Allowing the adversary to modify the output via a linear function does not reveal any additional information about the honest parties' inputs beyond what is already implied by the "correct" output of f, thereby preserving privacy. For the remainder of this discussion, we focus on proving that for certain functions, an MPC secure up to additive attacks also satisfies L-malicious security. To do so, we must demonstrate the existence of an appropriate simulator.

How to Simulate? From [GIP+14], we know there exists a simulator, $\mathcal{S}_{<\text{last}}$, that can generate an indistinguishable view for the adversary in all but the final round while extracting the corrupt parties' initial inputs and any injected additive errors. The challenge is extending this simulator to construct $\mathcal{S}^{\mathcal{F}'}$, which must also simulate an indistinguishable view in the final round, where the output is reconstructed. To prove security, we need to show that the joint distribution of the adversary's simulated view and the output computed by $\mathcal{F}'$ is indistinguishable from that in the real protocol.

How can we hope to achieve this? Since the output is reconstructed in the final round, $\mathcal{S}^{\mathcal{F}'}$ must use the ideal functionality $\mathcal{F}'$ to simulate it. However, due to adversarially injected additive errors during multiplication gates, the computation of f may be altered in ways that are not always representable as a linear function of the correct output. Even when such a linear representation exists, the simulator may not have enough information to determine it solely from the extracted inputs and additive errors.

Using this observation as our starting point, we consider two simple functions where adversarial additive errors do result in a final mauled output that is effectively a linear function of the correct output. Let x, y, z be secret shared inputs.

Affine Functions over Secret Shared Inputs: The simplest kind of functions to compute are affine functions of the form $f(x, y) = A \cdot x + B \cdot y + C$, where A, B, C are some public constants. According to the first observation in [GIP+14], in an MPC protocol secure against additive attacks, a malicious adversary can only introduce errors during the multiplication of two secret-shared values. Since an affine function does not involve any such multiplications, the adversary has no opportunity to introduce additive errors when using such an MPC to compute this function.

Then, given $\mathcal{S}_{<\text{last}}$, designing an extended simulator $\mathcal{S}^{\mathcal{F}'}$ that also simulates the output in the final round is straightforward. The simulator can simply query the ideal functionality $\mathcal{F}'$ using the adversary's initially extracted inputs (which can be extracted by $\mathcal{S}_{<\text{last}}$) and setting L to be the identity function. Since L in this case is always the identity function, an MPC secure up to additive attacks is trivially secure against malicious adversaries for such functions.

Degree-Two Computations over Secret Shared Inputs: The next kind of functions that we consider are degree-2 or depth-1 computations, e.g., $f(x, y, z) = x \cdot y + A \cdot z + B$, where A, B are public constants. From the first observation in [GIP+14], we know that when using an MPC secure up to additive attacks to compute f, the adversary can an inject arbitrary additive error ϵ during the com-

putation of $x \cdot y$. In other words, the final output of this computation in the presence of a malicious adversary could be $x \cdot y + A \cdot z + B + \epsilon$.

Given $\mathcal{S}_{<\text{last}}$, we can extend this simulator to simulate the final output. Since $\mathcal{S}^{\mathcal{F}'}$ can use $\mathcal{S}_{<\text{last}}$ to extract the adversary's initial inputs and the injected error ϵ, it can query the ideal functionality $\mathcal{F}'$ on the initially extracted inputs and L such that $L(x) = x + \epsilon$. Finally, it can use the output sent by the ideal functionality to simulate the adversary's view in the last round. Thus, we have shown that an MPC secure up to additive attacks will also be L-maliciously secure for such functions.

While these functions may appear simplistic with no standalone applications, we demonstrate their practical utility in the generation of zk-SNARKs.

Difference Between Standard Malicious Security and t-Zero-Knowledge. Before presenting an example of such a zk-SNARK that can be computed using just affine or degree-2 computations, we first review the definition of malicious security for a collaborative zk-SNARK protocol. As discussed in section 1, [OB22] define a notion of t-zero-knowledge for collaborative zk-SNARKs, ensuring that malicious provers cannot learn the witnesses of honest provers. This is formalized by requiring a simulator that, given a bit indicating whether the honest and corrupt party's joint witness is valid, can simulate an indistinguishable view for the adversary.

Note that, this differs from standard maliciously secure MPC, where the simulator has oracle access to an ideal functionality computing the target function. Here, the simulator only receives information about the validity of the combined witnesses, without access to an ideal functionality computing the proof. This difference is because t-zero-knowledge is designed to capture both zero-knowledge of the proof and malicious security of the MPC used to generate the proof. If the simulator had access to an ideal proof-generating functionality, we would need a separate argument to ensure that the proof itself does not leak any meaningful information about the honest parties' witnesses beyond their validity.

2.2.1 Application: Collaborative Groth16 Proof Generation

The computation of Groth16 zk-SNARKs [Gro16] can be expressed as a degree-2 computation over the extended witness. Consequently, any semi-honest MPC protocol that is secure against additive attacks for computing Groth16 proofs inherently achieves L-malicious security. Furthermore, due to the self-verifiability of zk-SNARK proofs, such an MPC protocol also achieves standard malicious security. In other words, there exists a simulator $\mathcal{S}$ that, given oracle access to an ideal functionality computing the Groth16 proof, can generate a view for the adversary that is indistinguishable from its view in the actual protocol.

To further demonstrate that such an MPC satisfies the above notion of t-zero-knowledge against malicious adversaries, we modify $\mathcal{S}$ such that, instead of querying the ideal functionality computing the Groth16 proof, it invokes the simulator $\mathcal{S}_{\text{zk}}$ of the Groth16 proof system (whose existence follows from the zero-knowledge property of Groth16). $\mathcal{S}_{\text{zk}}$ can simulate a proof using only the information about whether the combined witness is valid or not. This modification to $\mathcal{S}$ allows us to establish that a semi-honest MPC for computing Groth16 proofs constitutes a collaborative zk-SNARK that achieves t-zero-knowledge against malicious adversaries. Since this proof follows easily from the above observations, we omit a formal proof.

2.3 Malicious Security for Free in Honest Majority: Reactive Functions

In the previous section, we discussed two simple types of *non-reactive* functionalities – affine and degree-two computations – where the function processes the input in a single step to produce the output. The computation of Groth16 zk-SNARKs was an example of a non-reactive degree-two computation.

However, many zk-SNARKs used in practice are derived by applying the Fiat-Shamir transform [FS87] to interactive proofs, making their proof computation a *reactive* functionality. Here, the function iteratively computes intermediate outputs, which are then used to determine subsequent inputs until the final output is obtained. For instance, given initial inputs x_1, y_1, the parties compute $z_1 = f_1(x_1, y_1)$, reconstruct z_1, and use it to determine the next inputs x_2, y_2. They then compute $z_2 = f_2(z_1, x_1, y_1, x_2, y_2)$, reconstruct z_2, and proceed similarly with x_3, y_3 to compute the final output: $z_3 = f_3(z_1, z_2, x_1, y_1, x_2, y_2, x_3, y_3)$.

In this section, we first revisit the observations from the previous section in the context of reactive functionalities. Then, in section 2.3.1, we demonstrate how these ideas can be used to show that a semi-honest collaborative Bulletproofs generation [BBB+18] is inherently secure even against malicious adversaries. However, the initial discussion in this section alone is not sufficient; in section 2.3.1, we also make another interesting observation about the structure of bulletproofs which helps us establish this result.

MPC Secure up to Additive Attacks for Reactive Functions. Recall that the observations from [GIP+14], discussed in section 2.1, apply specifically to semi-honest MPC protocols computing non-reactive functionalities. Indeed, the second observation in [GIP+14] states that certain semi-honest MPC protocols remain private against malicious adversaries until the final output reconstruction – assuming the entire output is revealed only in the last round.

However, when computing a reactive functionality, as discussed above, the parties need to reconstruct intermediate outputs at various steps. Consequently, we can only guarantee that each individual sub-function computation remains private against a malicious adversary until its corresponding output is reconstructed. Effectively, the computation of such reactive functions can be seen as a sequence of *multiple sub-MPC protocols* – one for each sub-function computation – each independently satisfying the two observations made in [GIP+14].

L-Malicious Security for MPC Computing Reactive Functions. We extend the notion of L-malicious security to MPC protocols computing reactive functionalities, considering an ideal functionality $\mathcal{F}'$ that, before sending each intermediate output, receives an arbitrary linear function from the adversary. For the example above, $\mathcal{F}'$ receives three linear functions (denoted L_1, L_2, L_3) from the adversary at different stages and computes the outputs as follows:

$$z_1 = L_1(f_1(x_1, y_1)), \quad z_2 = L_2(f_2(z_1, x_1, y_1, x_2, y_2)),$$

$$z_3 = L_3(f_3(z_1, z_2, x_1, y_1, x_2, y_2, x_3, y_3)).$$

This definition ensures that adversarial influence remains constrained to linear modifications of intermediate and final results while preserving the privacy guarantees of the protocol.

We now explore some generic examples of reactive functions composed of affine and degree-2 sub-functions. And then discuss whether a semi-honest, honest majority MPC (satisfying the above properties) for computing such reactive functions can also be shown to be inherently L-maliciously secure.

Reactive Functions Composed of Affine and Degree-Two Sub-Functions. Consider a reactive functionality similar to the previous example, where the sub-functions are defined as follows: (1) f_1 is an affine function over x_1, y_1, (2) f_2 is a degree-2 computation over x_1, y_1, x_2, y_2 and (3) f_3 is an affine function over $x_1, y_1, x_2, y_2, x_3, y_3$ [4]. Here $x_1, y_1, x_2, y_2, x_3, y_3$ are all secret shared inputs. Next, we analyze whether an MPC of the above form for computing this reactive function is also L-maliciously secure.

Let $\mathcal{S}^1_{<\text{last}}, \mathcal{S}^2_{<\text{last}}, \mathcal{S}^3_{<\text{last}}$ be the simulators corresponding to the three sub-MPC protocols, guaranteed by the observations in [GIP+14], that simulate an indistinguishable view for the adversary in all

[4] Note that we do not need to specify whether these sub-functions are affine or degree-2 over z_1 and z_2 as these values are publicly reconstructed and can thus be treated as public constants.

but the final round of each sub-protocol. To establish security against a malicious adversary for the overall protocol, our goal is to construct a new simulator $\mathcal{S}^{\mathcal{F}'}$ that leverages these simulators and, with oracle access to an ideal functionality $\mathcal{F}'$, produces an indistinguishable view of the adversary across the entire protocol.

- Since f_1 is an affine function, we can use $\mathcal{S}^1_{<\text{last}}$ along with the techniques from section 2.2 to simulate the entire first sub-MPC, including output reconstruction. Moreover, as f_1 is affine, no adversarial errors are introduced in this step and hence the linear function sent to $\mathcal{F}'$ by the simulator at this point will be the identity function.
- Next, for the degree-2 function f_2, we similarly utilize $\mathcal{S}^2_{<\text{last}}$ and the ideas from section 2.2 to simulate the adversary's view throughout the second sub-protocol. This includes determining the effective linear function for mauling the output, which will be sent to $\mathcal{F}'$.
- Finally, the adversary's view during the computation of f_3 can be simulated in a manner similar to how we did it for f_1.

Hence, we have shown that an MPC secure up to additive attacks, when computing this reactive functionality, also achieves L-malicious security. Importantly, these ideas can be extended to any reactive function composed of an arbitrary composition of affine and degree-2 computations.

2.3.1 Application: Collaborative Bulletproofs Generation

Bulletproofs [BBB+18] is a popular succinct proof system that is used in platforms such as Renegade, Monero, Halo2, for BLS signature aggregation in ethereum, etc. Recently, Renegade, a blockchain startup, proposed a collaborative zk-SNARK based on Bulletproofs in their white paper [BKb]. The semi-honest version of their protocol is an MPC secure up to additive attacks. We now discuss the main observations that help us show that this semi-honest protocol is also inherently maliciously secure. Finally, we will discuss why this constitutes a collaborative zk-SNARK achieving t-zero-knowledge against malicious adversaries.

Bulletproofs is an example of a zk-SNARK that is obtained by transforming an interactive proof system into a non-interactive one in the random oracle model. The non-interactive variant of Bulletproofs is an example of a reactive function comprising the following sub-functions:

1. *First Sub-Function.* Only requires affine computations over the extended witness.
2. *Second Sub-Function.* After the first message is computed, computing the second message only requires some additional degree-two computations over the extended witness.
3. *Third Sub-Function.* Only requires some affine operations over the extended witness after the first and second messages have been computed.

This mirrors the previous example. From that discussion and the self-verifiability of Bulletproofs, it follows that a semi-honest MPC protocol, secure against additive attacks for computing the above version of Bulletproofs, is inherently secure against a malicious adversary. However, this only establishes the existence of a simulator that can generate an indistinguishable view for the adversary, assuming oracle access to an ideal functionality computing the above reactive functionality (i.e., the Bulletproof).

Similar to our discussion of collaborative Groth16 proof generation in the previous section, to further demonstrate that this MPC is a collaborative zk-SNARK achieving t-zero-knowledge against malicious adversaries, we need to slightly modify this simulator. Specifically, instead of giving the simulator oracle access to an ideal functionality computing the Bulletproof, we instead allow it to

simulate the Bulletproof itself – which is possible due to the zero-knowledge property of Bulletproofs. The simulator then uses this simulated proof to generate an indistinguishable view of the adversary throughout the protocol.

Last Step in Bulletproofs. We note that the Bulletproofs protocol does not end here. Specifically, the output of the third sub-function above is *not succinct*, as it includes two long vectors, $\mathbf{l}$ and $\mathbf{r}$, along with a scalar $\hat{t}$, which must satisfy the relation $\langle \mathbf{l}, \mathbf{r} \rangle = \hat{t}$, if the witness is valid. Instead of directly sending $\mathbf{l}$ and $\mathbf{r}$ to the verifier, Bulletproofs' prover employs a *succinct inner-product argument* to prove that the inner product of these vectors – committed inside two compressing commitments – equals $\hat{t}$. This inner-product argument is a logarithmic-round interactive protocol, which can be made non-interactive using the Fiat-Shamir paradigm in the random oracle model, effectively constituting a reactive functionality comprising logarithmically-many sub-functions. The work of [BKb] proposes a collaborative version of Bulletproofs, requiring the proof generation algorithm – including this inner-product argument – to be computed within an MPC protocol.

Collaborative Generation of Succinct Bulletproofs. We now examine whether a semi-honest MPC protocol, secure against additive attacks (such as the one used in [BKb]) for computing the Bulletproofs – including the inner-product argument – also achieves t-zero-knowledge against malicious adversaries. To establish this, we construct a simulator $\mathcal{S}$ that receives a bit b indicating whether the combined witnesses of all parties are valid, as follows:

- Recall that the above three-round, non-succinct variant of Bulletproofs is a zero-knowledge proof system. Consequently, there exists a simulator $\mathcal{S}_{\text{zk}}$, corresponding to the zero-knowledge property of this proof system, which can simulate the non-succinct version of Bulletproofs without knowledge of the witness. This simulated Bulletproof must also include the values $\mathbf{l}, \mathbf{r}, \hat{t}$. Using b and $\mathcal{S}_{\text{zk}}$, $\mathcal{S}$ first simulates a non-succinct Bulletproof.

- Next, $\mathcal{S}$ uses this simulated proof to simulate the adversary's view during the computation of the first two sub-functions, as discussed earlier.

- The third sub-function in the above description is replaced with a reactive functionality (i.e., the inner-product argument) consisting of logarithmically many sub-functions. Observe that, regardless of the computations performed in these sub-functions, since $\mathcal{S}$ already knows $\mathbf{l}, \mathbf{r}, \hat{t}$ – the only inputs to the inner-product argument – it can honestly emulate the adversary's view during the collaborative generation of the inner-product argument using any semi-honest MPC protocol secure against additive attacks. Because this computation is performed using such an MPC, the simulator can also extract any additive errors introduced during the inner-product argument. However, since $\mathcal{S}$ has complete knowledge of the inputs, it can fully simulate all intermediate computations and final output reconstruction steps within this reactive functionality. As a result, the adversary's view during the collaborative generation of the entire inner-product argument remains simulatable.

This concludes the discussion on why the semi-honest variant of collaborative Bulletproofs [BKb] also achieves t-zero-knowledge against a malicious adversary. We refer the reader to section 6.2 for a detailed proof.

2.4 Malicious Security for Free in Honest Majority: Randomized Computations

So far, we have not explicitly leveraged the fact that the generation of zero-knowledge SNARKs inherently involves randomized computations. Now: (1) in section 2.4.1, we first examine whether the above observations extend to more complex non-reactive functions that also take random values as input; (2) then, in section 2.4.2, we show that these observations can similarly be extended to certain *randomized encodings* of *deterministic* non-reactive functionalities. Finally, in section 2.4.3,

we apply these two ideas to establish that semi-honest secure collaborative Plonk [GWC19] proof generation is also inherently secure against malicious adversaries.

2.4.1 Special Randomized Functions

We consider arbitrary-depth randomized functions that satisfy the following two properties: (1) the honestly computed output of the function is uniformly distributed; (2) Even if the adversary injects arbitrary additive errors during the computation of multiplication gates, the cumulative error in the final output will be uniformly distributed.

A simple example of a randomized function that satisfies both properties is $f(x, y; r) = r \cdot (x \cdot y)$, where, all three inputs x, y, r are assumed to be secret-shared, but r is uniformly distributed. If the adversary does not introduce any errors, the final output $r \cdot (x \cdot y)$ remains uniformly distributed. Moreover, if the adversary introduces an error ϵ_1 during the computation of $x \cdot y$ and an error ϵ_2 during the subsequent multiplication with r, the resulting output becomes $r(x \cdot y) + r\epsilon_1 + \epsilon_2$. Since r is uniformly distributed, the error $r\epsilon_1$ is also uniformly distributed.

When computing such functions using an MPC protocol secure against additive attacks, the final output remains uniformly distributed regardless of any errors introduced by a malicious adversary. Consequently, the adversary's view in the final round is straightforward for the simulator to simulate. Therefore, a semi-honest MPC secure up to additive attacks for computing such functions remains secure against malicious adversaries.

2.4.2 Randomized Encoding

Next we consider deterministic functions that are essentially sequential multiplications of the form $x_1 \cdot x_2 \cdot x_3 \cdot \ldots$, where each x_i is a secret-shared input. While this does not fall into any of the three types of non-reactive functions we have discussed so far, we focus on how it is computed in the protocol described in [OB22].

Instead of performing these sequential multiplications over secret-shared values x_i in multiple rounds of interaction, [OB22] employs Bar-Ilan and Beaver's [BIB89] constant-round MPC protocol. This protocol first reduces the sequential multiplications to $O(n)$ parallel randomized multiplications over secret-shared values. The final value of $x_1 \cdot x_2 \cdot x_3 \cdot \ldots$ is then obtained by multiplying the reconstructed outputs of the parallel multiplications and performing some additional linear operations.

Using such an MPC to compute the above sub-function effectively transforms the computation process into a special randomized function, allowing a simulator to simulate the view of an adversary throughout the entire computation of this sub-function. We note that this is possible because, for the three types of non-reactive functions we have discussed so far, we only placed restrictions on the computation done over *secret-shared values*, not on the complexity of the actual function being computed.

2.4.3 Application: Collaborative Plonk [GWC19] Proof Generation

Next, we turn our attention to another widely used zk-SNARK – Plonk [GWC19]. Similar to Bulletproofs, Plonk is derived by transforming an interactive proof system into a non-interactive one using the Fiat-Shamir transform. Consequently, its computation can be modeled as a reactive functionality composed of the following sub-functions:

1. *First Sub-Function.* Only requires linear operations.
2. *Second Sub-Function.* The second sub-function essentially consists of linear operations and sequential multiplications of the form $x_1 \cdot x_2 \cdot x_3 \cdot \ldots$.
3. *Third Sub-Function.* This can be viewed as a special randomized function.

4. *Fourth Sub-Function.* This can be computed using only linear operations.

5. *Fifth Sub-Function.* This can also be computed using only linear operations.

Simulating the Plonk Reactive Functionality. Given the above observation, let us now revisit whether we can simulate an adversary's view during the computation of the Plonk reactive functionality. Following the ideas from section 2.3, we can show that it is possible to simulate the entire view of an adversary during the computation of the first two sub-functions. Moreover, errors from the second message do not influence the output distribution of the third sub-function and hence the adversary's view during its computation can be simulated. However, while the fourth and fifth messages involve only linear operations, it is less clear whether we can simulate them due to potential errors introduced during the computation of the second sub-function.

In section 7.2, we show that while the output of the fourth sub-function can indeed be simulated, the fifth one cannot. To address this, we propose a slight modification to the semi-honest collaborative Plonk protocol. In particular, we show that it suffices for the parties to check whether a specific wire value during the computation of the fifth message equals zero and to reconstruct the final output (i.e., the fifth message) only if this condition holds. We then demonstrate that the semi-honest collaborative Plonk protocol, with this simple additional constant-sized check, is secure against malicious adversaries. Finally, using similar observations as in the previous subsections, we establish that such an MPC constitutes a collaborative zk-SNARK that achieves t-zero-knowledge against malicious adversaries.

Remark. As discussed above, computations of the form $x_1 \cdot x_2 \cdot x_3 \cdot \ldots$ can be performed either through sequential calls to the multiplication subroutine or by using Bar-Ilan and Beaver's constant-round protocol. The computation of the second message in Plonk involves two types of sequential multiplication: one in which the number of terms to be multiplied depends on the size of the NP relation and the other in which only three values need to be multiplied together. Although the Bar-Ilan and Beaver protocol is clearly a better choice for multiplying a large number of terms, the same is not true when only a small, constant number of values are involved. This is because, in order to multiply n terms, the Bar-Ilan and Beaver protocol makes $2(n-1)$ calls to the multiplication subroutine. When $n = 3$, this requires 4 calls to a multiplication subroutine, whereas a naive approach will only require making 2 calls to the multiplication subroutine. The round complexity for $n = 3$ is also similar in both methods.

In the collaborative Plonk presented in [OB22], it wasn't clear whether they suggested using the Bar-Ilan and Beaver protocol for multiplying the 3 terms together or just use the naive approach. We opted to use the Bar-Ilan and Beaver protocol for this step because that allowed us to prove that the output of the second sub-function is simulatable. If instead, we had used the naive approach, the computation of the second sub-function would not constitute a special randomized function. In that case, we would have had to introduce additional consistency checks before reconstructing the second output to detect errors. Using generic approaches to perform these checks would have doubled the number of calls that we made previously to the multiplication subroutine when computing the second message, leading to higher communication overhead compared to our approach.

Dishonest Majority Setting. We note that these observations about the semi-honest collaborative zk-SNARKs for Bulletproofs, Plonk and Groth16 apply only in the honest majority setting and do not extend to the dishonest majority setting. This is because the nature of additive attacks in the dishonest majority setting is slightly different. In that setting, a malicious adversary can also inject additive errors during the computation of addition gates. As a result, we can no longer argue that that the adversary cannot introduce errors during the computation of linear gates.

3 Preliminaries

Notations. For any $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \cdots, n\}$. Let λ be the security parameter. Let $\mathbb{F}$ be a large field whose size is prime and superpolynomial ($|\mathbb{F}| = \lambda^{\omega(1)}$). Our polynomials will be defined over $\mathbb{F}$. Let $H = \{1, \omega, \ldots, \omega^{n-1}\}$ be the nth roots of unity in $\mathbb{F}$. I.e. H is a cyclic multiplicative subgroup of $\mathbb{F}$ of order n that is generated by ω.

We now define a few polynomials and some polynomial algorithms that will be used in our protocols. Let $\mathsf{Z}_H(X) = X^n - 1$, the vanishing polynomial over H. Note that $\mathsf{Z}_H(x) = 0$ if and only if $x \in H$, and note that $\mathsf{Z}_H(X\omega) = \mathsf{Z}_H(X)$. A Lagrange basis $\{\mathsf{L}_i(X)\}_{i\in[n]}$ satisfies the property that for each $i \in [n]$, and each $j \in [n]\backslash\{i\}$, $\mathsf{L}_i(\omega^i) = 1$ and $\mathsf{L}_i(\omega^j) = 0$. Particularly, $\mathsf{L}_i(X) = \frac{\omega^i \cdot (X^n-1)}{n\cdot(X-\omega^i)}$.

Given two polynomials $\mathsf{a}(X), \mathsf{b}(X)$ such that $\mathsf{b}(X) \neq 0$, we denote the quotient and remainder polynomials obtained from dividing $\mathsf{a}(X)$ by $\mathsf{b}(X)$ by $\mathsf{Qt}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right)$ and $\mathsf{Rd}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right)$, respectively. I.e. $\mathsf{a}(X) = \mathsf{Qt}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right) \cdot \mathsf{b}(X) + \mathsf{Rd}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right)$.

3.1 Multiparty Computation (MPC) Functionalities

In this section, we will define some standard honest majority MPC functionalities and some specific functionalities that are invoked in our main protocols.

Secret Sharing. We will use a t-out-of-n secret sharing[5]. For $x \in \mathbb{F}$, we use $[x]$ to denote the t-out-of-n shares of x. We use $[x]_i$ to denote party P_i's share, and $[x]_S$ to denote shares held by a subset S of parties. We use the function $\mathsf{share}(x)$ to compute shares of x, and the function $\mathsf{share}\left(x, \{x'_j\}_{j\in J}\right)$, for $J \subset [n]$ with $|J| \leq t$, to generate shares of x such that $[x]_J = \{x'_j\}_{j\in J}$. The function $\mathsf{open}([x], t)$ reconstructs shares, such that: if $[x]$ is the correct sharing of x, then $\mathsf{open}([x], t)$ outputs x or $\perp$ (if some dishonest parties deviate from the protocol) and if incorrect, then it outputs $\perp$ with overwhelming probability. In all the secret-sharing schemes we use, the value of x can be reconstructed from any t+1 shares of $[x]$ by applying a public *linear* function to the shares.

Some standard MPC functionalities. Our protocol will make use of some standard sub-protocols from [CGH+18], which are represented by the following ideal functionalities: $\mathcal{F}_{\text{input}}$ is used to generate secret shares of M inputs; $\mathcal{F}_{\text{rand}}$ is used to generate secret shares of a random field element; $\mathcal{F}_{\text{coin}}$ is a used to generate a random field element; $\mathcal{F}_{\text{checkZero}}$ is used to check if $x = 0$ or not, given the shares $[x]$ (false negatives and positives are possible); $\mathcal{F}_{\text{mult}}$ is used to multiply secret shares $[x]$ and $[y]$, where a malicious adversary is allowed to specify an additive error ε and $\mathcal{F}_{\text{mult}}$ outputs $[x \cdot y + \varepsilon]$. Detailed definitions of these functionalities are given in appendix A.

$\mathcal{F}_{\text{rand}}$ as described in appendix A assumes that this ideal functionality picks a random value r and then deals an "honest" sharing of r to all honest parties, which can be instantiated using a maliciously secure sub-protocol [LN17]. We describe all our protocols in the $\mathcal{F}_{\text{rand}}$-hybrid model for ease of exposition[6]. Furthermore, we can always assume that all the random sharings needed throughout the protocol can be generated in an offline phase along with the extended witness generation using a maliciously secure generic MPC.

However, we note that $\mathcal{F}_{\text{rand}}$ can also be instantiated using a semi-honest subprotocol [DN07] if the number of corrupted parties is exactly $N/2 - 1$. But the semi-honest subprotocols for fewer corruptions result in a slightly modified functionality $\mathcal{F}'_{\text{rand}}$: in particular, the adversary can also inject additive errors into secret shares of the random value that $\mathcal{F}'_{\text{rand}}$ computes. We conjecture

[5] See [CGH+18] for more detail.

[6] Most prior work on maliciously secure MPC, discussed in section 1.2, also assume that the underlying semi-honest MPC works in the $\mathcal{F}_{\text{rand}}$-hybrid model.

that our current proof for the malicious security of the semi-honest collaborative zk-SNARK in the $\mathcal{F}_{\text{rand}}$-hybrid model can be extended to the $\mathcal{F}'_{\text{rand}}$-hybrid model[7]. We leave a formal proof of this for the future.

MPC In The Exponent. In protocols such as Plonk, we represent some secret values "in the exponent" of a group element. It is possible to secret-share these values and to use them as inputs to simple multiparty computations. Prior works [ST19, OB22] have explored generalizations of polynomial-based secret-sharing schemes for group operations. Let $\mathbb{G}_1$ be a group of order p, with generator g_1, such that each element $X \in \mathbb{G}_1$ can be represented as g_1^x, where $x \in \mathbb{Z}_p$. The main idea in these works is to first compute secret shares of 1: $[1] = (a_1, \ldots, a_n)$ and then compute shares of $[x]_1 := g_1^x$ as $[[x]_1] := ((g_1^x)^{a_1}, \ldots, (g_1^x)^{a_n})$. This allows us to perform arithmetic field operations in the exponent which can be used for group exponentiation and for multiplying group elements. An example of simple multiparty computations in the exponent that we can compute is as follows: given three types of secret inputs $[x] = (x_1, \ldots, x_n)$, $[y]_1 = g_1^y$, $[[z]_1] = (g_1^{z_1}, \ldots, g_1^{z_n})$, and public coefficients $\alpha, \beta \in \mathbb{F}$, $[[\alpha \cdot x \cdot y + \beta \cdot z]_1]$ can be computed locally by every party (and hence is immune from additive attacks).

MPC That Acts On Polynomials. We now give notations and descriptions of a few functionalities that capture MPC operations on secret shares of polynomials. The shares of a polynomial $\mathsf{a}(X)$ of degree $\leq d_a$ are denoted by $[\mathsf{a}(X)]$ (which are the shares of its coefficients). Using only local operations on shares of $[\mathsf{a}(X)]$ and $[\mathsf{b}(X)]$, the parties can compute the following simple operations on polynomials: multiplying $[\mathsf{a}(X)]$ by a public polynomial $\mathsf{b}(X)$ to obtain $[\mathsf{a}(X) \cdot \mathsf{b}(X)]$; dividing $[\mathsf{a}(X)]$ by a public polynomial $\mathsf{b}(X)$ to obtain $\left[\mathsf{Qt}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right)\right]$ and $\left[\mathsf{Rd}\left(\frac{\mathsf{a}(X)}{\mathsf{b}(X)}\right)\right]$; taking a linear combination of $[\mathsf{a}(X)]$ and $[\mathsf{b}(X)]$ to obtain $[x \cdot \mathsf{a}(X) + y \cdot \mathsf{b}(X)]$; evaluation at a point x to get $[\mathsf{a}(x)]$; evaluation in the exponent given $([\mathsf{a}(X)], [1]_1, [x]_1, \ldots, [x^{d_a}]_1)$ to get $[[\mathsf{a}(x)]_1]$.

We additionally define a functionality to multiply two secret-shared polynomials (up to additive attacks), denoted by $\mathcal{F}_{\text{polyMult}}$. It takes as input two secret shared polynomials $[\mathsf{a}(X)]$ and $[\mathsf{b}(X)]$ and combines all the additive errors specified by the adversary to output $[\mathsf{a}(X) \cdot \mathsf{b}(X) + \varepsilon(X)]$. The detailed functionality description and a sub-protocol to realize it is given in appendix A.2.

3.2 Collaborative zk-SNARKs

A *collaborative zk-SNARK* is a non-interactive proof system in which multiple mutually distrustful provers compute a proof from their combined witnesses in a way that preserves zero-knowledge against an adversarial subset of $\leq t$ provers. Unless otherwise indicated, the definitions in this section come almost verbatim from [OB22].

Relation: [AB09, Gol01] For any language L in NP, a **relation** $\mathfrak{R}_L$ for L is a set of instance-witness pairs $(\mathcal{X}, \mathsf{wtn})$ such that $\mathcal{X} \in L$, and wtn is a witness for $\mathcal{X}$. Furthermore, there is a polynomial-time function $V_{\mathfrak{R}}$ to verify the witness. For all $(\mathcal{X}, \mathsf{wtn}) \in \mathfrak{R}$: $V_{\mathfrak{R}}(\mathcal{X}, \mathsf{wtn}) = 1$. And for all $\mathcal{X} \notin L$, there does not exist a wtn such that $V_{\mathfrak{R}}(\mathcal{X}, \mathsf{wtn}) = 1$. Finally, the size of the relation, $|\mathfrak{R}|$, is the time needed to compute $V_{\mathfrak{R}}(\mathcal{X}, \mathsf{wtn})$ (or the size of the circuit that computes it).

Random oracle: Let $\mathsf{p}(\lambda)$ be a polynomial upper bound on the total communication of the proof system. Let $\mathcal{U}(\lambda) := \{H | H : \{0,1\}^{\leq \mathsf{p}(\lambda)} \to \{0,1\}^{\lambda}\}$. A random oracle H is a function sampled uniformly at random from $\mathcal{U}(\lambda)$. H can be reprogrammed on certain inputs. The reprogrammed oracle is written as $H[\mu]$, where μ specifies the reprogrammed points. Specifically, μ is a partial

[7] The instantiation of $\mathcal{F}_{\text{mult}}$ in [CGH+18] is also designed in the $\mathcal{F}_{\text{rand}}$-hybrid model. When switching to $\mathcal{F}'_{\text{rand}}$, the $\mathcal{F}_{\text{mult}}$ functionality also changes to $\mathcal{F}'_{\text{mult}}$, allowing the adversary to inject additive errors on the shares received by the honest parties.

function whose domain is a subset of Domain(H). $H[\mu]$ maps x to $\mu(x)$ if $x \in \text{Domain}(\mu)$ and maps x to $H(x)$ if $x \notin \text{Domain}(\mu)$.

Protocol Syntax: A collaborative zk-SNARK in the random oracle model is a protocol among N provers $\vec{\mathcal{P}} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$ and a verifier $\mathcal{V}$. Each prover $\mathcal{P}_i$ has a witness fragment wtn_i, and the witness fragments combine to form the witness: $\mathsf{wtn} = (\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$. The protocol consists of the following phases:

- $\mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \to \mathsf{pp}$: Generates the public parameters pp.
- $\mathsf{Prove}^H(\mathsf{pp}, \mathcal{X}, \mathsf{wtn}) \to \{\pi, \perp\}$: If $(\mathcal{X}, \mathsf{wtn}) \in \mathfrak{R}$, then it outputs a proof π. Otherwise, it outputs $\perp$. Prove is run by the provers.
- $\mathsf{Verify}^H(\mathsf{pp}, \mathcal{X}, \pi) \to \{0, 1\}$: Verifies π. It outputs 1 to indicate acceptance and 0 to indicate rejection. Verify is run by the verifier.

Each phase can query the random oracle H. Also, the Prove phase can be subdivided into two steps: extended witness generation and proof generation.

Definition 3.1 (Collaborative zk-SNARK [OB22])**.** In the random oracle model, a collaborative zk-SNARK $(\mathsf{Setup}^H, \mathsf{Prove}^H, \mathsf{Verify}^H)$ for a relation $\mathfrak{R}$ secure against t malicious provers satisfies the following properties:

Completeness: For any $(\mathcal{X}, \mathsf{wtn}) \in \mathfrak{R}$, the verifier will accept the proof with overwhelming probability. In other words, for any $(\mathcal{X}, \mathsf{wtn}) \in \mathfrak{R}$, there is a negligible function $\varepsilon(\lambda)$ such that:

$$\Pr\left[\mathsf{Verify}^H(\mathsf{pp}, \mathcal{X}, \pi) = 0 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \\ \pi \leftarrow \mathsf{Prove}^H(\mathsf{pp}, \mathcal{X}, \mathsf{wtn}) \end{array}\right] = \varepsilon(\lambda)$$

Knowledge Soundness: Informally, knowledge soundness says that if the provers can convince the verifier to accept with some probability, then there is an extractor that can extract from the provers a valid witness wtn, with similar probability.

Formally, the protocol has knowledge soundness if for any $\mathcal{X}$ and any efficient provers $\vec{\mathcal{P}} = (\mathcal{P}_1, \ldots, \mathcal{P}_N)$, there exists an efficient extractor Ext with the following properties: Ext gets oracle access to H. Ext can run the provers multiple times, reprogramming H each time, and Ext receives only the provers' final output from each run. This is denoted as $\mathsf{Ext}^{H, \vec{\mathcal{P}}^H}$. Finally, there exists a negligible function $\varepsilon(\lambda)$ such that:

$$\Pr\left[(\mathcal{X}, \mathsf{wtn}) \in \mathfrak{R} : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \\ \mathsf{wtn} \leftarrow \mathsf{Ext}^{H, \vec{\mathcal{P}}^H}(\mathsf{pp}, \mathcal{X}) \end{array}\right] \geq \Pr\left[\mathsf{Verify}^H(\mathsf{pp}, \mathcal{X}, \pi) = 1 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \\ \pi \leftarrow \vec{\mathcal{P}}^H(\mathsf{pp}, \mathcal{X}) \end{array}\right] - \varepsilon(\lambda)$$

Succinctness: The size of π and the runtime of $\mathsf{Verify}^H(\mathsf{pp}, \mathcal{X}, \pi)$ are $o(|\mathfrak{R}|)$.

t-Zero-Knowledge: t-zero-knowledge ensures that no subset of t parties or fewer will learn anything about the witnesses of the other parties, except whether the combined witness is valid. Formally, a collaborative zk-SNARK satisfies t-zero-knowledge if for any PPT malicious adversary $\mathcal{A}$ controlling $k \leq t$ provers $(\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k})$, there exists an efficient simulator Sim such that for all $(\mathcal{X}, \mathsf{wtn})$, and for all efficient distinguishers D, there is a negligible function $\varepsilon(\lambda)$ such that:

$$\left|\Pr\left[D^{H[\mu]}(\mathsf{tr}) = 1 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \\ b \leftarrow V_{\mathfrak{R}}(\mathcal{X}, \mathsf{wtn}) \in \{0,1\} \\ (\mathsf{tr}, \mu) \leftarrow \mathsf{Sim}^H(\mathsf{pp}, \mathcal{X}, \mathsf{wtn}_{i_1}, \ldots, \mathsf{wtn}_{i_k}, b) \end{array}\right] - \Pr\left[D^H(\mathsf{tr}) = 1 : \begin{array}{c} H \xleftarrow{\$} \mathcal{U}(\lambda) \\ \mathsf{pp} \leftarrow \mathsf{Setup}^H(1^\lambda, \mathfrak{R}) \\ \mathsf{tr} \leftarrow \mathsf{View}^H_{\mathcal{A}}[\mathcal{X}, \mathsf{wtn}] \end{array}\right]\right| = \varepsilon(\lambda)$$

Here, tr is the protocol transcript, which lists the messages sent among the parties. And $\mathsf{View}^H_{\mathcal{A}}[\mathcal{X}, \mathsf{wtn}]$ is $\mathcal{A}$'s view of the protocol.

4 Pitfalls in Existing Approaches for Achieving Malicious Security

As discussed in section 1.1, there are two major pitfalls in the existing *maliciously secure* collaborative zk-SNARK design template. In this section, we formally describe these attacks. The first pitfall results from running the collaborative proof generation on invalid witnesses. As discussed in section 1.1, this can be due to an insider attack (which can be launched in the dishonest majority setting), or an outsider attack. In section 4.1, we show that a vulnerability in Groth16 [Gro16] – a widely-used, state-of-the-art zk-SNARK leads to an attack showing that a natural collaborative zk-SNARK based on Groth16 does not automatically hide invalid witnesses. Towards the end of that section, we discuss some potential mitigation strategies to address this pitfall. Next, we describe the second pitfall in section 4.2. In particular, we show that if the protocol only does a consistency check in the final round of a multi-round functionality, then an adversary can learn part of the witness, by injecting additive errors in round 1 and observing the output of that round – thereby breaking t-zero-knowledge.

In section 5, we propose a mitigation strategy to address pitfall 2, by designing a general compiler for transforming semi-honest collaborative proof generation to a maliciously secure one. This compiler assumes that the parties hold the shares of a valid combined witness and that pitfall 1 has been mitigated.

4.1 Pitfall 1: Insider and Outsider Attacks

The typical zero-knowledge guarantee in (single prover) proof systems states that *if the witness is valid*, then the resulting proof can be simulated without the knowledge of the witness. Nothing is guaranteed about proofs generated using invalid witnesses. In fact, a Groth16 proof generated using an invalid witness does not hide the witness. In this section, we show that an adversary can learn which of the two invalid witnesses was used to generate a Groth16 proof by simply inspecting it. This attack does not even require the adversary to inject errors into the prover's computation.

While not an issue in the single prover setting, this attack poses a significant issue in the multi-prover setting. In particular, recall that collaborative zk-SNARKs require a stronger form of zero-knowledge (called t-zero-knowledge) that ensures privacy of the honest parties' inputs even when the combined witness is invalid. In other words, t-zero-knowledge guarantees that for any combined witness (valid or not), the resulting proof can be simulated without knowledge of the honest parties' witnesses using only the witnesses of the adversary and a bit indicating whether the combined witness is valid.

4.1.1 Outsider Attack

A naïve approach for designing a collaborative zk-SNARK using an existing zk-SNARK would be to compute zk-SNARK proof *as-is* (i.e., without modifications), using a maliciously secure MPC. As discussed earlier, prior works on collaborative zk-SNARK essentially design a custom MPC for computing such zkSNARKs in a privacy-preserving distributed manner. In light of our attack, it is essential for the distributed provers to first check validity of the extended witness before computing the proof and only run the collaborative proof generation, if the extended witness is valid. We note that this attack works both in the honest and dishonest majority settings.

In the following theorem, we show an attack in which the use of an invalid witness in collaborative proof generation can leak the honest parties' witness fragment, even if no errors are introduced during the protocol itself. This attack works on a natural collaborative zk-SNARK based on Groth16 [Gro16] – a widely-used, state-of-the-art zk-SNARK. Prior work on collaborative zk-SNARKs [OB22, GGJ+23, LZW+24b, LZW+24a, BKb] does not explicitly address the need for verifying the extended witness during the witness extension step.

Theorem 4.1. *There is a collaborative zk-SNARK based on Groth16, such that if the protocol is modified so that the provers do not verify the validity of the extended witness before generating the proof, then the protocol no longer satisfies t-zero-knowledge.*

Proof. The collaborative zk-SNARK takes an extended witness, expressed as a rank-1 constraint system (R1CS), and generates a Groth16 proof using maliciously secure MPC. If the collaborative zk-SNARK does not verify validity of the extended witness before proceeding with collaborative proof generation, then we show that there is an attack that breaks t-zero-knowledge. We now describe the feature of Groth16 that allows us to launch an attack.

Rank 1 Constraint System (R1CS). Groth16 [Gro16] represents the NP relation to be verified as a rank-1 constraint system (R1CS), which is a generalization of arithmetic circuits. Let $\mathbb{Z}_p$ be the finite field over which the NP relation is defined. The R1CS system consists of three matrices $A, B, C \in \mathbb{Z}_p^{n \times m}$, such that for a valid statement-witness vector $Z \in \mathbb{Z}_p^{m \times 1}$, the following holds:

$$(A \cdot Z) \times (B \cdot Z) = (C \cdot Z)$$

where n, m are some parameters that depend on the size of the NP relation, and $\times$ is the element-wise product.

Overview of vulnerability in Groth16. Groth16 zk-SNARKs rely on bilinear pairings and operate in the common reference string (CRS) model, which encodes the matrices A, B and C. Given group generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, the prover computes three group elements $g_1^{\pi_1}$, $g_2^{\pi_2}$, and $g_1^{\pi_3}$ and sends them to the verifier. The verifier checks if $e(g_1^{\pi_1}, g_2^{\pi_2}) = e(g_1^{\pi_3}, \text{crs}_1) + f(\text{crs})$, where e denotes the bilinear pairing operation, crs_1 denotes some group element in the CRS and $f(\text{crs})$ some publicly computable function over the CRS.

For simplicity, assume that $n = 2$ and $m = 3$. The R1CS system can be re-written as:

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \times \begin{pmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} + \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

where $(y_1, y_2) = (0, 0)$ if and only if $Z = (z_1, z_2, z_3)$ is a valid statement-witness vector.

We observe that if the collaborative Groth16 zk-SNARK is computed using an invalid Z, then the verification check can leak y_1 and y_2, which in turn can reveal the values of z_1, z_2, z_3. In particular, given $g_1^{\pi_1}$, $g_2^{\pi_2}$, and $g_1^{\pi_3}$ computed using an invalid Z, an adversary can compute $e(g_1^{\pi_1}, g_2^{\pi_2}) - e(g_1^{\pi_3}, \text{crs}_1) - f(\text{crs})$. This value will be equal to $q(y_1, y_2, \text{crs})$, where q can be deterministically computed given y_1, y_2 and the CRS.

Concrete Attack on t-zero-knowledge. Consider an NP relation with statement $z_1 = 1$ and honest-party witness $z_2 = h$ and corrupted-party witness $z_3 = 1$ that must satisfy the following: $z_2 + z_3 = z_1$ and $z_2 \cdot z_3 = z_2$. This relation can be represented in R1CS as:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The statement-witness vector $Z = (1, h, 1)$ satisfies the following:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ 1 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ 1 \end{pmatrix} + \begin{pmatrix} h \\ 0 \end{pmatrix}$$

Note that the extended witness is valid if and only if $h = 0$.

t-zero-knowledge requires that for any two invalid witnesses, for which the corrupted party's inputs are the same, the corresponding outputs of the collaborative zk-SNARK will be indistinguishable to the corrupted party. However, in this example, it is easy to distinguish proofs generated from $h = 1$ and $h = 2$.

Observe that the values of y_1 and y_2 in this example depend only on the honest party's witness. An adversary can simply compute $q(h, 0, \text{crs})$ for $h = 1$ and $h = 2$ and determine which h-value matches the result of $e(g_1^{\pi_1}, g_2^{\pi_2}) - e(g_1^{\pi_3}, \text{crs}_1) - f(\text{crs})$. This allows the adversary to extract h from an invalid proof. □

4.1.2 Insider Attack

While the previous attack was effective only when the parties began with an invalid combined witness, in this section we reveal a more serious vulnerability in the dishonest majority setting. In particular, in this setting, we show that even if the provers start with a valid combined witness, the adversary can modify their shares of the extended witness such that the resulting sharing corresponds to that of an invalid witness. It can then then learn the honest party's witness fragment using an attack similar to the one discussed above. To mitigate this attack, the protocol should detect whether the adversary has changed the secret-shared value at any stage of the protocol.

Theorem 4.2. *There is a collaborative zk-SNARK based on Groth16, such that if the protocol is modified so that the witness extension step outputs a* regular additive sharing *of the extended witness, then the protocol no longer satisfies t-zero-knowledge.*

Proof. Consider an NP relation with statement $z_1 = 1$, honest-party witness $(z_2, z_3) = (h, h')$, and corrupted-party witness $z_4 = c$ that must satisfy the following: $z_2 + z_3 + z_4 = z_1$ and $z_3 \cdot z_4 = z_3$. This relation can be represented in R1CS as:

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The statement-witness vector $Z = (1, h, h', c)$ satisfies the following:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ h' \\ c \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ h' \\ c \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ h \\ h' \\ c \end{pmatrix} + \begin{pmatrix} h + h' + c - 1 \\ h' \cdot c - h' \end{pmatrix}$$

Note that the extended witness is valid if $c = 1$ and $h + h' = 0$.

Let us say that the extended witness is either $(h, h', c) = (0, 0, 1)$ or $(h, h', c) = (-1, 1, 1)$. In both cases, the extended witness is valid, and the corrupted party's witness fragment is the same: $c = 1$.

The corrupted party can learn which of the two witnesses was used in the protocol as follows. After the witness extension step, when the parties jointly hold an additive sharing $[c]$, the corrupted party adds 1 to its share so that the sharing represents $[c + 1]$. Then the corrupted party participates honestly in the proof generation protocol. Since c was replaced with $c + 1$, then $(y_1, y_2) = (1, h')$. The adversary computes $q(1, h', \text{crs})$ for $h' = 0$ and $h' = 1$ and determines for which h'-value does q match the result of $e(g_1^{\pi_1}, g_2^{\pi_2}) - e(g_1^{\pi_3}, \text{crs}_1) - f(\text{crs})$. This allows the adversary to learn h'.

□

Mitigation. To mitigate this attack, we must ensure that the adversary cannot modify their shares of the extended witness between the witness extension and proof generation steps. This could be achieved using robust secret sharing based on AMD codes [CDF+08], which will detect if the adversary modifies the secret-shared value. A detailed mitigation of this attack is beyond the scope of our

work. We just highlight here that in the dishonest majority setting, the witness extension and proof generation phases should not be considered independently, as was done in prior works. From here on, our focus is on the honest majority setting.

4.2 Pitfall 2: Computing Reactive Functionalities Requires Multiple Consistency Checks.

As discussed in section 2.1, recent works on maliciously secure MPC have used the additive attack paradigm [GIP+14] to achieve malicious security efficiently. The technique entails running two copies of a semi-honest MPC and detecting additive errors in a consistency check before reconstructing the final output. This works well for *non-reactive* functionalities, which only open their outputs at the end of the protocol. In particular, for zk-SNARKs that do not use an underlying interactive proof, such as Groth16, we can apply this technique without modifications. However, for many zk-SNARKs (obtained from multi-round proofs), such as Plonk [GWC19], the prover is naturally expressed as a reactive functionality, i.e., they open the outputs of intermediate rounds before starting the next round.

In such cases, it may seem acceptable to defer the consistency check to the final round of the protocol, and at that point to check for errors in all previous rounds. However, this check comes too late to prevent the adversary from violating input privacy. In this section, we show that there exists a collaborative zk-SNARK, where the adversary can learn part of the witness if they are allowed to inject additive errors in the first round and observe the resulting output of the first round. If we wait until the final round to do the consistency check, then although we will detect the errors injected in the first round, it will be too late to prevent the adversary from learning part of the witness. We prove this formally through an attack in the following theorem.

Theorem 4.3. *There exists a collaborative zk-SNARK such that if the adversary is allowed to inject additive errors in the first round and observe the output of the first round, then the protocol no longer satisfies t-zero-knowledge.*

Proof. Let wtn^0 and wtn^1 be two accepting witnesses, defined as follows:

$$\mathsf{wtn}^0 = (a^0, b^0, c^0) = (0, 0, 0)$$
$$\mathsf{wtn}^1 = (a^1, b^1, c^1) = (0, 0, 1)$$

The only difference between them is the value of c.

Now consider a zk-SNARK derived from an interactive proof system in the random oracle model. Let this zk-SNARK be such that in the first round, the prover computes $d = a \cdot b \cdot c$ and outputs 1 if $d = 0$, and otherwise it outputs 0. Both wtn^0 and wtn^1 will result in $d = 0$ if the adversary does not introduce any errors.

The collaborative computation of this step involves computing:

$$\mathcal{F}_{\text{checkZero}}(\mathcal{F}_{\text{mult}}(\mathcal{F}_{\text{mult}}([a], [b]), [c]))$$

Let us also assume that the collaborative zk-SNARK for such a proof system only checks for errors in the final round.

When using a semi-honest secure MPC to do this computation, a malicious adversary can add additive errors on each multiplication. Let our adverary introduce an error of $\varepsilon = 1$ when a and b are multiplied. Then the output will be different for the two witnesses:

$$d^0 = (a^0 \cdot b^0 + 1) \cdot c^0 = a^0 \cdot b^0 \cdot c^0 + c^0 = 0$$
$$d^1 = (a^1 \cdot b^1 + 1) \cdot c^1 = a^1 \cdot b^1 \cdot c^1 + c^1 = 1$$

The adversary then observes the output of round 1. If the output is 1 (i.e., $d = 0$), the adversary infers that wtn^0 was used in the computation,. Otherwise, it infers that wtn^1 was used.

If the error detection check is only performed in the final round, then the error $\varepsilon = 1$ will be detected, but by this point, the adversary has already learned which witness was used, so the privacy of the honest parties' inputs are compromised. □

5 General Compiler for Malicious Security in Collaborative Proof Generation

In this section, we show a compiler that takes any collaborative proof generation algorithm that is secure against semi-honest adversaries and generically adds security against malicious adversaries in the honest-majority setting. This helps to mitigate pitfall 2 against the zkSNARKs with reactive proof generation. Following the approach in [CGH+18], the maliciously secure protocol runs two copies of the semi-honest protocol but randomizes the wire values in one of the copies. Then before opening any values, the protocol does a consistency check between the two copies to detect whether the adversary has introduced any errors. It is important to note that the consistency check must be done at the end of every round before opening the round's outputs, and the randomization value cannot be reused across rounds.

Let $\Pi = (\Pi_1, \ldots, \Pi_R)$ be an R-round protocol for a collaborative proof system that computes the Prove function. Each Π_i computes round i of the protocol. Π_i takes several inputs $(\mathsf{in}_1, \ldots, \mathsf{in}_I)$ and secret-shares them using $\mathcal{F}_{\text{input}}$. Then Π_i applies the following basic arithmetic operations on the secret-shared values – addition, scalar multiplication, and calls to $\mathcal{F}_{\text{mult}}$ – and may also make calls to a random oracle. Finally Π_i opens some secret-shared values $(\mathsf{out}_1, \ldots, \mathsf{out}_O)$ at the end of the round. Also let M be the number of multiplication gates in Π_i, and let gate $k \in [M]$ map inputs $[\mathsf{x}_k]$, $[\mathsf{y}_k]$ to output $[\mathsf{z}_k]$.

Let Π satisfy t-zero-knowledge for semi-honest adversaries. Then we will use Π to construct a new protocol $\Sigma = (\Sigma_1, \ldots, \Sigma_R)$ that is secure against malicious adversaries in the honest-majority setting. Σ works in the $(\mathcal{F}_{\text{input}}, \mathcal{F}_{\text{mult}}, \mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{checkZero}})$-hybrid model. Note that typical protocols for multiplying secret-shared values in the presence of semi-honest adversaries securely realize $\mathcal{F}_{\text{mult}}$ against malicious adversaries in the honest-majority setting.

Protocol Σ_i

1. **Input sharing:** For each input $j \in [I]$, run $\mathcal{F}_{\text{input}}$ on in_j to generate the shares $[\mathsf{in}_j]$ and distribute them among the parties.

2. **Randomize the inputs:** Call $\mathcal{F}_{\text{rand}}$ to obtain $[\mathsf{r}]$ for $\mathsf{r} \xleftarrow{\$} \mathbb{F}$. Then compute:

$$[\mathsf{rin}_j] = \mathcal{F}_{\text{mult}}([\mathsf{r}], [\mathsf{in}_j])$$

3. Σ_i follows in order the steps of Π_i that compute addition, scalar multiplication, calls to $\mathcal{F}_{\text{mult}}$, and calls to the random oracle. For each intermediate variable $[\mathsf{z}]$ that Π_i computes, Σ_i computes $[\mathsf{z}]$ and $[\mathsf{rz}]$ as follows:

   (a) **Addition:** Whenever Π_i adds two secret-shared values $[\mathsf{x}]$ and $[\mathsf{y}]$, Σ_i computes

$$[\mathsf{z}] = [\mathsf{x}] + [\mathsf{y}]$$
$$[\mathsf{rz}] = [\mathsf{rx}] + [\mathsf{ry}]$$

Note that Σ_i has computed $[\mathsf{rx}]$ and $[\mathsf{ry}]$ from previous steps.

(b) **Scalar Multiplication:** Whenever Π_i multiplies a secret-shared value $[\mathsf{x}]$ by a public scalar s, Σ_i computes:

$$[\mathsf{z}] = s \cdot [\mathsf{x}]$$
$$[\mathsf{rz}] = s \cdot [\mathsf{rx}]$$

(c) **Calls to $\mathcal{F}_{\text{mult}}$:** For each multiplication gate $k \in [M]$, Σ_i has computed $\mathsf{x}_k, \mathsf{y}_k, \mathsf{rx}_k$ from prior steps. Then Σ_i computes the following:

$$[\mathsf{z}_k] = \mathcal{F}_{\text{mult}}([\mathsf{x}_k], [\mathsf{y}_k])$$
$$[\mathsf{rz}_k] = \mathcal{F}_{\text{mult}}([\mathsf{rx}_k], [\mathsf{y}_k])$$

(d) **Calls to the random oracle:** Σ_i computes the calls to the random oracle as prescribed by Π_i.

4. **Consistency Check:** After Σ_i has completed all the addition, multiplication, and scalar multiplication steps of Π_i but before opening the outputs, Σ_i does the following consistency check.

(a) Call $\mathcal{F}_{\text{coin}}$ $I + M$ times to obtain $[\alpha_j]_{j\in[I]}$ and $[\beta_k]_{k\in[M]}$.

(b) Compute $r = \mathsf{open}([r])$.

(c) Compute:

$$[u] := \sum_{j\in[I]} \alpha_j \cdot ([\mathsf{rin}_j] - \mathsf{r} \cdot [\mathsf{in}_j]) + \sum_{k\in[M]} \beta_k \cdot ([\mathsf{rz}_k] - \mathsf{r} \cdot [\mathsf{z}_k])$$

(d) Compute $\mathcal{F}_{\text{checkZero}}([u])$. If the result is reject ($u \neq 0$), then output $\perp$ and abort the computation. Otherwise, continue.

5. **Open:** For every output $\ell \in [O]$, compute $\mathsf{out}_\ell = \mathsf{open}([\mathsf{out}_\ell])$ and output it.

Theorem 5.1. *If Π is a collaborative proof generation for a relation $\mathfrak{R}$, against $t < N/2$ semi-honest provers, then Σ is a collaborative proof generation for a relation $\mathfrak{R}$ against t malicious provers in the ($\mathcal{F}_{input}$,$\mathcal{F}_{mult}$, $\mathcal{F}_{rand}$, $\mathcal{F}_{coin}$, $\mathcal{F}_{checkZero}$)-hybrid model.*

Proof. The completness, knowledge soundness and succinctness of Σ follow directly from the corresponding properties of Π. We now prove the t-zero-kowledge property for Σ against t corruptions, by first describing the simulator as follows.

The Simulator: Let $\mathcal{A}$ be an efficient malicious adversary controlling $k \leq t$ provers. The only difference between malicious and semi-honest adversaries in this setting is that malicious adversaries can specify an additive error ε whenever $\mathcal{F}_{\text{mult}}$ is called. Let $\mathcal{A}'$ be a semi-honest adversary that behaves the same as the malicious adversary $\mathcal{A}$, except that $\mathcal{A}'$ doesn't output any additive errors ε. We can construct $\mathcal{A}'$ by having it run $\mathcal{A}$ internally and ignore any ε-values that $\mathcal{A}$ produces.

Next, since Π satisfies t-zero-knowledge against semi-honest adversaries, that means that there exists a simulator Sim_Π that simulates the view of $\mathcal{A}'$ in Π.

Now let us construct the simulator Sim_Σ for protocol Σ:

Simulator Sim_Σ

Setup:

1. Sim_Σ receives as input the public parameters pp, the instance $\mathcal{X}$, the corrupted provers' witness fragments $(\mathsf{wtn}_{i_1}, \ldots, \mathsf{wtn}_{i_k})$, a bit b indicating whether the combined witness is accepting or rejecting, and query access to a random oracle H.
2. Sim_Σ runs $\mathsf{Sim}_\Pi^H(\mathsf{pp}, \mathcal{X}, \mathsf{wtn}_{i_1}, \ldots, \mathsf{wtn}_{i_k}, b)$ until it outputs (tr', μ).
3. Sim_Σ initializes $\mathcal{A}$ on $(\mathsf{pp}, \mathcal{X}, \mathsf{wtn}_{i_1}, \ldots, \mathsf{wtn}_{i_k})$ and runs $\mathcal{A}$ internally throughout the simulation.

Main Protocol: For each round $i \in [R]$, Sim_Σ simulates Σ_i as follows:

1. **Input sharing:** Sim_Σ receives from $\mathcal{A}$ the corrupted parties' inputs as well as the corrupted parties' shares of every input: $([\mathsf{in}_1]_C, \ldots, [\mathsf{in}_I]_C)$. Sim_Σ stores these values and sends $([\mathsf{in}_1]_C, \ldots, [\mathsf{in}_I]_C)$ back to the adversary.
2. $\mathcal{F}_{\mathbf{rand}}$**:** Sim_Σ receives from $\mathcal{A}$ the corrupted parties' shares $\{\rho_j\}_{\forall j \in C}$ of $[r]$ and stores these values. Sim_Σ samples $r \leftarrow \mathbb{F}$ and computes $[r] = \mathsf{share}(r, \{\rho_j\}_{\forall j \in C})$.
3. Sim_Σ simulates each step of Σ_i that involves addition, scalar multiplication, calls to $\mathcal{F}_{\mathrm{mult}}$, or calls to the random oracle. At each step, Sim_Σ computes the shares of the result that the corrupted parties would hold if they followed the protocol honestly.
   (a) **Calls to** $\mathcal{F}_{\mathbf{mult}}$**:** To multiply two values (a, b), Sim_Σ sends to $\mathcal{A}$ the corrupted parties' shares $[a]_C$ and $[b]_C$, which the simulator has computed from prior steps. Next, $\mathcal{A}$ sends to Sim_Σ the additive error ε as well as the corrupted parties' shares of the output $[c]_C$. Sim_Σ stores these values.
4. **Consistency Check:**
   (a) To simulate a call to $\mathcal{F}_{\mathrm{coin}}$, Sim_Σ samples a random value $v \in \mathbb{F}$ uniformly at random and sends it to $\mathcal{A}$.
   (b) $\mathsf{open}([r])$**:** Sim_Σ publishes $[r]_{\mathcal{H}}$ and receives from C the corrupted parties' shares of r. If the corrupted parties' shares do not match $[r]_C$ that were computed earlier, then Sim_Σ halts and outputs $\perp$.
   (c) If on any call to $\mathcal{F}_{\mathrm{mult}}$ during Σ_i, the adversary sent an additive error $\varepsilon \neq 0$, then Sim_Σ sets $u \neq 0$. Otherwise, Sim_Σ sets $u = 0$.
   (d) $\mathcal{F}_{\mathbf{checkZero}}([u])$**:** If $u = 0$, then Sim_Σ sends 0 to $\mathcal{A}$. $\mathcal{A}$ responds with accept or reject, and Sim_Σ forwards $\mathcal{A}$'s response to the honest parties.
   If $u \neq 0$, then:
      i. With probability $\frac{1}{|\mathbb{F}|}$, Sim_Σ sends accept to all parties.
      ii. With probability $1 - \frac{1}{|\mathbb{F}|}$, Sim_Σ sends reject to all parties.
   (e) If the result of $\mathcal{F}_{\mathrm{checkZero}}([u])$ is reject, then Sim_Σ outputs $\perp$ and halts. Otherwise, Sim_Σ continues.
5. **Open:** To open the outputs of round i, Sim_Σ uses the following procedure. For each $\ell \in [O]$:

(a) Sim_Σ has already computed the shares of $[\mathsf{out}_\ell]$ that the corrupted parties would hold if they followed the protocol honestly. Let us call those shares $(\alpha_j)_{\forall j \in C}$. Next, Sim_Σ reads the plaintext value out_ℓ from tr'.

(b) Sim_Σ computes:

$$\left(\alpha_j\right)_{\forall j \in [N]} = \mathsf{share}(\mathsf{out}_\ell, (\alpha_j)_{\forall j \in C})$$

(c) Sim_Σ sends $\left(\alpha_j\right)_{j \in \mathcal{H}}$ to $\mathcal{A}$, and $\mathcal{A}$ sends the corrupted parties' shares to Sim_Σ. If any of the corrupted parties' shares do not match $\left(\alpha_j\right)_{j \in C}$, then Sim_Σ outputs $\perp$ and halts.

6. If Sim_Σ reaches the end of the simulation of Σ without halting, then Sim_Σ outputs (tr', μ).

Analysis: If the adversary introduces a non-zero additive error during some call to $\mathcal{F}_{\text{mult}}$ during some round i, then with overwhelming probability, $u \neq 0$. This follows from lemma 5.1. Next, if $u \neq 0$, then with overwhelming probability, $\mathcal{F}_{\text{checkZero}}([u])$ outputs reject, and Σ_i will be aborted before the outputs of round i are opened. The simulator simulates this behavior correctly.

Next, if the adversary does not introduce any additive errors, then Sim_Σ correctly simulates the view of the adversary.

Let us imagine running protocol Π with adversary $\mathcal{A}'$, which runs $\mathcal{A}$ internally, and ignores any ε values that $\mathcal{A}$ produces. Let us condition on the event that $\mathcal{A}$ outputs $\varepsilon = 0$ for every call to $\mathcal{F}_{\text{mult}}$. Then the view of $\mathcal{A}$ is the same as if $\mathcal{A}$ were running in protocol Σ.

The output of Sim_Σ, conditioned on the event that the output is not $\perp$ is the view of $\mathcal{A}$ running in protocol Σ, conditioned on the event that $\mathcal{A}$ outputs $\varepsilon = 0$ for every call to $\mathcal{F}_{\text{mult}}$.

Lemma 5.1. *If the adversary introduces a non-zero additive error during any call to $\mathcal{F}_{mult}$ in Σ_i, then with overwhelming probability, $u \neq 0$.*

Proof. Let us define the additive errors that the adversary can introduce in Σ_i. For each $j \in [I]$, let $\varepsilon_{j,1}$ be the error introduced during $\mathcal{F}_{\text{mult}}([\mathsf{r}], [\mathsf{in}_j])$. Then:

$$\mathsf{rin}_j = \mathsf{r} \cdot \mathsf{in}_j + \varepsilon_{j,1}$$

Next, for each $k \in [M]$, let $\varepsilon_{k,2}$ be the error introduced during $\mathcal{F}_{\text{mult}}([\mathsf{x}_k], [\mathsf{y}_k])$, and let $\varepsilon_{k,3}$ be the error introduced during $\mathcal{F}_{\text{mult}}([\mathsf{rx}_k], [\mathsf{y}_k])$. Then:

$$\mathsf{z}_k = \mathsf{x}_k \cdot \mathsf{y}_k + \varepsilon_{k,2}$$
$$\mathsf{rz}_k = \mathsf{rx}_k \cdot \mathsf{y}_k + \varepsilon_{k,3}$$

Note that all of the additive errors are chosen independently of r because they are chosen before $[\mathsf{r}]$ is opened, and the perfect secrecy of the secret-sharing scheme hides r.

Lemma 5.2 shows that if for some $j \in [I]$, $\varepsilon_{j,1} \neq 0$, then with overwhelming probability $u \neq 0$. This is because $\varepsilon_{j,1} = \mathsf{rin}_j - \mathsf{r} \cdot \mathsf{in}_j$.

Next, let us assume that $\mathsf{rx}_k = \mathsf{r} \cdot \mathsf{x}_k$ for all $k \in [M]$ because otherwise, $u \neq 0$ with overwhelming probability. Then $\mathsf{rz}_k - \mathsf{r} \cdot \mathsf{z}_k$ has the following simple description:

$$\begin{aligned}\mathsf{rz}_k - \mathsf{r} \cdot \mathsf{z}_k &= \mathsf{rx}_k \cdot \mathsf{y}_k + \varepsilon_{k,3} - \mathsf{r} \cdot \mathsf{x}_k \cdot \mathsf{y}_k - \mathsf{r} \cdot \varepsilon_{k,2} \\ &= \mathsf{r} \cdot \mathsf{x}_k \cdot \mathsf{y}_k + \varepsilon_{k,3} - \mathsf{r} \cdot \mathsf{x}_k \cdot \mathsf{y}_k - \mathsf{r} \cdot \varepsilon_{k,2} \\ &= \varepsilon_{k,3} - \mathsf{r} \cdot \varepsilon_{k,2}\end{aligned}$$

If $\varepsilon_{k,2} \neq 0$ or $\varepsilon_{k,3} \neq 0$, then with overwhelming probability over r,

$$\varepsilon_{k,3} - \mathsf{r} \cdot \varepsilon_{k,2} \neq 0$$

In this case, $\mathsf{rz}_k - \mathsf{r} \cdot \mathsf{z}_k \neq 0$, and by lemma 5.2, $u \neq 0$ with overwhelming probability. □

Lemma 5.2. *If there exists an input $j \in [I]$ such that $\mathsf{rin}_j \neq \mathsf{r} \cdot \mathsf{in}_j$ or if there exists a multiplication gate $k \in [M]$ such that $\mathsf{rx}_k = \mathsf{r} \cdot \mathsf{x}_k$ or $\mathsf{rz}_k = \mathsf{r} \cdot \mathsf{z}_k$, then with overwhelming probability over the randomness of the α_j- and β_k-values, $u \neq 0$.*

Proof. We will prove this inductively. Every input to round i satisfies the invariant $\mathsf{rin}_j = \mathsf{r} \cdot \mathsf{in}_j$ or else $u \neq 0$ with overwhelming probability. Next, for every intermediate gate (addition, scalar multiplication, and $\mathcal{F}_{\text{mult}}$), if the inputs satisfy the invariant $\mathsf{rx} = \mathsf{r} \cdot \mathsf{x}$ and $\mathsf{ry} = \mathsf{r} \cdot \mathsf{y}$, then the outputs do too – $\mathsf{rz} = \mathsf{r} \cdot \mathsf{z}$ – or else $u \neq 0$ with overwhelming probability. Then by induction, the inputs and outputs to every $\mathcal{F}_{\text{mult}}$ gate satisfy the invariant: $\mathsf{rx}_k = \mathsf{r} \cdot \mathsf{x}_k$ and $\mathsf{rz}_k = \mathsf{r} \cdot \mathsf{z}_k$ or else $u \neq 0$ with overwhelming probability.

Base Case: If for any $j \in [I]$, $\mathsf{rin}_j \neq \mathsf{r} \cdot \mathsf{in}_j$, then with overwhelming probability, $u \neq 0$. This is because $\alpha_j \cdot (\mathsf{rin}_j - \mathsf{r} \cdot \mathsf{in}_j)$ is the only term of u that depends on α_j, and the coefficient of α_j – $(\mathsf{rin}_j - \mathsf{r} \cdot \mathsf{in}_j)$ – is non-zero. Then with overwhelming probability over the randomness of α_j, $u \neq 0$.

Inductive Case ($\mathcal{F}_{\text{mult}}$): Likewise, for any multiplication gate $k \in [M]$, if the output does not satisfy the invariant – $\mathsf{rz}_k \neq \mathsf{r} \cdot \mathsf{z}_k$ – then with overwhelming probability, $u \neq 0$. This is because $\beta_k \cdot (\mathsf{rz}_k - \mathsf{r} \cdot \mathsf{z}_k)$ is the only term of u that depends on β_k, and the coefficient of β_k is non-zero. Then with overwhelming probability over the randomness of β_k, $u \neq 0$.

Inductive Case (Addition and Scalar Multiplication): For any addition gate, if $\mathsf{rx} = \mathsf{r} \cdot \mathsf{x}$ and $\mathsf{ry} = \mathsf{r} \cdot \mathsf{y}$, then

$$\begin{aligned} \mathsf{rz} &= \mathsf{r} \cdot (\mathsf{x} + \mathsf{y}) \\ &= \mathsf{r} \cdot \mathsf{z} \end{aligned}$$

For any scalar multiplication gate, if $\mathsf{rx} = \mathsf{r} \cdot \mathsf{x}$, then

$$\begin{aligned} \mathsf{rz} &= s \cdot \mathsf{r} \cdot \mathsf{x} \\ &= \mathsf{r} \cdot \mathsf{z} \end{aligned}$$

By induction, the inputs and outputs of every $\mathcal{F}_{\text{mult}}$ must satisfy the invariant – $\mathsf{rx}_k = \mathsf{r} \cdot \mathsf{x}_k$ and $\mathsf{rz}_k = \mathsf{r} \cdot \mathsf{z}_k$ – or else $u \neq 0$ with overwhelming probability. □

□

6 Collaborative zk-SNARK Based On Bulletproofs

In this section, we describe and prove that the semi-honest protocol for collaborative proof generation based on Bulletproofs [BBB+18] is, in fact, secure against malicious provers, in the honest majority setting. We begin by describing some notations used in this section and describe the collaborative zk-SNARK protocol for Bulletproofs in section 6.1 using standard MPC functionalities from section 3.1. We prove the malicious security of this protocol in theorem 6.1.

Notation. For the underlying Bulletproofs zk-SNARK protocol, we refer the reader to [BBB+18, Section 5]. We use the same notations in our collaborative zk-SNARK protocol, which are described as follows. The inputs to the Bulletproofs protocol include an instance $\mathcal{X}$ and witness wtn:

Common Preprocessed Inputs: The following group elements are sampled independently and uniformly at random: $g, h \in \mathbb{G}$; $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n$.
Instance $\mathcal{X}$ (held by the provers and $\mathcal{V}$): $\mathbf{V} \in \mathbb{G}^m$; $\mathbf{W}_L, \mathbf{W}_R, \mathbf{W}_O \in \mathbb{Z}_p^{Q \times n}$; $\mathbf{W}_V \in \mathbb{Z}_p^{Q \times m}$; $\mathbf{c} \in \mathbb{Z}_p^Q$.
Witness wtn (distributed among the provers): $\mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O \in \mathbb{Z}_p^n$; $\mathbf{v}, \boldsymbol{\gamma} \in \mathbb{Z}_p^m$
Relation $\mathfrak{R}_{\text{BP}}$: $\{(\mathcal{X}, \text{wtn}) : V_j = g^{v_j} h^{\gamma_j} \forall j \in [m] \wedge \mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O \wedge (\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O = \mathbf{W}_V \cdot \mathbf{v} + \mathbf{c})\}$
Collaborative Extended Witness Generation. Recall that in collaborative zk-SNARKs, each prover holds a witness fragment wtn_i, which needs to be compiled into a valid witness wtn. We

formally define the following functionality for witness extension below. This functionality can be realized using maliciously secure MPC techniques, the details of which are beyond the scope of this work.

Functionality $\mathcal{F}_{\text{BP-WE}}$: Bulletproofs Witness Extension

1: *Inputs:* $\mathcal{F}_{\text{BP-WE}}$ receives the instance $\mathcal{X}$ and each party's witness fragment $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$. It also receives from the adversary the corrupted parties' shares of the combined witness: $[\mathsf{wtn}]_C$.

2: The functionality verifies the witness by checking that $V(\mathcal{X}, (\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)) = 1$. If verification fails, the functionality sends $\perp$ to all parties and halts. Otherwise, the functionality continues.

3: The functionality uses $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$ to compute the combined witness $\mathsf{wtn} = (\mathbf{a}_L, \mathbf{a}_R, \mathbf{a}_O, \boldsymbol{\gamma})$ and the sharing of wtn:

$$[\mathsf{wtn}] = \mathsf{share}(\mathsf{wtn}, [\mathsf{wtn}]_C)$$

4: The functionality sends each party $j \in \{1, \ldots, N\}$ their share $[\mathsf{wtn}]_j$.

6.1 The Collaborative Bulletproof Protocol

In this section, we describe the collaborative zk-SNARK protocol for Bulletproofs. The protocol begins by running the witness extension by invoking $\mathcal{F}_{\text{BP-WE}}$, and then generates the proof of Bulletproof from [BBB+18, Section 5], by running a combination of two MPC functionalities: $\mathcal{F}_{\text{rand}}$ and $\mathcal{F}_{\text{mult}}$. We describe the four rounds of the protocol in detail below.

Round 1 : The Bulletproof prover computes Pedersen commitments to the vector of left inputs $\mathbf{a}_L$, right inputs $\mathbf{a}_R$, and the outputs $\mathbf{a}_O$. The prover chooses binding vectors $\mathbf{s}_L$ and $\mathbf{s}_R$ and generates their Pedersen commitments. In collaborative Bulletproof, all these computations are done locally on the shares of these vectors by the provers.

Round 1: The prover parties do the following:

1: **Witness Extension:** Send $\mathcal{X}$ and $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$ to $\mathcal{F}_{\text{BP-WE}}$. If $\mathcal{F}_{\text{BP-WE}}$ outputs $\perp$, then abort the protocol. Otherwise, $\mathcal{F}_{\text{BP-WE}}$ outputs: $[\mathbf{a}_L]$, $[\mathbf{a}_R]$, $[\mathbf{a}_O]$, $[\boldsymbol{\gamma}]$

2: Call $\mathcal{F}_{\text{rand}}$ $2n + 3$ times to obtain $[\alpha]$, $[\beta]$, $[\rho]$, $[\mathbf{s}_L]$, $[\mathbf{s}_R]$, where $\alpha, \beta, \rho \in \mathbb{Z}_p$ and $\mathbf{s}_L, \mathbf{s}_R \in \mathbb{Z}_p^n$.

3: Compute:

$$[A_I] = h^{[\alpha]} \cdot \prod_{i=1}^{n} \mathbf{g}_i^{[(\mathbf{a}_L)_i]} \cdot \prod_{i=1}^{n} \mathbf{h}_i^{[(\mathbf{a}_R)_i]}; \quad [A_O] = h^{[\beta]} \cdot \prod_{i=1}^{n} \mathbf{g}_i^{[(\mathbf{a}_O)_i]}$$

$$[S] = h^{[\rho]} \cdot \prod_{i=1}^{n} \mathbf{g}_i^{[(\mathbf{s}_L)_i]} \cdot \prod_{i=1}^{n} \mathbf{h}_i^{[(\mathbf{s}_R)_i]}$$

4: Run open on $[A_I]$, $[A_O]$, $[S]$ to obtain (A_I, A_O, S) and send it to $\mathcal{V}$.

Round 2 : The Bulletproof prover computes polynomials $\ell(X)$, $\mathsf{r}(X)$ and $\mathsf{t}(X)$ along with their commitments. These polynomials contain the input and output wire values such that in Rounds 3 and 4, the goal of the prover reduces to proving that $\ell(X)$, $\mathsf{r}(X)$, $\mathsf{t}(X)$ are well-formed and that $\langle \ell(X), \mathsf{r}(X) \rangle = \mathsf{t}(X)$. In the collaborative Bulletproofs protocol, these computations can all be done with invocations to $\mathcal{F}_{\text{mult}}$ and local computations by the provers.

Round 2: $\mathcal{V}$ samples $y, z \xleftarrow{\$} \mathbb{Z}_p^*$ independently and uniformly at random and sends (y, z) to the provers. Then the provers do the following:

1: Compute:

$$\mathbf{y}^n = (1, y, y^2, \ldots, y^{n-1}) \in \mathbb{Z}_p^n$$
$$\mathbf{y}^{-n} = (1, y^{-1}, y^{-2}, \ldots, y^{-(n-1)}) \in \mathbb{Z}_p^n$$
$$\mathbf{z}_{[1:]}^{Q+1} = (z, z^2, \ldots, z^Q) \in \mathbb{Z}_p^Q$$
$$\delta(y, z) = \langle \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R), \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L \rangle$$

2: Compute:

$$[\mathbf{l}_1] = [\mathbf{a}_L] + \mathbf{y}^{-n} \circ (\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R); \quad [\mathbf{l}_2] = [\mathbf{a}_O]; \quad [\mathbf{l}_3] = [\mathbf{s}_L]$$
$$[\ell(X)] = [\mathbf{l}_1] \cdot X + [\mathbf{l}_2] \cdot X^2 + [\mathbf{l}_3] \cdot X^3$$
$$\mathbf{r}_0 = -\mathbf{y}^n + \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O; [\mathbf{r}_1] = \mathbf{y}^n \circ [\mathbf{a}_R] + \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L; [\mathbf{r}_3] = \mathbf{y}^n \circ [\mathbf{s}_R]$$
$$[\mathrm{r}(X)] = \mathbf{r}_0 + [\mathbf{r}_1] \cdot X + [\mathbf{r}_3] \cdot X^3$$

$$[\mathbf{w}] = \mathbf{W}_L \cdot [\mathbf{a}_L] + \mathbf{W}_R \cdot [\mathbf{a}_R] + \mathbf{W}_O \cdot [\mathbf{a}_O]$$

$$[t_1] = \sum_{i=1}^{n} [(\mathbf{l}_1)_i] \cdot (\mathbf{r}_0)_i; \quad [t_2] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_1)_i], [(\mathbf{r}_1)_i]\big) + [(\mathbf{l}_2)_i] \cdot (\mathbf{r}_0)_i$$
$$[t_3] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_2)_i], [(\mathbf{r}_1)_i]\big) + [(\mathbf{l}_3)_i] \cdot (\mathbf{r}_0)_i$$
$$[t_4] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_1)_i], [(\mathbf{r}_3)_i]\big) + \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_3)_i], [(\mathbf{r}_1)_i]\big)$$
$$[t_5] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_2)_i], [(\mathbf{r}_3)_i]\big); \quad [t_6] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l}_3)_i], [(\mathbf{r}_3)_i]\big)$$
$$[\mathrm{t}(X)] = \sum_{i=1}^{6} [t_i] \cdot X^i$$

3: Call $\mathcal{F}_{\text{rand}}$ 5 times to obtain $([\tau_1], [\tau_3], [\tau_4], [\tau_5], [\tau_6])$, where $\tau_i \in \mathbb{Z}_p, \forall i \in [6]$, and compute:

$$[T_1] = g^{[t_1]} \cdot h^{[\tau_1]}; \quad [T_3] = g^{[t_3]} \cdot h^{[\tau_3]}; \quad [T_4] = g^{[t_4]} \cdot h^{[\tau_4]}$$
$$[T_5] = g^{[t_5]} \cdot h^{[\tau_5]}; \quad [T_6] = g^{[t_6]} \cdot h^{[\tau_6]}$$

4: Call open on $([T_1], [T_3], [T_4], [T_5], [T_6])$ and send $(T_1, T_3, T_4, T_5, T_6)$ to $\mathcal{V}$.

Round 3 : The Bulletproof prover computes the evaluations of $\ell(X)$ and $\mathrm{r}(X)$ at the random point x sent by $\mathcal{V}$, the inner product of these evaluations, $\hat{t}$, and the blinding values that the verifier needs to check the commitments to all these three values. In collaborative Bulletproofs, this requires invoking $\mathcal{F}_{\text{mult}}$ and some local computations by the provers.

Round 3: $\mathcal{V}$ samples $x \xleftarrow{\$} \mathbb{Z}_p^*$ and sends x to the provers. Then the provers do the following:

1: Compute:

$$[\mathbf{l}] = [\ell(x)] = [\mathbf{l}_1] \cdot x + [\mathbf{l}_2] \cdot x^2 + [\mathbf{l}_3] \cdot x^3;$$

$$[\mathbf{r}] = [\mathrm{r}(x)] = \mathbf{r}_0 + [\mathbf{r}_1] \cdot x + [\mathbf{r}_3] \cdot x^3; \quad [\hat{t}] = \sum_{i=1}^{n} \mathcal{F}_{\text{mult}}\big([(\mathbf{l})_i], [(\mathbf{r})_i]\big)$$

$$[\tau_x] = \sum_{i \in \{1,\ldots,6\} \setminus \{2\}} [\tau_i] \cdot x^i + x^2 \cdot \sum_{i=1}^{Q} z^i \cdot (\mathbf{W}_V \cdot [\boldsymbol{\gamma}])_i;$$

$$[\mu] = [\alpha] \cdot x + [\beta] \cdot x^2 + [\rho] \cdot x^3$$

2: Run open on $([\tau_x], [\mu], [\hat{t}], [\mathbf{l}], [\mathbf{r}])$ and sends $(\tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r})$ to $\mathcal{V}$.

Note that the output of round 3 is not succinct, but we can make it succinct using the inner product argument from Bulletproofs. Looking ahead, our zero knowledge simulator can simply compute the inner product argument directly, as it knows $\mathbf{l}$ and $\mathbf{r}$.

Round 4 is the same in Bulletproofs and collaborative Bulletproofs. $\mathcal{V}$ verifies the correctness of the inner product computed by the prover.

Round 4: $\mathcal{V}$ verifies the proof according to the procedure in [BBB+18, Protocol 3], lines 83-94. In the end, $\mathcal{V}$ outputs accept or reject.

The Final Protocol: The interactive protocol described above can be converted to a non-interactive protocol in the random oracle model. In particular, to make random oracle queries on parts of the transcript, the provers reconstruct the shares of the transcript and query the random oracle locally.

6.2 Malicious Security of Collaborative Bulletproofs

Theorem 6.1. *The final protocol described in section 6.1 is a collaborative proof generation based on Bulletproofs, i.e., for relation $\mathfrak{R}_{\text{BP}}$, against $t < N/2$ malicious provers in the $(\mathcal{F}_{\text{BP-WE}}, \mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{mult}})$-hybrid model.*

Proof. The completeness, knowledge soundness, and succinctness properties directly follow from the corresponding properties of Bulletproofs.

We will prove the t-zero-knowledge property for the protocol, in the $(\mathcal{F}_{\text{BP-WE}}, \mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{mult}})$-hybrid model against a malicious adversary that controls at most $t < N/2$ provers. We show that even though this protocol is based on semi-honest techniques that allow a malicious adversary to introduce additive errors, the additive errors turn out to be simulatable.

The Simulator. We begin by describing the simulator $\mathcal{S}$, based on the zero-knowledge simulator in [BBB+18, Appendix D]. $\mathcal{S}$ simultates the malicious adversary's view of the collaborative Bulletproofs protocol.

Simulator $\mathcal{S}$

1. **Round 1:**

(a) **Inputs:** $\mathcal{S}$ receives from $\mathcal{A}$ the corrupted parties inputs $(\mathsf{wtn}_j)_{j \in C}$ as well as the corrupted parties' shares of the combined witness $([\mathbf{a}_L]_C, [\mathbf{a}_R]_C, [\mathbf{a}_O]_C, [\boldsymbol{\gamma}]_C)$. $\mathcal{S}$ also receives from the t-zero-knowledge challenger a bit $b \leftarrow V_{\mathfrak{R}}(X, \mathsf{wtn})$ indicating whether the witness is valid. $\mathcal{S}$ stores all the inputs for later.

(b) **Witness Extension:** If $b = 0$ (meaning the witness is invalid), then $\mathcal{S}$ sends

$\perp$ to all parties and halts. Otherwise, $\mathcal{S}$ continues and sends to $\mathcal{A}$ the shares $([\mathbf{a}_L]_C, [\mathbf{a}_R]_C, [\mathbf{a}_O]_C, [\boldsymbol{\gamma}]_C)$.

(c) **Calls to $\mathcal{F}_{\text{rand}}$:** Whenever Round 1 calls $\mathcal{F}_{\text{rand}}$, $\mathcal{S}$ does the following: $\mathcal{S}$ receives from $\mathcal{A}$ the values $(r_j)_{j\in C}$, which represent the corrupted parties' shares of the random value, and $\mathcal{S}$ stores these shares.

(d) **Local Computations:** The protocol for computing $([A_I], [A_O], [S])$ in Round 1 does not require communication among the parties. Such computations are called *local*. To simulate these local computations, $\mathcal{S}$ simply computes the shares of $([A_I], [A_O], [S])$ that the corrupted parties would hold if they followed the protocol honestly. $\mathcal{S}$ can do this because they have the corrupted parties' shares of $([\mathbf{a}_L], [\mathbf{a}_R], [\mathbf{a}_O], [\alpha], [\beta], [\rho], [\mathbf{s}_L], [\mathbf{s}_R])$, which are the inputs to the local computation.

(e) **Output:** $\mathcal{S}$ samples: $(x, y, z) \xleftarrow{\$} \mathbb{Z}_p^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $(A_I, A_O) \xleftarrow{\$} \mathbb{G} \times \mathbb{G}$, $(\mathbf{l}, \mathbf{r}) \xleftarrow{\$} \mathbb{Z}_p^n \times \mathbb{Z}_p^n$ and $\mu \xleftarrow{\$} \mathbb{Z}_p$ and computes:

$$S = \Big(A_I^{-x} \cdot A_O^{-x^2} \cdot h^{\mu} \cdot \mathbf{g}^{\mathbf{l}-x\cdot\mathbf{y}^{-n}\circ\left(\mathbf{z}_{[1:]}^{Q+1}\cdot\mathbf{W}_R\right)} \cdot \mathbf{h}^{\mathbf{y}^{-n}\circ\left(\mathbf{r}+\mathbf{y}^n-\mathbf{z}_{[1:]}^{Q+1}\cdot(x\cdot\mathbf{W}_L+\mathbf{W}_O)\right)}\Big)^{(x^{-3})} \tag{1}$$

(f) **Opening:** $\mathcal{S}$ opens $[A_I]$ to A_I using the following procedure.

i. $\mathcal{S}$ has already computed the shares of $[A_I]$ that the corrupted parties would hold if they followed the protocol honestly. Let us call those shares $(\alpha_j)_{j\in C}$.

ii. $\mathcal{S}$ computes:
$$\left(\alpha_j\right)_{j\in[N]} = \mathsf{share}(A_I, (\alpha_j)_{j\in C})$$

iii. $\mathcal{S}$ sends $\left(\alpha_j\right)_{j\in\mathcal{H}}$ to $\mathcal{A}$, and $\mathcal{A}$ sends the corrupted parties' shares to $\mathcal{S}$. If any of the corrupted parties' shares do not match $\left(\alpha_j\right)_{j\in C}$, then $\mathcal{S}$ outputs abort and aborts the protocol.

(g) $\mathcal{S}$ also opens $[A_O]$ to A_O and $[S]$ to S using essentially the same procedure as the one for A_I.

2. **Round 2:**

(a) $\mathcal{S}$ outputs (y, z) on behalf of the verifier.

(b) $\mathcal{S}$ follows the steps of Round 2 and computes the shares of each variable that the corrupted party would hold if they followed the protocol honestly. $\mathcal{S}$ handles local computations and calls to $\mathcal{F}_{\text{rand}}$ the same way it did for round 1. The local computations include the computation of $(\mathbf{y}^n, \mathbf{z}_{[1:]}^{Q+1}, \delta(y,z), [\ell(X)]_C, [\mathrm{r}(X)]_C, [\mathbf{w}]_C, [T_1]_C, [T_3]_C, [T_4]_C, [T_5]_C, [T_6]_C)$.

(c) **Calls to $\mathcal{F}_{\text{mult}}$:**

i. Whenever Round 2 calls $\mathcal{F}_{\text{mult}}$, $\mathcal{S}$ does the following: The call has the form $\mathcal{F}_{\text{mult}}\big([(\mathbf{l}_j)_i], [(\mathbf{r}_k)_i]\big)$. Then $\mathcal{S}$ sends to $\mathcal{A}$ the corrupted parties' shares of the inputs $[(\mathbf{l}_j)_i]_C$ and $[(\mathbf{r}_k)_i]_C$, which the simulator has computed from prior steps. Next, $\mathcal{A}$ sends to $\mathcal{S}$ the additive error $\varepsilon_{i,j,k}$ as well as the corrupted parties' shares of the output $[\alpha]_C$. $\mathcal{S}$ stores these values.

ii. $\mathcal{S}$ computes

$$\varepsilon_2 = \sum_{i\in[n]} \varepsilon_{i,1,1}, \qquad \varepsilon_3 = \sum_{i\in[n]} \varepsilon_{i,2,1}, \qquad \varepsilon_4 = \sum_{i\in[n]} \varepsilon_{i,1,3} + \varepsilon_{i,3,1},$$
$$\varepsilon_5 = \sum_{i\in[n]} \varepsilon_{i,2,3}, \qquad \varepsilon_6 = \sum_{i\in[n]} \varepsilon_{i,3,3}$$

(d) $\mathcal{S}$ samples: $(T_3, T_4, T_5, T_6) \xleftarrow{\$} \mathbb{G}\times\mathbb{G}\times\mathbb{G}\times\mathbb{G}$ and $\tau_x \xleftarrow{\$} \mathbb{Z}_p$ and computes:

$$T_1 = \left(h^{\tau_x} \cdot g^{\langle \mathbf{l},\mathbf{r}\rangle - x^2\cdot\left(\delta(y,z)+\langle \mathbf{z}^{Q+1}_{[1:]},\mathbf{c}\rangle\right)} \cdot \mathbf{V}^{-x^2\cdot\left(\mathbf{z}^{Q+1}_{[1:]}\cdot\mathbf{W}_V\right)} \cdot \prod_{i=3}^{6}(T_i\cdot g^{-\varepsilon_i})^{-(x^i)}\right)^{(x^{-1})} \tag{2}$$

(e) $\mathcal{S}$ opens $([T_1],[T_3],[T_4],[T_5],[T_6])$ to (T_1,T_3,T_4,T_5,T_6) using the **Opening** procedure from the simulation of round 1.

3. **Round 3:**

(a) $\mathcal{S}$ outputs x on behalf of the verifier.

(b) $\mathcal{S}$ handles local computations the same way it did for Rounds 1 and 2, which includes the computation of $([\mathbf{l}]_C, [\mathbf{r}]_C, [\tau_x]_C, [\mu]_C)$.

(c) **Calls to $\mathcal{F}_{\text{mult}}$:** For each call to $\mathcal{F}_{\text{mult}}$, which has the form $\mathcal{F}_{\text{mult}}([(\mathbf{l})_i], [(\mathbf{r})_i])$, $\mathcal{S}$ handles this the same way it handled calls to $\mathcal{F}_{\text{mult}}$ in Round 2. This includes recording the adversary's additive error $\hat{\varepsilon}_i$.

(d) $\mathcal{S}$ computes: $\hat{\varepsilon} = \sum_{i\in[n]} \hat{\varepsilon}_i$ and $\hat{t} = \langle \mathbf{l},\mathbf{r}\rangle + \hat{\varepsilon}$.

(e) $\mathcal{S}$ opens $([\tau_x],[\mu],[\hat{t}],[\mathbf{l}],[\mathbf{r}])$ to $(\tau_x,\mu,\hat{t},\mathbf{l},\mathbf{r})$ using the **Opening** procedure from the simulation of Round 1.

We now prove that $\mathcal{S}$ correctly simulates the view of the malicious provers in the real execution of the protocol in section 6.1. Firstly, if the witness is rejecting, then $\mathcal{S}$ outputs $\perp$, which the real protocol also does. Therefore, we can assume that the witness is accepting.

Now, the adversary can only introduce additive errors when $\mathcal{F}_{\text{mult}}$ is called, which only occurs in the computation of $([t_2],[t_3],[t_4],[t_5],[t_6])$ in Round 2 and in the computation of $[\hat{t}]$ in Round 3. For example, with an additive error, t_3 gets modified as follows: the adversary specifies several error values $\left(\varepsilon_{i,2,1}\right)_{i\in[n]}$, and we let $\varepsilon_3 = \sum_{i=1}^{n}\varepsilon_{i,2,1}$. Thus,

$$t_3 = \left(\sum_{i=1}^{n}(\mathbf{l}_2)_i\cdot(\mathbf{r}_1)_i + \varepsilon_{i,2,1}\right) + \langle \mathbf{l}_3,\mathbf{r}_0\rangle = \langle \mathbf{l}_2,\mathbf{r}_1\rangle + \langle \mathbf{l}_3,\mathbf{r}_0\rangle + \varepsilon_3$$

ε_3 is simply added to the correct value of t_3: $\langle \mathbf{l}_2,\mathbf{r}_1\rangle + \langle \mathbf{l}_3,\mathbf{r}_0\rangle$. The errors work similarly for $(t_2,t_4,t_5,t_6,\hat{t})$:

$$t_2 = \langle \mathbf{l}_1,\mathbf{r}_1\rangle + \langle \mathbf{l}_2,\mathbf{r}_0\rangle + \varepsilon_2; \quad t_4 = \langle \mathbf{l}_1,\mathbf{r}_3\rangle + \langle \mathbf{l}_3,\mathbf{r}_1\rangle + \varepsilon_4; \quad t_5 = \langle \mathbf{l}_2,\mathbf{r}_3\rangle + \varepsilon_5;$$
$$t_6 = \langle \mathbf{l}_3,\mathbf{r}_3\rangle + \varepsilon_6; \quad \hat{t} = \langle \mathbf{l},\mathbf{r}\rangle + \hat{\varepsilon}$$

which gives: $\langle \ell(X), \mathrm{r}(X)\rangle = t_1\cdot X + \sum_{i=2}^{6}(t_i-\varepsilon_i)\cdot X^i$.
The following claim completes the proof by showing that if the witness is accepting, then variables output by $\mathcal{S}$ are identically distributed as in the real protocol.

Claim. If the witness is accepting, then the variables $(A_I, A_O, S; y, z; T_1, T_3, T_4, T_5, T_6; x; \tau_x, \mu, \hat{t}, \mathbf{l}, \mathbf{r})$ output by $\mathcal{S}$ have the same distribution as they do in the real protocol.

Proof. We prove that the distribution of each of the variables listed is identical in the simulated and real worlds. Firstly, in the real protocol, the following variables are sampled independently and uniformly at random over their respective sample spaces:

$$\alpha, \beta, \rho, x, y, z, \tau_1, \tau_3, \tau_4, \tau_5, \tau_6, \mathbf{l}, \mathbf{r}$$

Due to the randomness of the variables above, the following variables are independent and uniformly random over their respective sample spaces, as long as $x, y \neq 0$:

$$A_I, A_O, \mu, x, y, z, \tau_x, T_3, T_4, T_5, T_6, \mathbf{s}_L, \mathbf{s}_R$$

More specifically, the randomness of (α, β, ρ) makes (A_I, A_O, μ) uniformly random and independent; the randomness of $(\tau_1, \tau_3, \tau_4, \tau_5, \tau_6)$ makes $(\tau_x, T_3, T_4, T_5, T_6)$ uniformly random and independent; the randomness of $(\mathbf{s}_L, \mathbf{s}_R)$ makes $(\mathbf{l}, \mathbf{r})$ uniformly random and independent.

Now consider S from Round 1(e). It can be computed by eq. (1) since it is uniquely determined by the values above – $(\alpha, \beta, \rho, x, y, z, \tau_1, \tau_3, \tau_4, \tau_5, \tau_6, \mathbf{l}, \mathbf{r})$ and $(A_I, A_O, \mu, \tau_x, T_3, T_4, T_5, T_6, \mathbf{s}_L, \mathbf{s}_R)$. This computation of S can be done as shown below, which matches eq. (1):

$$\begin{aligned}
&\left(-\mathbf{y}^n + \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O\right) + x \cdot \left(\mathbf{y}^n \circ \mathbf{a}_R + \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L\right) + x^3 \cdot (\mathbf{y}^n \circ \mathbf{s}_R) = \mathbf{r} \\
&\mathbf{r} + \mathbf{y}^n - \mathbf{z}_{[1:]}^{Q+1} \cdot (x \cdot \mathbf{W}_L + \mathbf{W}_O) = x \cdot \mathbf{y}^n \circ \mathbf{a}_R + x^3 \cdot (\mathbf{y}^n \circ \mathbf{s}_R) \\
&\mathbf{y}^{-n} \circ \left(\mathbf{r} + \mathbf{y}^n - \mathbf{z}_{[1:]}^{Q+1} \cdot (x \cdot \mathbf{W}_L + \mathbf{W}_O)\right) = x \cdot \mathbf{a}_R + x^3 \cdot \mathbf{s}_R
\end{aligned}$$

$$\begin{aligned}
&A_I^x \cdot A_O^{x^2} \cdot S^{x^3} = (h^{x \cdot \alpha} \cdot \mathbf{g}^{x \cdot \mathbf{a}_L} \cdot \mathbf{h}^{x \cdot \mathbf{a}_R}) \cdot \left(h^{x^2 \cdot \beta} \cdot \mathbf{g}^{x^2 \cdot \mathbf{a}_O}\right) \cdot \left(h^{x^3 \cdot \rho} \cdot \mathbf{g}^{x^3 \cdot \mathbf{s}_L} \cdot \mathbf{h}^{x^3 \cdot \mathbf{s}_R}\right) \\
&= \left(h^{\alpha \cdot x + \beta \cdot x^2 + \rho \cdot x^3}\right) \cdot \left(\mathbf{g}^{x \cdot \mathbf{a}_L + x^2 \cdot \mathbf{a}_O + x^3 \cdot \mathbf{s}_L}\right) \cdot \left(\mathbf{h}^{x \cdot \mathbf{a}_R + x^3 \cdot \mathbf{s}_R}\right) \\
&= h^{\mu} \cdot \mathbf{g}^{\mathbf{l} - x \cdot \mathbf{y}^{-n} \circ \left(\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R\right)} \cdot \mathbf{h}^{\mathbf{y}^{-n} \circ \left(\mathbf{r} + \mathbf{y}^n - \mathbf{z}_{[1:]}^{Q+1} \cdot (x \cdot \mathbf{W}_L + \mathbf{W}_O)\right)} \\
&S = \left(A_I^{-x} \cdot A_O^{-x^2} \cdot h^{\mu} \cdot \mathbf{g}^{\mathbf{l} - x \cdot \mathbf{y}^{-n} \circ \left(\mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R\right)} \cdot \mathbf{h}^{\mathbf{y}^{-n} \circ \left(\mathbf{r} + \mathbf{y}^n - \mathbf{z}_{[1:]}^{Q+1} \cdot (x \cdot \mathbf{W}_L + \mathbf{W}_O)\right)}\right)^{(x^{-3})}
\end{aligned}$$

Next, consider T_1 from eq. (2). We show below that it is uniquely determined by the values above, and hence can be computed correctly.
If the witness is valid, then $(\mathbf{V})_i = g^{(\mathbf{v})_i} \cdot h^{(\gamma)_i}, \quad \forall i \in [m]$, $\mathbf{a}_L \circ \mathbf{a}_R = \mathbf{a}_O$ and $\mathbf{W}_V \cdot \mathbf{v} + \mathbf{c} = \mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O$. Therefore, we have

$$\begin{aligned}
\langle \ell(X), \mathrm{r}(X) \rangle &= t_1 \cdot X + \sum_{i=2}^{6} (t_i - \varepsilon_i) \cdot X^i \\
\langle \mathbf{l}, \mathbf{r} \rangle = \langle \ell(x), \mathrm{r}(x) \rangle &= t_1 \cdot x + \sum_{i=2}^{6} (t_i - \varepsilon_i) \cdot x^i
\end{aligned}$$

This implies that:

$$\begin{aligned}
&t_2 - \varepsilon_2 = \langle \mathbf{l}_1, \mathbf{r}_1 \rangle + \langle \mathbf{l}_2, \mathbf{r}_0 \rangle \\
&= \left(\langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle - \langle \mathbf{a}_O, \mathbf{y}^n \rangle\right) + \left(\langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_R, \mathbf{a}_R \rangle\right. \\
&\left. + \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_L, \mathbf{a}_L \rangle + \langle \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_O, \mathbf{a}_O \rangle\right) + \delta(y, z) \\
&= \langle (\mathbf{a}_L \circ \mathbf{a}_R - \mathbf{a}_O), \mathbf{y}^n \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, (\mathbf{W}_L \cdot \mathbf{a}_L + \mathbf{W}_R \cdot \mathbf{a}_R + \mathbf{W}_O \cdot \mathbf{a}_O) \rangle + \delta(y, z) \\
&= \langle \mathbf{z}_{[1:]}^{Q+1}, (\mathbf{W}_V \cdot \mathbf{v} + \mathbf{c}) \rangle + \delta(y, z) = \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle + \delta(y, z)
\end{aligned}$$

which further implies that:

$$
\begin{aligned}
T_1^x \cdot \prod_{i=3}^{6} (T_i \cdot g^{-\varepsilon_i})^{(x^i)} &= g^{t_1 \cdot x} \cdot h^{\tau_1 \cdot x} \cdot \prod_{i=3}^{6} \left(g^{(t_i - \varepsilon_i) \cdot x^i} \cdot h^{\tau_i \cdot x^i} \right) \\
&= g^{t_1 \cdot x + \sum_{i=3}^{6} (t_i - \varepsilon_i) \cdot x^i} \cdot h^{\tau_1 \cdot x + \sum_{i=3}^{6} \tau_i \cdot x^i} \\
&= g^{\langle \mathbf{l}, \mathbf{r} \rangle - (t_2 - \varepsilon_2) \cdot x^2} \cdot h^{\tau_x - x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \boldsymbol{\gamma} \rangle} \\
&= g^{\langle \mathbf{l}, \mathbf{r} \rangle - x^2 \cdot \left(\langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} \rangle + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle + \delta(y,z) \right)} \cdot h^{\tau_x - x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \boldsymbol{\gamma} \rangle} \\
&= h^{\tau_x} \cdot g^{\langle \mathbf{l}, \mathbf{r} \rangle - x^2 \cdot \left(\langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle + \delta(y,z) \right)} \cdot \left(g^{-x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \mathbf{v} \rangle} \cdot h^{-x^2 \cdot \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{W}_V \cdot \boldsymbol{\gamma} \rangle} \right) \\
&= h^{\tau_x} \cdot g^{\langle \mathbf{l}, \mathbf{r} \rangle - x^2 \cdot \left(\langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle + \delta(y,z) \right)} \cdot \left(\mathbf{V}^{-x^2 \cdot \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V} \right) \\
\therefore T_1 &= \left(h^{\tau_x} \cdot g^{\langle \mathbf{l}, \mathbf{r} \rangle - x^2 \cdot \left(\delta(y,z) + \langle \mathbf{z}_{[1:]}^{Q+1}, \mathbf{c} \rangle \right)} \cdot \mathbf{V}^{-x^2 \cdot \mathbf{z}_{[1:]}^{Q+1} \cdot \mathbf{W}_V} \cdot \prod_{i=3}^{6} (T_i \cdot g^{-\varepsilon_i})^{-(x^i)} \right)^{(x^{-1})}
\end{aligned}
$$

Finally, note that in the real protocol and the simulated protocol, $\hat{t} = \langle \mathbf{l}, \mathbf{r} \rangle + \hat{\varepsilon}$. □

□

7 Collaborative zk-SNARK Based On Plonk

In this section, we describe and prove that the semi-honest protocol for collaborative proof generation based on Plonk [GWC19] is, with a minor change, secure against malicious provers in the honest majority setting. We begin by describing some notations used in this section and describe the collaborative zk-SNARK protocol for Plonk in section 7.1 using standard MPC functionalities from section 3.1. We prove the malicious security of this protocol in section 7.2.

Notation. For the underlying Plonk zk-SNARK protocol, we refer the reader to [GWC19, Section 8]. We use the same notations in our collaborative zk-SNARK protocol, which are described as follows. Common variables used: $n \in \mathbb{N}$, the number of gates in the arithmetic circuit; $\ell \in \mathbb{N}$, the number of public input wires to the circuit; $(w_i)_{i \in [3n]}$ are the wires of the arithmetic circuit; $(w_i)_{i \in [\ell]}$ are the public input wires; $\left(q_{M_i}, q_{L_i}, q_{R_i}, q_{O_i}, q_{C_i}\right)_{i \in [n]} \in (\mathbb{F}^5)^n$ define the arithmetic circuit; $\mathsf{q}_M(X), \mathsf{q}_L(X), \mathsf{q}_R(X), \mathsf{q}_O(X), \mathsf{q}_C(X) \in \mathbb{F}[X]$ are selector polynomials that encode $\left(q_{M_i}, q_{L_i}, q_{R_i}, q_{O_i}, q_{C_i}\right)_{i \in [n]}$; $\omega \in \mathbb{F}$ is an nth root of unity; $H = \{1, \omega, \ldots, \omega^{n-1}\}$; $k_1, k_2 \in \mathbb{F}$ have the property that $H, k_1 \cdot H, k_2 \cdot H$ are distinct cosets of H; $H' = H \cup (k_1 \cdot H) \cup (k_2 \cdot H)$; $\sigma^* : [3n] \to H'$ is a one-to-one function; $\mathsf{S}_{\sigma 1}(X), \mathsf{S}_{\sigma 2}(X), \mathsf{S}_{\sigma 3}(X) \in \mathbb{F}[X]$: encode σ^*; $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}$ is a random oracle; $x \xleftarrow{\$} \mathbb{F}$. The inputs to the Plonk protocol include some common preprocessed inputs, public instance $\mathcal{X}$ and witness wtn:

Common Preprocessed Inputs:

- $n, \left([x]_1, \ldots, [x^{n+5}]_1\right), \left(q_{M_i}, q_{L_i}, q_{R_i}, q_{O_i}, q_{C_i}\right)_{i \in [n]}, \sigma^*$
- $\mathsf{q}_M(X), \mathsf{q}_L(X), \mathsf{q}_R(X), \mathsf{q}_O(X), \mathsf{q}_C(X)$
- $\mathsf{S}_{\sigma 1}(X), \mathsf{S}_{\sigma 2}(X), \mathsf{S}_{\sigma 3}(X)$

Instance $\mathcal{X}$ (held by the provers and $\mathcal{V}$): $\ell, (w_i)_{i \in [\ell]}$
Witness wtn (distributed among the provers): $\mathsf{wtn} = (w_i)_{i \in [3n]}$
Relation $\mathfrak{R}_{\mathsf{Plonk}}$: $\{(\mathcal{X}, \mathsf{wtn}) : \forall i \in [n] \ (q_{M_i} w_i w_{n+i} + q_{L_i} w_i + q_{R_i} w_{n+i} + q_{O_i} w_{2n+i} + q_{C_i} = 0) \wedge (\forall i \in$

$[3n]\ w_i = w_{\sigma^*(i)})\}$

Collaborative Extended Witness Generation. Recall that in collaborative zk-SNARKs, each prover holds a witness fragment wtn_i, which needs to be compiled into a valid witness wtn. We formally define the following functionality for witness extension below. This functionality can be realized using maliciously secure MPC techniques, the details of which are beyond the scope of this work.

Functinality $\mathcal{F}_{\text{Plonk-WE}}$: Plonk Witness Extension

1: *Inputs:* $\mathcal{F}_{\text{Plonk-WE}}$ receives the instance $\mathcal{X}$ and each party's witness fragment $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$. It also receives from the adversary the corrupted parties' shares of the wire values: $(w_1^j, \ldots, w_{3n}^j)_{\forall j \in C}$.

2: The functionality verifies the witness by checking that $V(\mathcal{X}, (\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)) = 1$. If verification fails, the functionality sends $\perp$ to all parties and halts. Otherwise, the functionality continues.

3: The functionality uses $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$ to compute the corresponding wire values $(w_1, \ldots, w_{3n})$ for the Plonk arithmetic circuit.

4: The functionality computes the shares of the wire values for each party as follows. For each $i \in \{1, \ldots, 3n\}$, the functionality computes:

$$(w_i^1, \ldots, w_i^N) = \mathsf{share}(w_i, (w_i^j)_{\forall j \in C})$$

5: The functionality sends each party $j \in \{1, \ldots, N\}$ their shares of the wire values:

$$(w_1^j, \ldots, w_{3n}^j)$$

7.1 The Collaborative Plonk Protocol

In this section, we describe the collaborative zk-SNARK protocol for Plonk. The protocol begins by running the witness extension by invoking $\mathcal{F}_{\text{Plonk-WE}}$, and then generates the Plonk proof from [GWC19, Section 8] by running a combination of MPC functionalities: $\mathcal{F}_{\text{input}}$, $\mathcal{F}_{\text{rand}}$, $\mathcal{F}_{\text{coin}}$, $\mathcal{F}_{\text{checkZero}}$, $\mathcal{F}_{\text{mult}}$, and $\mathcal{F}_{\text{polyMult}}$. We describe the five rounds of the protocol in detail below. There is only a small modification to the semi-honest collaborative Plonk protocol to adapt to the honest-majority malicious security setting in Round 5 (highlighted in green).

Round 1 : The Plonk prover computes random blinding scalars and computes the wire polynomials $\mathsf{a}(X)$, $\mathsf{b}(X)$ and $\mathsf{c}(X)$. In collaborative Plonk, all these computations are done locally on the shares of these vectors by the provers.

Round 1: The provers do the following:

1: **Witness Extension:** Send the inputs $(\mathsf{wtn}_1, \ldots, \mathsf{wtn}_N)$ to $\mathcal{F}_{\text{Plonk-WE}}$. If $\mathcal{F}_{\text{Plonk-WE}}$ outputs $\perp$, then abort the protocol. Otherwise, $\mathcal{F}_{\text{Plonk-WE}}$ outputs: $([w_1], \ldots, [w_{3n}])$

2: Call $\mathcal{F}_{\text{rand}}$ 9 times to generate blinding scalars in secret-shared form: $([b_1], \ldots, [b_9])$.

3: Compute the wire polynomials:

$$[\mathrm{a}(X)] = ([b_1]X + [b_2]) \cdot Z_H(X) + \sum_{i=1}^{n} [w_i] \cdot L_i(X)$$
$$[\mathrm{b}(X)] = ([b_3]X + [b_4]) \cdot Z_H(X) + \sum_{i=1}^{n} [w_{n+i}] \cdot L_i(X)$$
$$[\mathrm{c}(X)] = ([b_5]X + [b_6]) \cdot Z_H(X) + \sum_{i=1}^{n} [w_{2n+i}] \cdot L_i(X)$$

4: Evaluate each polynomial at x in the exponent to compute $[[\mathrm{a}(x)]_1]$, $[[\mathrm{b}(x)]_1]$, $[[\mathrm{c}(x)]_1]$.
5: Compute: $[\mathrm{a}(x)]_1 = \mathrm{open}([[\mathrm{a}(x)]_1])$, $[\mathrm{b}(x)]_1 = \mathrm{open}([[\mathrm{b}(x)]_1])$ and $[\mathrm{c}(x)]_1 = \mathrm{open}([[\mathrm{c}(x)]_1])$ Here, parties publish their shares and reconstruct.
6: Output $[\mathrm{a}(x)]_1, [\mathrm{b}(x)]_1, [\mathrm{c}(x)]_1$.

Round 2 : The Plonk prover computes the permutation challenges by running the random oracle RO and then computes the permutation polynomial $\mathrm{z}(X)$. This involves computing $n-1$ partial products, with each term of the partial product involving three multiplications divided by three multiplications. This polynomial captures the permutation check for the wires. In the collaborative Plonk protocol, we use the Bar-Ilan and Beaver technique [BIB89] (implicit in the description below) to mask the terms of the partial product in a way that the partial product computation can be done in the clear on the masked values, with just 4 rounds of communication between the provers.

Round 2: The provers do the following:

1: Let transcript be a tuple of the common preprocessed inputs, the public inputs, and the outputs of round 1, $([\mathrm{a}(x)]_1, [\mathrm{b}(x)]_1, [\mathrm{c}(x)]_1)$. Then compute $\beta = \mathrm{RO}(\mathrm{transcript}, 0)$ and $\gamma = \mathrm{RO}(\mathrm{transcript}, 1)$.
2: **Factors of the partial products:** For each $i \in [n-1]$, compute the following factors:

$$[e_{i,1}] = [w_i] + \beta\omega^i + \gamma; \qquad [e_{i,2}] = [w_{n+i}] + \beta k_1 \omega^i + \gamma$$
$$[e_{i,3}] = [w_{2n+i}] + \beta k_2 \omega^i + \gamma; \qquad [e_{i,4}] = [w_i] + \sigma^*(i) \cdot \beta + \gamma$$
$$[e_{i,5}] = [w_{n+i}] + \sigma^*(n+i) \cdot \beta + \gamma; \qquad [e_{i,6}] = [w_{2n+i}] + \sigma^*(2n+i) \cdot \beta + \gamma$$

3: **Sample random masks:** For each $k \in \{0, 1, \ldots, 6 \cdot (n-1)\}$, call $\mathcal{F}_{\mathrm{rand}}$ twice to generate $[r_k]$ and $[s_k]$. For each k such that $k = 0$ or $k \neq 0 \mod 3$, compute:

$$[t_k] = \mathcal{F}_{\mathrm{mult}}([r_k], [s_k])$$

and for each $i \in [n-1]$, compute:

$$[u_i] = \mathcal{F}_{\mathrm{mult}}([s_0], [r_{6 \cdot i}])$$

4: **Mask the factors:** $\forall (i, j) \in [n-1] \times [3]$, $k = 6 \cdot (i-1) + j$, compute:

$$[f_{i,j}] = \mathcal{F}_{\mathrm{mult}}\left(\mathcal{F}_{\mathrm{mult}}\left([r_{k-1}], [e_{i,j}]\right), [s_k]\right)$$

and $\forall (i, j) \in [n-1] \times \{4, 5, 6\}$, $k = 6 \cdot (i-1) + j$, compute:

$$[f_{i,j}] = \mathcal{F}_{\mathrm{mult}}\left(\mathcal{F}_{\mathrm{mult}}\left([s_{k-1}], [e_{i,j}]\right), [r_k]\right)$$

5: **Open the t_ks and $f_{i,j}$s:** $\forall k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$, compute: $t_k = \text{open}([t_k])$ and $\forall (i,j) \in [n-1] \times [6]$, compute: $f_{i,j} = \text{open}\left([f_{i,j}]\right)$.

6: **Combine the factors:** Let $h_0 = 1$ and $\mathsf{PP}_0 = 1$. Then for each $i \in [n-1]$ and $k = 6 \cdot (i-1)$, compute the following:

$$g_i = \frac{f_{i,1} \cdot f_{i,2} \cdot f_{i,3}}{f_{i,4} \cdot f_{i,5} \cdot f_{i,6}} \cdot \frac{t_{k+4} \cdot t_{k+5}}{t_{k+1} \cdot t_{k+2}}; \quad h_i = h_{i-1} \cdot g_i; \quad [\mathsf{PP}_i] = \frac{h_i}{t_0} \cdot [u_i]$$

7: **Compute the permutation polynomial** $[\mathsf{z}(X)]$:

$$[\mathsf{z}(X)] = \left([b_7]X^2 + [b_8]X + [b_9]\right) \cdot \mathsf{Z}_H(X) + \sum_{i=1}^{n} [\mathsf{PP}_{i-1}] \cdot \mathsf{L}_i(X)$$

8: Evaluate $[\mathsf{z}(X)]$ at x in the exponent to compute $[[\mathsf{z}(x)]_1]$.

9: Compute $\text{open}([[\mathsf{z}(x)]_1])$ and output $[\mathsf{z}(x)]_1$.

Round 3 : The Plonk prover computes the random oracle output on the transcript to get the quotient challenge. It then computes a quotient polynomial $\mathsf{t}(X)$ that essentially combines all the checks (gate and wire) needed into a single polynomial and computes the quotient on division by the vanishing polynomial. For optimization, the Plonk prover also splits this huge polynomial into three polynomials of degree at most $n + 5$ each. The collaborative Plonk provers compute this by invoking the $\mathcal{F}_{\text{polyMult}}$ functionality, along with local computations.

Round 3: The provers do the following:

1: Append $[\mathsf{z}(x)]_1$, the output of round 2, to the end of transcript and compute $\alpha = \mathsf{RO}(\text{transcript})$.

2: Compute the following polynomials:

$$\begin{aligned}
&[\mathsf{a}'(X)] := [\mathsf{a}(X)] + \beta X + \gamma; && [\mathsf{b}'(X)] := [\mathsf{b}(X)] + \beta k_1 X + \gamma;\\
&[\mathsf{c}'(X)] := [\mathsf{c}(X)] + \beta k_2 X + \gamma; && [\mathsf{a}''(X)] := [\mathsf{a}(X)] + \beta S_{\sigma 1}(X) + \gamma;\\
&[\mathsf{b}''(X)] := [\mathsf{b}(X)] + \beta S_{\sigma 2}(X) + \gamma; && [\mathsf{c}''(X)] := [\mathsf{c}(X)] + \beta S_{\sigma 3}(X) + \gamma
\end{aligned}$$

3: Compute $[\mathsf{t}(X)]$ as follows:

$$\begin{aligned}
&[\mathsf{line}_1(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{a}(X)], [\mathsf{b}(X)]\right) \cdot \mathsf{q}_M(X)\\
&+ [\mathsf{a}(X)] \cdot q_L(X) + [\mathsf{b}(X)] \cdot q_R(X) + [\mathsf{c}(X)] \cdot q_O(X) + PI(X) + q_C(X)\\
&[\mathsf{line}_2(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{a}'(X)], [\mathsf{b}'(X)]\right)\\
&[\mathsf{line}_2'(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{line}_2(X)], [\mathsf{c}'(X)]\right)\\
&[\mathsf{line}_2''(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{line}_2'(X)], [\mathsf{z}(X)]\right) \cdot \alpha\\
&[\mathsf{line}_3(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{a}''(X)], [\mathsf{b}''(X)]\right)\\
&[\mathsf{line}_3'(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{line}_3(X)], [\mathsf{c}''(X)]\right)\\
&[\mathsf{line}_3''(X)] = \mathcal{F}_{\text{polyMult}}\left([\mathsf{line}_3'(X)], [\mathsf{z}(X\omega)]\right) \cdot (-\alpha)\\
&[\mathsf{line}_4(X)] = ([\mathsf{z}(X)] - 1) \cdot L_1(X) \cdot \alpha^2
\end{aligned}$$

$$[\mathsf{t}(X)] = \mathsf{Qt}\left(\frac{[\mathsf{line}_1(X)] + [\mathsf{line}_2''(X)] + [\mathsf{line}_3''(X)] + [\mathsf{line}_4(X)]}{\mathsf{Z}_H(X)}\right)$$

4: Split $[\mathsf{t}(X)]$ into three polynomials $[\mathsf{t}'_{\mathsf{lo}}(X)], [\mathsf{t}'_{\mathsf{mid}}(X)], [\mathsf{t}'_{\mathsf{hi}}(X)]$ such that

$$[\mathsf{t}(X)] = [\mathsf{t}'_{\mathsf{lo}}(X)] + X^n \cdot [\mathsf{t}'_{\mathsf{mid}}(X)] + X^{2n} \cdot [\mathsf{t}'_{\mathsf{hi}}(X)]$$

and $\deg\left(\mathsf{t}'_{\mathsf{lo}}(X)\right) \leq n-1, \deg\left(\mathsf{t}'_{\mathsf{mid}}(X)\right) \leq n-1, \deg\left(\mathsf{t}'_{\mathsf{hi}}(X)\right) \leq n+5$.

5: Call $\mathcal{F}_{\mathsf{rand}}$ twice to obtain secret sharings of two blinding scalars, $[b_{10}], [b_{11}]$.

6: Compute the following polynomials:

$$[\mathsf{t}_{\mathsf{lo}}(X)] = [\mathsf{t}'_{\mathsf{lo}}(X)] + [b_{10}] \cdot X^n; \quad [\mathsf{t}_{\mathsf{mid}}(X)] = [\mathsf{t}'_{\mathsf{mid}}(X)] - [b_{10}] + [b_{11}] \cdot X^n$$
$$[\mathsf{t}_{\mathsf{hi}}(X)] = [\mathsf{t}'_{\mathsf{hi}}(X)] - [b_{11}]$$

7: Evaluate $[\mathsf{t}_{\mathsf{lo}}(X)], [\mathsf{t}_{\mathsf{mid}}(X)], [\mathsf{t}_{\mathsf{hi}}(X)]$ at x in the exponent to compute:

$$[[\mathsf{t}_{\mathsf{lo}}(x)]_1], [[\mathsf{t}_{\mathsf{mid}}(x)]_1], [[\mathsf{t}_{\mathsf{hi}}(x)]_1]$$

8: Compute and output the openings: $[\mathsf{t}_{\mathsf{lo}}(x)]_1 = \mathsf{open}([[\mathsf{t}_{\mathsf{lo}}(x)]_1])$, $[\mathsf{t}_{\mathsf{mid}}(x)]_1 = \mathsf{open}([[\mathsf{t}_{\mathsf{mid}}(x)]_1])$ and $[\mathsf{t}_{\mathsf{hi}}(x)]_1 = \mathsf{open}([[\mathsf{t}_{\mathsf{hi}}(x)]_1])$.

Round 4 : The Plonk prover computes the evaluation challenge using RO and then computes the opening evaluations. The collaborative Plonk provers only need to perform local computations for this.

Round 4: The parties do the following:

1: Concatenate to the end of transcript the output of round 3, $([\mathsf{t}_{\mathsf{lo}}(x)]_1, [\mathsf{t}_{\mathsf{mid}}(x)]_1, [\mathsf{t}_{\mathsf{hi}}(x)]_1)$ and compute $\mathfrak{z} = \mathsf{RO}(\mathsf{transcript})$.

2: Evaluate the following polynomials at $\mathfrak{z}$:

$$[\overline{a}] := [\mathsf{a}(\mathfrak{z})], \quad [\overline{b}] := [\mathsf{b}(\mathfrak{z})], \quad [\overline{c}] := [\mathsf{c}(\mathfrak{z})],$$
$$\overline{\mathsf{s}}_{\sigma 1} := \mathsf{S}_{\sigma 1}(\mathfrak{z}), \quad \overline{\mathsf{s}}_{\sigma 2} := \mathsf{S}_{\sigma 2}(\mathfrak{z}), \quad [\overline{z}_\omega] := [\mathsf{z}(\mathfrak{z}\omega)]$$

3: Call open on each secret-shared value to obtain $(\overline{a}, \overline{b}, \overline{c}, \overline{z}_\omega)$.

4: Output $(\overline{a}, \overline{b}, \overline{c}, \overline{\mathsf{s}}_{\sigma 1}, \overline{\mathsf{s}}_{\sigma 2}, \overline{z}_\omega)$.

Round 5 : The Plonk prover computes the opening challenge using RO, the linearization polynomial $\mathsf{r}(X)$, and the opening proof polynomials $\mathsf{W}_{\mathfrak{z}}(X)$ and $\mathsf{W}_{\mathfrak{z}\omega}(X)$. Finally, it evaluates these polynomials in the exponent at x and sends them to the verifier. The collaborative Plonk provers can compute all these values locally. The only addition we make to the protocol is an additional check if the numerator of the opening proof polynomial $\mathsf{W}_{\mathfrak{z}}(X)$ is divisible by $(X - \mathfrak{z})$ or not. The reason we do this one check is to catch the additive errors that the malicious provers could have introduced in any of the previous rounds. If they did indeed, then we prove that this remainder will not be a zero polynomial.

Round 5: The parties do the following:

1: Concatenate to the end of transcript the output of round 4, $(\overline{a}, \overline{b}, \overline{c}, \overline{\mathsf{s}}_{\sigma 1}, \overline{\mathsf{s}}_{\sigma 2}, \overline{z}_\omega)$ and compute $v = \mathsf{RO}(\mathsf{transcript})$.

2: Compute the polynomial $[\mathsf{r}(X)]$ as follows:

$$
\begin{aligned}
[\mathsf{r}(X)] &= \overline{a}\overline{b} \cdot q_M(X) + \overline{a} \cdot q_L(X) + \overline{b} \cdot q_R(X) + \overline{c} \cdot q_O(X) + PI(\mathfrak{z}) \\
&+ q_C(X) + (\overline{a} + \beta\mathfrak{z} + \gamma) \cdot (\overline{b} + \beta k_1 \mathfrak{z} + \gamma) \cdot (\overline{c} + \beta k_2 \mathfrak{z} + \gamma) \cdot [\mathsf{z}(X)] \cdot \alpha \\
&- (\overline{a} + \beta \overline{\mathsf{s}}_{\sigma 1} + \gamma) \cdot (\overline{b} + \beta \overline{\mathsf{s}}_{\sigma 2} + \gamma) \cdot (\overline{c} + \beta \mathsf{S}_{\sigma 3}(X) + \gamma) \cdot \overline{z}_\omega \cdot \alpha + ([\mathsf{z}(X)] \\
&- 1) \cdot L_1(\mathfrak{z}) \cdot \alpha^2 - \mathsf{Z}_H(\mathfrak{z}) \cdot \left([\mathsf{t}_{\mathsf{lo}}(X)] + \mathfrak{z}^n [\mathsf{t}_{\mathsf{mid}}(X)] + \mathfrak{z}^{2n} [\mathsf{t}_{\mathsf{hi}}(X)]\right)
\end{aligned}
$$

3: Compute the polynomials $[\mathsf{W}_{\mathfrak{z}}(X)]$ and $[\mathsf{W}_{\mathfrak{z}\omega}(X)]$ as follows:

$$
\begin{aligned}
[\mathsf{num}(X)] &= [\mathsf{r}(X)] + v([\mathsf{a}(X)] - \overline{a}) + v^2([\mathsf{b}(X)] - \overline{b}) + v^3([\mathsf{c}(X)] - \overline{c}) \\
&\quad + v^4([\mathsf{S}_{\sigma 1}(X)] - \overline{\mathsf{s}}_{\sigma 1}) + v^5([\mathsf{S}_{\sigma 2}(X)] - \overline{\mathsf{s}}_{\sigma 2}) \\
[\mathsf{W}_{\mathfrak{z}}(X)] &= \mathsf{Qt}\left(\frac{[\mathsf{num}(X)]}{X - \mathfrak{z}}\right); \quad [\mathsf{W}_{\mathfrak{z}\omega}(X)] = \mathsf{Qt}\left(\frac{[\mathsf{z}(X)] - \overline{z}_\omega}{X - \mathfrak{z}\omega}\right)
\end{aligned}
$$

4: **Verify the following:**

1. Compute $[V] := \mathsf{Rd}\left(\frac{[\mathsf{num}(X)]}{X - \mathfrak{z}}\right)$.
2. Compute $\mathcal{F}_{\mathsf{checkZero}}([V])$. If the check passes, continue. Otherwise, output $\perp$ and abort the computation.

5: Evaluate $[\mathsf{W}_{\mathfrak{z}}(X)]$ and $[\mathsf{W}_{\mathfrak{z}\omega}(X)]$ at x in the exponent to compute $[[\mathsf{W}_{\mathfrak{z}}(x)]_1]$ and $[[\mathsf{W}_{\mathfrak{z}\omega}(x)]_1]$.

6: Compute and output $[\mathsf{W}_{\mathfrak{z}}(x)]_1 = \mathsf{open}\left([[\mathsf{W}_{\mathfrak{z}}(x)]_1]\right)$ and $[\mathsf{W}_{\mathfrak{z}\omega}(x)]_1 = \mathsf{open}\left([[\mathsf{W}_{\mathfrak{z}\omega}(x)]_1]\right)$.

The verification checks are the same as the original Plonk protocol [GWC19, Section 8] Note that the above protocol is non-interactive and just describes the prover computations in five separate rounds along with the RO queries on parts of the transcript. Our communication cost for the honest-majority malicious protocol above is almost identical to the semi-honest setting.

Theorem 7.1. *The protocol described in Section section 7.1 is a collaborative proof generation based on Plonk, i.e., for relation $\mathfrak{R}_{\mathsf{Plonk}}$, against $t < N/2$ malicious provers in the $(\mathcal{F}_{\mathit{Plonk\text{-}WE}}, \mathcal{F}_{\mathit{input}}, \mathcal{F}_{\mathit{rand}}, \mathcal{F}_{\mathit{coin}}, \mathcal{F}_{\mathit{checkZero}}, \mathcal{F}_{\mathit{mult}}, \mathcal{F}_{\mathit{polyMult}})$-hybrid model.*

Proof. The completeness, knowledge soundness, and succinctness properties directly follow from the corresponding properties of Plonk. We will prove the t-zero-knowledge property for the protocol in the $(\mathcal{F}_{\text{Plonk-WE}}, \mathcal{F}_{\text{input}}, \mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{coin}}, \mathcal{F}_{\text{checkZero}}, \mathcal{F}_{\text{mult}}, \mathcal{F}_{\text{polyMult}})$-hybrid model against a malicious adversary that controls at most $t < N/2$ provers in section 7.2 below. □

7.2 t-Zero-Knowledge Against Malicious Provers

In this section, we prove that the protocol in section 7.1 satisfies t-zero-knowledge against $t < N/2$ malicious provers. We show that even though this protocol is based on semi-honest techniques[8] that allow a malicious adversary to introduce additive errors, the additive errors turn out to be simulatable.

The Simulator. The first job of the simulator is to keep track of the shares the adversary would hold if they followed the protocol honestly. This allows the simulator to simulate the messages sent to the adversary during the protocol and to decide whether to abort the open procedure if the adversary produces incorrect shares. The second job of the simulator is to sample the output of each round from the correct distribution. We show that for rounds 1,2,3 and 4, the simulator can pick the outputs uniformly (and independently) at random.

[8] except for the one verification check that we add in Round 5.

Simulator $\mathcal{S}$ for Plonk

1. **Round 1:**

(a) **Inputs:** $\mathcal{S}$ receives from $\mathcal{A}$ the corrupted parties inputs $(\mathsf{wtn}_j)_{\forall j \in C}$ as well as the corrupted parties' shares of every wire value: $(w_1^j, \ldots, w_{3n}^j)_{\forall j \in C}$. $\mathcal{S}$ also receives from the t-zero-knowledge challenger a bit $b \leftarrow \mathfrak{R}(X, \mathsf{wtn})$ indicating whether the witness is valid. $\mathcal{S}$ stores all the inputs for later.

(b) **Witness Extension:** If $b = 0$ (meaning the witness is invalid), then $\mathcal{S}$ sends $\perp$ to all parties and halts. Otherwise, $\mathcal{S}$ continues. Then $\mathcal{S}$ sends to $\mathcal{A}$ the values $(w_1^j, \ldots, w_{3n}^j)_{\forall j \in C}$.

(c) **Calls to $\mathcal{F}_{\mathbf{rand}}$:** Whenever Round 1 calls $\mathcal{F}_{\text{rand}}$, $\mathcal{S}$ does the following: $\mathcal{S}$ receives from $\mathcal{A}$ the values $(v_j)_{\forall j \in C}$, which represent the corrupted parties' shares of the random value, and $\mathcal{S}$ stores these shares.

(d) **Local Computations:** The protocol for computing

$$([\mathsf{a}(X)], [\mathsf{b}(X)], [\mathsf{c}(X)], [[\mathsf{a}(x)]_1], [[\mathsf{b}(x)]_1], [[\mathsf{c}(x)]_1])$$

in Round 1 does not require communication between parties (local computations). $\mathcal{S}$ computes the shares of $([\mathsf{a}(X)], [\mathsf{b}(X)], [\mathsf{c}(X)], [[\mathsf{a}(x)]_1], [[\mathsf{b}(x)]_1], [[\mathsf{c}(x)]_1])$ that the corrupted parties would hold if they followed the protocol honestly using the corrupted parties' shares of $(b_1, \ldots, b_6, w_1, \ldots, w_{3n})$.

(e) **Output:** $\mathcal{S}$ samples $(A, B, C) \xleftarrow{\$} \mathbb{G}_1^3$.

(f) **Opening:** $\mathcal{S}$ opens $[[\mathsf{a}(x)]_1]$ to A using the following procedure.

i. $\mathcal{S}$ has already computed the shares of $[[\mathsf{a}(x)]_1]$ that the corrupted parties would hold if they followed the protocol honestly. Let us call those shares $(\alpha_j)_{\forall j \in C}$.

ii. $\mathcal{S}$ computes:

$$(\alpha_j)_{\forall j \in [N]} = \mathsf{share}(A, (\alpha_j)_{\forall j \in C})$$

iii. $\mathcal{S}$ sends $(\alpha_j)_{j \in \mathcal{H}}$ to $\mathcal{A}$, and $\mathcal{A}$ sends the corrupted parties' shares to $\mathcal{S}$. If any of the corrupted parties' shares do not match $(\alpha_j)_{j \in C}$, then $\mathcal{S}$ outputs abort and aborts the protocol.

(g) $\mathcal{S}$ also opens $[[\mathsf{b}(x)]_1]$ to B and $[[\mathsf{c}(x)]_1]$ to C using the same procedure as in the **Opening** step for $[[\mathsf{a}(x)]_1]$.

2. **Round 2:**

(a) $\mathcal{S}$ handles local computations and calls to $\mathcal{F}_{\text{rand}}$ the same way it did for round 1. The local computations include computing transcript, calling RO to compute β and γ, and computing the g_i and h_i values.

(b) **Calls to $\mathcal{F}_{\mathbf{mult}}$:** Whenever Round 2 calls $\mathcal{F}_{\text{mult}}$, $\mathcal{S}$ does the following:
To multiply two values (x, y), $\mathcal{S}$ sends to $\mathcal{A}$ the corrupted parties' shares $[x]_C$ and $[y]_C$, which the simulator has computed from prior steps. Next, $\mathcal{A}$ sends to $\mathcal{S}$ the additive error ε as well as the corrupted parties' shares of the output $[z]_C$. $\mathcal{S}$ stores these values.

(c) **Open the t_ks and $f_{i,j}$s:** For each $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$, $\mathcal{S}$ samples $t_k \xleftarrow{\$} \mathbb{F}$ uniformly at random and then opens $[t_k]$ to t_k using the **Opening**

procedure from the simulation of round 1. For each $(i, j) \in [n-1] \times [6]$, $\mathcal{S}$ samples $f_{i,j} \xleftarrow{\$} \mathbb{F}$ uniformly at random and then opens $[f_{i,j}]$ to $f_{i,j}$ using the **Opening** procedure from the simulation of round 1.

(d) **Output:** $\mathcal{S}$ samples $Z \xleftarrow{\$} \mathbb{G}_1$ and then opens $[\mathsf{z}(x)]_1$ to Z using the **Opening** procedure from the simulation of round 1.

3. **Round 3:**

(a) $\mathcal{S}$ handles local computations and calls to $\mathcal{F}_{\text{rand}}$ the same way it did for the previous rounds.

(b) **Polynomial Multiplication:** Whenever round 3 calls $\mathcal{F}_{\text{polyMult}}$, $\mathcal{S}$ does the following:
To multiply two polynomials $(\mathsf{p}(X), \mathsf{q}(X))$, $\mathcal{S}$ sends to $\mathcal{A}$ the corrupted parties' shares $[\mathsf{p}(X)]_C$ and $[\mathsf{q}(X)]_C$, which the simulator has computed from prior steps. Next, $\mathcal{A}$ sends to $\mathcal{S}$ the additive error $\varepsilon(X)$ as well as the corrupted parties' shares of the output $[\mathsf{r}(X)]_C$. $\mathcal{S}$ stores these values.
Let $(\varepsilon_1(X), \ldots, \varepsilon_7(X))$ be the error polynomials introduced in round 3 each time PolyMult is called.

(c) **Output:** $\mathcal{S}$ samples $(T_{\mathsf{lo}}, T_{\mathsf{mid}}, T_{\mathsf{hi}}) \xleftarrow{\$} \mathbb{G}_1^3$.

(d) $\mathcal{S}$ opens $([\mathsf{t}_{\mathsf{lo}}(x)]_1, [\mathsf{t}_{\mathsf{mid}}(x)]_1, [\mathsf{t}_{\mathsf{hi}}(x)]_1)$ to the values $(T_{\mathsf{lo}}, T_{\mathsf{mid}}, T_{\mathsf{hi}})$ using the **Opening** procedure from the simulation of round 1.

4. **Round 4:**

(a) $\mathcal{S}$ handles local computations the same way it did for the previous rounds.

(b) $\mathcal{S}$ computes $\bar{\mathsf{s}}_{\sigma 1} = \mathsf{S}_{\sigma 1}(\mathfrak{z})$ and $\bar{\mathsf{s}}_{\sigma 2} = \mathsf{S}_{\sigma 2}(\mathfrak{z})$ honestly.

(c) **Output:** $\mathcal{S}$ samples $\left(\overline{A}, \overline{B}, \overline{C}, \overline{Z}_\omega\right) \xleftarrow{\$} \mathbb{F}^4$.

(d) $\mathcal{S}$ opens $\left([\overline{a}], [\overline{b}], [\overline{c}], [\bar{\mathsf{s}}_{\sigma 1}], [\bar{\mathsf{s}}_{\sigma 2}], [\overline{z}_\omega]\right)$ to the values $\left(\overline{A}, \overline{B}, \overline{C}, \bar{\mathsf{s}}_{\sigma 1}, \bar{\mathsf{s}}_{\sigma 2}, \overline{Z}_\omega\right)$ the **Opening** procedure from the simulation of round 1.

5. **Round 5:**

(a) $\mathcal{S}$ handles local computations the same way as before.

(b) **Verification:**

i. $\mathcal{S}$ checks the following conditions:
Conditions:
A. In round 2, every call to $\mathcal{F}_{\text{mult}}$ has an additive error of $\varepsilon = 0$.
B. $0 = \varepsilon_2(X) = \varepsilon_3(X) = \varepsilon_5(X) = \varepsilon_6(X)$.
C. $\mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right) = 0$
If A, B and C hold, then $\mathcal{S}$ sets $V = 0$. Otherwise, $\mathcal{S}$ sets $V \neq 0$.

(c) $\mathcal{S}$ simulates $\mathcal{F}_{\text{checkZero}}([V])$ as follows. If $V = 0$, then $\mathcal{S}$ sends 0 to $\mathcal{A}$. $\mathcal{A}$ responds with accept or reject, and $\mathcal{S}$ forwards $\mathcal{A}$'s response to the honest parties.
If $V \neq 0$, then:

i. With probability $\frac{1}{|\mathbb{F}|}$, $\mathcal{S}$ sends accept to all parties.

ii. With probability $1 - \frac{1}{|\mathbb{F}|}$, $\mathcal{S}$ sends reject to all parties.

If $\mathcal{F}_{\text{checkZero}}([V])$ rejects, then $\mathcal{S}$ outputs $\perp$ and aborts the protocol. Otherwise, $\mathcal{S}$ continues.

(d) **Output:** $\mathcal{S}$ computes:

$$
\begin{aligned}
&[\mathsf{r}(x)]_1 = \overline{a}\overline{b} \cdot [\mathsf{q}_M(x)]_1 + \overline{a} \cdot [\mathsf{q}_L(x)]_1 + \overline{b} \cdot [\mathsf{q}_R(x)]_1 + \overline{c} \cdot [\mathsf{q}_O(x)]_1 \\
&+ PI(\mathfrak{z}) + [\mathsf{q}_C(x)]_1 + (\overline{a} + \beta\mathfrak{z} + \gamma) \cdot (\overline{b} + \beta k_1 \mathfrak{z} + \gamma) \cdot (\overline{c} + \beta k_2 \mathfrak{z} + \gamma) \\
&\cdot Z \cdot \alpha - (\overline{a} + \beta \overline{\mathsf{s}}_{\sigma 1} + \gamma) \cdot (\overline{b} + \beta \overline{\mathsf{s}}_{\sigma 2} + \gamma) \cdot (\overline{c} + \beta [\mathsf{S}_{\sigma 3}(x)]_1 + \gamma) \\
&\cdot \overline{z}_\omega \cdot \alpha + (Z - 1) \cdot L_1(\mathfrak{z}) \cdot \alpha^2 - \mathsf{Z}_H(\mathfrak{z}) \cdot \left(T_{\text{lo}} + \mathfrak{z}^n T_{\text{mid}} + \mathfrak{z}^{2n} T_{\text{hi}}\right)
\end{aligned}
$$

$$
\begin{aligned}
&[\mathsf{num}(x)]_1 = [\mathsf{r}(x)]_1 + v(A - \overline{a}) + v^2(B - \overline{b}) + v^3(C - \overline{c}) \\
&+ v^4([\mathsf{S}_{\sigma 1}(x)]_1 - \overline{\mathsf{s}}_{\sigma 1}) + v^5([\mathsf{S}_{\sigma 2}(x)]_1 - \overline{\mathsf{s}}_{\sigma 2})
\end{aligned}
$$

$$
W_{\mathfrak{z}} := \frac{[\mathsf{num}(x)]_1}{x - \mathfrak{z}}; \quad W_{\mathfrak{z}\omega} := \frac{Z - [\overline{z}_\omega]_1}{x - \mathfrak{z}\omega}
$$

For this step, $\mathcal{S}$ will use the fact that they know x in the clear.

(e) $\mathcal{S}$ opens $\left([[\mathsf{W}_{\mathfrak{z}}(x)]_1], [[\mathsf{W}_{\mathfrak{z}\omega}(x)]_1]\right)$ to $\left(W_{\mathfrak{z}}, W_{\mathfrak{z}\omega}\right)$ using the **Opening** procedure from the simulation of round 1.

We now prove that the output of $\mathcal{S}$ is statistically close to the view of the malicious provers in the real execution of the protocol in section 7.1.

Key ideas in the proof. Note that the adversary cannot introduce additive errors into rounds 1, 4, and 5 because these rounds do not call $\mathcal{F}_{\text{mult}}$ or $\mathcal{F}_{\text{polyMult}}$. All computations in these rounds are local. Rounds 2 and 3 are the only rounds in which additive errors are possible. The key idea is to prove two claims: first, in round 2, because we use the random masking from the Bar-Ilan and Beaver technique, it helps in arguing that in the actual round 2, the simulator can pick the output of the round 2 at random, and can store the additive errors introduced through the $\mathcal{F}_{\text{mult}}$ calls (formally proved in lemma 7.4). It can do the same for round 3 due to other independent random maskings (lemma 7.1) in the protocol (formally proved in lemma 7.5). The second crucial step is to prove that the simple verification check added in round 5 detects whether the additive errors were added or not. Specifically, we show that if errors were added, then the numerator of the opening proof polynomial will not be divisible by $(X - \mathfrak{z})$ (formally proved in lemma 7.7 and lemma 7.8). This is captured by condition C in 5b) in the simulator.

We need the following lemma that proves the randomness of the blinding scalars, which helps us to prove that the simulator can generate the output of rounds 1-3 uniformly at random.

Lemma 7.1. *Let us assume that $x, \mathfrak{z}$ satisfy the following conditions: $x \neq \mathfrak{z}$, $x \neq \mathfrak{z} \cdot \omega$, $x \notin H \cup \{0\}$, $\mathfrak{z} \notin H \cup \{0\}$[9]. Then, over the randomness of $(b_1, \ldots, b_{11})$, the following 11 values are independent and uniformly random in $\mathbb{F}$:*
$a_1 := b_1 \cdot x + b_2$, $a_2 := b_1 \cdot \mathfrak{z} + b_2$, $a_3 := b_3 \cdot x + b_4$, $a_4 := b_3 \cdot \mathfrak{z} + b_4$, $a_5 := b_5 \cdot x + b_6$, $a_6 := b_5 \cdot \mathfrak{z} + b_6$, $a_7 := b_7 \cdot x^2 + b_8 \cdot x + b_9$, $a_8 := b_7 \cdot (x\omega)^2 + b_8 \cdot (x\omega) + b_9$, $a_9 := b_7 \cdot (\mathfrak{z}\omega)^2 + b_8 \cdot (\mathfrak{z}\omega) + b_9$, $a_{10} := b_{10} \cdot x^n$, *and* $a_{11} := -b_{11}$.

Proof. For fixed values of x and $\mathfrak{z}$, the function that maps $(b_1, \ldots, b_{11})$ to $(a_1, \ldots, a_{11})$ is a full-rank linear function. Since $(b_1, \ldots, b_{11})$ is uniformly random over $\mathbb{F}^{11}$, $(a_1, \ldots, a_{11})$ is also uniformly random over $\mathbb{F}^{11}$.

□

[9] This indeed holds with high probability as $x, \mathfrak{z}$ are sampled independently and uniformly from $\mathbb{F}$.

Now we analyze each round and show that the distribution output by the simulator is statistically close to the real view.

Round 1. The adversary cannot introduce additive errors in round 1 because $\mathcal{F}_{\text{mult}}$ and $\mathcal{F}_{\text{polyMult}}$ are not called in this round. Next, note that:

$$\mathsf{a}(x) = a_1 \cdot Z_H(x) + \sum_{i=1}^{n} w_i \cdot L_i(x); \quad \mathsf{b}(x) = a_3 \cdot Z_H(x) + \sum_{i=1}^{n} w_{n+i} \cdot L_i(x)$$
$$\mathsf{c}(x) = a_5 \cdot Z_H(x) + \sum_{i=1}^{n} w_{2n+i} \cdot L_i(x)$$

By lemma 7.1, since (a_1, a_3, a_5) are random, therefore, given any wtn, $(\mathsf{a}(x), \mathsf{b}(x), \mathsf{c}(x))$ are independent and uniformly random in $\mathbb{F}$.

Round 2. If the adversary introduces no additive errors in round 2, then round 2 correctly computes the partial products (eq. (3)) and the polynomial $\mathsf{z}(X)$. This is formally proved in lemma 7.2. Next, even if the adversary introduces additive errors, the outputs of round 2 will be uniformly random and independent values, the same distribution that the simulator outputs. Finally, if the adversary introduces an additive error, then with overwhelming probabilities, at least one of the partial products PP_i will not satisfy eq. (3) (proved formally in lemma 7.3). Looking ahead, this will cause the protocol to abort in round 5 with overwhelming probability.

Lemma 7.2 (Correctness)**.** *If the adversary does not introduce any additive errors during round 2, then for every $i \in [n-1]$, PP_i is equal to*

$$\prod_{i' \in [i]} \frac{\left(w_{i'} + \beta\omega^{i'} + \gamma\right) \cdot \left(w_{n+i'} + \beta k_1 \omega^{i'} + \gamma\right) \cdot \left(w_{2n+i'} + \beta k_2 \omega^{i'} + \gamma\right)}{\left(w_{i'} + \sigma^*(i') \cdot \beta + \gamma\right) \cdot \left(w_{n+i'} + \sigma^*(n+i') \cdot \beta + \gamma\right) \cdot \left(w_{2n+i'} + \sigma^*(2n+i') \cdot \beta + \gamma\right)} \tag{3}$$

Proof. The following computations show the correctness. For every $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$: $t_k = r_k \cdot s_k$. For every $i \in [n-1]$ and $k = 6 \cdot (i-1)$:

$$
\begin{aligned}
g_i &= \frac{f_{i,1} \cdot f_{i,2} \cdot f_{i,3}}{f_{i,4} \cdot f_{i,5} \cdot f_{i,6}} \cdot \frac{t_{k+4} \cdot t_{k+5}}{t_{k+1} \cdot t_{k+2}} \\
&= \frac{\left(r_k \cdot e_{i,1} \cdot s_{k+1}\right) \cdot \left(r_{k+1} \cdot e_{i,2} \cdot s_{k+2}\right) \cdot \left(r_{k+2} \cdot e_{i,3} \cdot s_{k+3}\right)}{\left(s_{k+3} \cdot e_{i,4} \cdot r_{k+4}\right) \cdot \left(s_{k+4} \cdot e_{i,5} \cdot r_{k+5}\right) \cdot \left(s_{k+5} \cdot e_{i,6} \cdot r_{k+6}\right)} \cdot \frac{r_{k+4} \cdot s_{k+4} \cdot r_{k+5} \cdot s_{k+5}}{r_{k+1} \cdot s_{k+1} \cdot r_{k+2} \cdot s_{k+2}} \\
&= r_k \cdot \frac{e_{i,1} \cdot e_{i,2} \cdot e_{i,3}}{e_{i,4} \cdot e_{i,5} \cdot e_{i,6}} \cdot r_{k+6}^{-1} \\
&= r_{6\cdot(i-1)} \cdot \frac{e_{i,1} \cdot e_{i,2} \cdot e_{i,3}}{e_{i,4} \cdot e_{i,5} \cdot e_{i,6}} \cdot r_{6\cdot i}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
h_i &= \prod_{i' \in [i]} g_{i'} \\
&= \prod_{i' \in [i]} \left(r_{6\cdot(i'-1)} \cdot \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \cdot r_{6\cdot i'}^{-1} \right) \\
&= r_0 \cdot \left(\prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \right) \cdot r_{6\cdot i}^{-1}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{PP}_i &= \frac{h_i}{t_0} \cdot u_i \\
&= r_0 \cdot \left(\prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \right) \cdot r_{6\cdot i}^{-1} \cdot \frac{s_0 \cdot r_{6\cdot i}}{r_0 \cdot s_0} \\
&= \prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \\
&= \prod_{i' \in [i]} \frac{\left(w_{i'} + \beta\omega^{i'} + \gamma\right) \cdot \left(w_{n+i'} + \beta k_1 \omega^{i'} + \gamma\right) \cdot \left(w_{2n+i'} + \beta k_2 \omega^{i'} + \gamma\right)}{\left(w_{i'} + \sigma^*(i') \cdot \beta + \gamma\right) \cdot \left(w_{n+i'} + \sigma^*(n+i') \cdot \beta + \gamma\right) \cdot \left(w_{2n+i'} + \sigma^*(2n+i') \cdot \beta + \gamma\right)}
\end{aligned}
$$

□

Lemma 7.3. *If a non-zero additive error is introduced in round 2 during some call to $\mathcal{F}_{mult}$, then for some $i \in [n-1]$,*

$$
\mathsf{PP}_i \neq \prod_{i' \in [i]} \frac{\left(w_{i'} + \beta\omega^{i'} + \gamma\right) \cdot \left(w_{n+i'} + \beta k_1 \omega^{i'} + \gamma\right) \cdot \left(w_{2n+i'} + \beta k_2 \omega^{i'} + \gamma\right)}{\left(w_{i'} + \sigma^*(i') \cdot \beta + \gamma\right) \cdot \left(w_{n+i'} + \sigma^*(n+i') \cdot \beta + \gamma\right) \cdot \left(w_{2n+i'} + \sigma^*(2n+i') \cdot \beta + \gamma\right)}
$$

with overwhelming probability.

Proof. We begin by listing all the additive errors that the adversary can introduce in round 2. For each $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$: $t_k = r_k \cdot s_k + \varepsilon_{1,k}$.

For each $i \in [n-1]$: $u_i = s_0 \cdot r_{6\cdot i} + \varepsilon_{2,i}$

For each $i \in [n-1]$ and $j \in \{1,2,3\}$ and $k = 6 \cdot (i-1) + j$:

$$
f_{i,j} = r_{k-1} \cdot e_{i,j} \cdot s_k + \varepsilon_{3,i,j} \cdot s_k + \varepsilon_{4,i,j}
$$

For each $i \in [n-1]$ and $j \in \{4,5,6\}$ and $k = 6 \cdot (i-1) + j$:

$$
f_{i,j} = s_{k-1} \cdot e_{i,j} \cdot r_k + \varepsilon_{3,i,j} \cdot r_k + \varepsilon_{4,i,j}
$$

Now, we show how to express these as multiplicative errors. For each $k \in \{0, 1, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$: $t_k = r_k \cdot s_k \cdot \delta_{1,k}$, for $\delta_{1,k} = 1 + \frac{\varepsilon_{1,k}}{r_k \cdot s_k}$.

For each $i \in [n-1]$: $u_i = s_0 \cdot r_{6\cdot i} \cdot \delta_{2,i}$, for $\delta_{2,i} = 1 + \frac{\varepsilon_{2,i}}{s_0 \cdot r_{6\cdot i}}$.
For each $i \in [n-1]$, $j \in \{1,2,3\}$, and $k = 6\cdot(i-1)+j$:

$$f_{i,j} = r_{k-1} \cdot e_{i,j} \cdot s_k \cdot \delta_{3,i,j}, \text{ for } \delta_{3,i,j} = 1 + \frac{\varepsilon_{3,i,j}}{r_{k-1} \cdot e_{i,j}} + \frac{\varepsilon_{4,i,j}}{r_{k-1} \cdot e_{i,j} \cdot s_k}$$

For each $i \in [n-1]$, $j \in \{4,5,6\}$, and $k = 6\cdot(i-1)+j$:

$$f_{i,j} = s_{k-1} \cdot e_{i,j} \cdot r_k \cdot \delta_{3,i,j}, \text{ for } \delta_{3,i,j} = 1 + \frac{\varepsilon_{3,i,j}}{s_{k-1} \cdot e_{i,j}} + \frac{\varepsilon_{4,i,j}}{s_{k-1} \cdot e_{i,j} \cdot r_k}$$

Now we prove in the following claim that the δ errors contribute the following multiplicative error Δ_i to each PP_i for $i \in [n-1]$:

$$\Delta_i = \frac{\delta_{2,i}}{\delta_{1,0}} \cdot \prod_{i' \in [i]} \left(\frac{\delta_{3,i',1} \cdot \delta_{3,i',2} \cdot \delta_{3,i',3}}{\delta_{3,i',4} \cdot \delta_{3,i',5} \cdot \delta_{3,i',6}} \cdot \frac{\delta_{1,6\cdot(i'-1)+4} \cdot \delta_{1,6\cdot(i'-1)+5}}{\delta_{1,6\cdot(i'-1)+1} \cdot \delta_{1,6\cdot(i'-1)+2}} \right)$$

Claim 7.1. *For each $i \in [n-1]$: $\mathsf{PP}_i = \Delta_i \cdot \prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}}$*

Proof. The following calculations prove the claim. For every $i \in [n-1]$ and $k = 6\cdot(i-1)$:

$$\begin{aligned}
g_i &= \frac{f_{i,1} \cdot f_{i,2} \cdot f_{i,3}}{f_{i,4} \cdot f_{i,5} \cdot f_{i,6}} \cdot \frac{t_{k+4} \cdot t_{k+5}}{t_{k+1} \cdot t_{k+2}} \\
&= r_{6\cdot(i-1)} \cdot \frac{e_{i,1} \cdot e_{i,2} \cdot e_{i,3}}{e_{i,4} \cdot e_{i,5} \cdot e_{i,6}} \cdot r_{6\cdot i}^{-1} \cdot \frac{\delta_{3,i,1} \cdot \delta_{3,i,2} \cdot \delta_{3,i,3}}{\delta_{3,i,4} \cdot \delta_{3,i,5} \cdot \delta_{3,i,6}} \cdot \frac{\delta_{1,k+4} \cdot \delta_{1,k+5}}{\delta_{1,k+1} \cdot \delta_{1,k+2}} \\
h_i &= \prod_{i' \in [i]} g_{i'} \\
&= \prod_{i' \in [i]} \left(r_{6\cdot(i'-1)} \cdot \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \cdot r_{6\cdot i'}^{-1} \cdot \frac{\delta_{3,i',1} \cdot \delta_{3,i',2} \cdot \delta_{3,i',3}}{\delta_{3,i',4} \cdot \delta_{3,i',5} \cdot \delta_{3,i',6}} \cdot \frac{\delta_{1,6\cdot(i'-1)+4} \cdot \delta_{1,6\cdot(i'-1)+5}}{\delta_{1,6\cdot(i'-1)+1} \cdot \delta_{1,6\cdot(i'-1)+2}} \right) \\
&= r_0 \cdot \left(\prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}} \cdot \frac{\delta_{3,i',1} \cdot \delta_{3,i',2} \cdot \delta_{3,i',3}}{\delta_{3,i',4} \cdot \delta_{3,i',5} \cdot \delta_{3,i',6}} \cdot \frac{\delta_{1,6\cdot(i'-1)+4} \cdot \delta_{1,6\cdot(i'-1)+5}}{\delta_{1,6\cdot(i'-1)+1} \cdot \delta_{1,6\cdot(i'-1)+2}} \right) \cdot r_{6\cdot i}^{-1}
\end{aligned}$$

$$\mathsf{PP}_i = \frac{h_i}{t_0} \cdot u_i = \frac{h_i}{r_0 \cdot s_0 \cdot \delta_{1,0}} \cdot s_0 \cdot r_{6\cdot i} \cdot \delta_{2,i} = \Delta_i \cdot \prod_{i' \in [i]} \frac{e_{i',1} \cdot e_{i',2} \cdot e_{i',3}}{e_{i',4} \cdot e_{i',5} \cdot e_{i',6}}$$

□

Now, for any $i \in [n-1]$, Δ_i depends on the following multiplicative errors:

- $\delta_{1,k}$ for all $k < 6\cdot i$ such that $k = 0$ or $k \neq 0 \mod 3$
- $\delta_{2,i}$
- $\delta_{3,i',j}$ for all $i' \leq i$ and $j \in [6]$

Every multiplicative error $\delta_{1,k}, \delta_{2,i'}$, or $\delta_{3,i',j}$ is a factor in some Δ_i.

If the adversary introduces a non-zero error ϵ, it will cause one of the δ errors to be statistically close to uniformly random, and most of the δ errors will be independent of each other. We analyze four cases below and show that one of the δ values will be statistically close to uniformly random in each case.

Let us assume that for all k, $r_k \neq 0$ and $s_k \neq 0$, and for all (i,j), $e_{i,j} \neq 0$. This occurs with overwhelming probability over the randomness of the (r_k, s_k)-values and γ.

1. For any $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$, if $\varepsilon_{1,k} \neq 0$ then $\delta_{1,k}$ will be uniformly random over $\mathbb{F}\backslash\{1\}$ due to the randomness of r_k. This is because for any $\delta_{1,k} \in \mathbb{F}\backslash\{1\}$, there is a unique value of r_k that will produce that $\delta_{1,k}$-value:

$$r_k = \frac{\varepsilon_{1,k}}{s_k \cdot (\delta_{1,k} - 1)}$$

   We can also show that $\delta_{1,k}$ will be uniformly random over $\mathbb{F}\backslash\{1\}$ due to the randomness of s_k. This is because for any $\delta_{1,k} \in \mathbb{F}\backslash\{1\}$, there is a unique value of s_k that will produce that $\delta_{1,k}$-value:

$$s_k = \frac{\varepsilon_{1,k}}{r_k \cdot (\delta_{1,k} - 1)}$$

2. For any $i \in [n-1]$: if $\varepsilon_{2,i} \neq 0$, then $\delta_{2,i}$ will be uniformly random over $\mathbb{F}\backslash\{1\}$ due to the randomness of $r_{6 \cdot i}$. This is because for any $\delta_{2,i} \in \mathbb{F}\backslash\{1\}$, there is a unique value of $r_{6 \cdot i}$ that will produce that $\delta_{2,i}$-value:

$$r_{6 \cdot i} = \frac{\varepsilon_{2,i}}{s_0 \cdot (\delta_{2,i} - 1)}$$

3. For each $i \in [n-1]$, $j \in \{1, 2, 3\}$, and $k = 6 \cdot (i-1) + j$: if $\varepsilon_{3,i,j} \neq 0$ or $\varepsilon_{4,i,j} \neq 0$, then with overwhelming probability, $\delta_{3,i,j} \neq 1$. The only way that $\delta_{3,i,j} = 1$ is if:

$$s_k = -\frac{\varepsilon_{4,i,j}}{\varepsilon_{3,i,j}}$$

   But s_k is uniformly random and independent of $\varepsilon_{4,i,j}$ and $\varepsilon_{3,i,j}$, so $\delta_{3,i,j} \neq 1$ with overwhelming probability.

   Next, $\delta_{3,i,j}$ will be uniformly random over $\mathbb{F}\backslash\{1\}$ due to the randomness of r_{k-1}. This is because for any $\delta_{3,i,j} \in \mathbb{F}\backslash\{1\}$, there is a unique value of r_{k-1} that will produce that $\delta_{3,i,j}$-value:

$$r_{k-1} = \frac{\varepsilon_{3,i,j}}{(\delta_{3,i,j} - 1) \cdot e_{i,j}} + \frac{\varepsilon_{4,i,j}}{(\delta_{3,i,j} - 1) \cdot e_{i,j} \cdot s_k}$$

4. For each $i \in [n-1]$, $j \in \{4, 5, 6\}$, and $k = 6 \cdot (i-1) + j$, if $\varepsilon_{3,i,j} \neq 0$ or $\varepsilon_{4,i,j} \neq 0$, then $\delta_{3,i,j} \neq 1$ with overwhelming probability.

   Furthermore, $\delta_{3,i,j}$ will be uniformly random over $\mathbb{F}\backslash\{1\}$ due to the randomness of s_{k-1}.

Using the above conclusions, we now prove that for some $i \in [n-1]$, $\Delta_i \neq 1$ with overwhelming probability.

Claim 7.2. *Let i be the smallest value in $[n-1]$ for which at least one of the following is true:*

- $\delta_{1,k} \neq 0$ *for some* $k < 6 \cdot i$ *for which* $k = 0$ *or* $k \neq 0 \mod 3$
- $\delta_{2,i} \neq 0$
- $\delta_{3,i,j} \neq 0$ *for some* $i' \leq i$ *and* $j \in [6]$

Then with overwhelming probability, $\Delta_i \neq 1$.

Proof. Δ_i depends on the following δ errors, at least one of which is $\neq 1$.

- $\delta_{1,k}$ for some $k < 6 \cdot i$ for which $k = 0$ or $k \neq 0 \mod 3$

- $\delta_{2,i}$
- $\delta_{3,i,j}$ for some $j \in [6]$

We showed in the four cases above that the δ errors that are $\neq 1$ are uniformly and independently random over $\mathbb{F}\backslash\{1\}$. Particularly, we showed that:

- For every $k < 6 \cdot i$ for which $k = 0$ or $k \mod 6 \in \{1, 2\}$, if $\delta_{1,k} \neq 1$, then it is uniformly random due to the randomness of s_k.
- For every $k < 6 \cdot i$ for which $k \mod 6 \in \{4, 5\}$, if $\delta_{1,k} \neq 1$, then it is uniformly random due to the randomness of r_k.
- If $\delta_{2,i} \neq 1$, then it is uniformly random due to the randomness of $r_{6 \cdot i}$.
- If any of $(\delta_{3,i,1}, \delta_{3,i,2}, \delta_{3,i,3})$ are $\neq 1$, then they are uniformly random due to the randomness of $(r_{6\cdot(i-1)}, r_{6\cdot(i-1)+1}, r_{6\cdot(i-1)+2})$, respectively.
- If any of $(\delta_{3,i,4}, \delta_{3,i,5}, \delta_{3,i,6})$ are $\neq 1$, then they are uniformly random due to the randomness of $(s_{6\cdot(i-1)+3}, s_{6\cdot(i-1)+4}, s_{6\cdot(i-1)+5})$, respectively.

Since Δ_i is the product and quotient of δ-values that are either 1 or independent and uniformly random over $\mathbb{F}\backslash\{1\}$, then Δ_i is statistically close to uniform, and the probability that $\Delta_i = 1$ is negligible. □

Finally, claim 7.1 and claim 7.2 imply that the desired inequality holds for some $i \in [n-1]$ with overwhelming probability if the adversary introduces some additive error $\varepsilon \neq 0$ during a call to $\mathcal{F}_{\text{mult}}$ in round 2.

□

We finally use lemma 7.1 to prove that the output distribution of round 2 output by the simulator is statistically close to the real output in round 2.

Lemma 7.4. *The simulated view of round 2 is statistically close to the view of the malicious provers in round 2 of the real execution.*

Proof. The view of the malicious provers in round 2 include:

- t_k, for all $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k = 0$ or $k \neq 0 \mod 3$
- $f_{i,j}$ for all $(i, j) \in [n-1] \times [6]$
- $[z(x)]_1$

The simulator samples independent and uniformly random values for these outputs. We will show that in the real protocol as well, these values are independent and uniformly random.

For every $k \in \{0, \ldots, 6\cdot(n-1)\}$ such that $k = 0$ or $k \mod 6 \in \{1, 2\}$, $t_k = r_k \cdot s_k + \varepsilon_{1,k}$ is uniformly random over $\mathbb{F}$ due to the randomness of s_k. Furthermore, for every $k \in \{0, \ldots, 6 \cdot (n-1)\}$ such that $k \mod 6 \in \{4, 5\}$, t_k is uniformly random over $\mathbb{F}$ due to the randomness of r_k.
For each $i \in [n-1]$ and $j \in \{1, 2, 3\}$ and $k = 6 \cdot (i-1) + j$:

$$f_{i,j} = r_{k-1} \cdot e_{i,j} \cdot s_k + \varepsilon_{3,i,j} \cdot s_k + \varepsilon_{4,i,j}$$

Again by randomness of r_{k-1}, $f_{i,j}$ is uniformly random over $\mathbb{F}$.
For each $i \in [n-1]$ and $j \in \{4, 5, 6\}$ and $k = 6 \cdot (i-1) + j$:

$$f_{i,j} = s_{k-1} \cdot e_{i,j} \cdot r_k + \varepsilon_{3,i,j} \cdot r_k + \varepsilon_{4,i,j}$$

$f_{i,j}$ is uniformly random over $\mathbb{F}$ due to the randomness of s_{k-1}.
Finally, $[\mathsf{z}(x)]_1$ is uniformly random over $\mathbb{G}_1$ due to the randomness of a_7, since

$$\begin{aligned}\mathsf{z}(x) &= (b_7 \cdot x^2 + b_8 \cdot x + b_9) \cdot \mathsf{Z}_H(x) + \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(x) \\ &= a_7 \cdot \mathsf{Z}_H(x) + \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(x),\end{aligned}$$

and the value of a_7 is independent of $[\mathsf{a}(x)]_1$, $[\mathsf{b}(x)]_1$, $[\mathsf{c}(x)]_1$, $(r_k, s_k)_k$ (by lemma 7.1). Therefore, the value of $[\mathsf{z}(x)]_1$ is uniformly random in $\mathbb{G}_1$ and independent of all previous outputs of the protocol.

We have shown that the outputs of round 2 – $(t_k)_k$, $(f_{i,j})_{i,j}$, $[\mathsf{z}(x)]_1$ – are uniformly random due to independent sources of randomness: $(r_k, s_k)_k$, a_7. Therefore, these outputs will be independently random. This is the same distribution of outputs that the simulator generates, so the simulator correctly simulates round 2.

□

Round 3. In round 3, the adversary can introduce 7 error polynomials, one for each call to $\mathcal{F}_{\text{polyMult}}$. We prove in lemma 7.5 that the randomness of a_{10}, a_8, and a_{11} (guaranteed by lemma 7.1) ensures that the simulated view in round 3 is statistically close to uniformly random and independent of the simulated transcript so far.

Looking ahead, we also need to prove that in the evaluation for $\mathsf{t}(X)$, a_8 is multiplied by a non-zero value (to be able to use this random mask for correct simulation in future). In lemma 7.6, we prove this assuming that the computational Diffie-Hellman (CDH) assumption holds in $\mathbb{G}_1$. Specifically, a_8 is multiplied by:

$$\alpha \cdot (\mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x)) \tag{4}$$

While it is possible that *eq.* (4) equals zero if the adversary chooses $\varepsilon_5(X)$ and $\varepsilon_6(X)$ cleverly, we show that if that happens, then this adversary can be used to break CDH. We formally state and prove these lemmas below.

Lemma 7.5. *The simulated view of round 3 – $([\mathsf{t}_{\mathsf{lo}}(x)]_1, [\mathsf{t}_{\mathsf{mid}}(x)]_1, [\mathsf{t}_{\mathsf{hi}}(x)]_1)$ – is statistically close to uniform over $\mathbb{F}^3$, and independent of the simulated transcript until round 2.*

Proof. The randomness of $\big(\mathsf{t}_{\mathsf{lo}}(x), \mathsf{t}_{\mathsf{mid}}(x), \mathsf{t}_{\mathsf{hi}}(x)\big)$ is due to the randomness of (a_8, a_{10}, a_{11}), which are independent of the transcript so far. Now, the adversary may add an error polynomial $\varepsilon(X)$ wherever two secret-shared polynomials are multiplied. We can express $\mathsf{t}(X)$, with the adversary's errors included, as follows:

$$\begin{aligned}\text{Let } \mathsf{s}(X) := {} & (\mathsf{a}(X) \cdot \mathsf{b}(X) + \varepsilon_1(X)) \cdot \mathsf{q}_M(X) \\ & + \mathsf{a}(X) \cdot \mathsf{q}_L(X) + \mathsf{b}(X) \cdot \mathsf{q}_R(X) + \mathsf{c}(X) \cdot \mathsf{q}_O(X) + PI(X) + \mathsf{q}_C(X) \\ & + (\mathsf{a}'(X) \cdot \mathsf{b}'(X) \cdot \mathsf{c}'(X) + \varepsilon_2(X) \cdot \mathsf{c}'(X) + \varepsilon_3(X)) \cdot \mathsf{z}(X) \cdot \alpha + \varepsilon_4(X) \cdot \alpha \\ & - (\mathsf{a}''(X) \cdot \mathsf{b}''(X) \cdot \mathsf{c}''(X) + \varepsilon_5(X) \cdot \mathsf{c}''(X) + \varepsilon_6(X)) \cdot \mathsf{z}(X\omega) \cdot \alpha + \varepsilon_7(X) \cdot \alpha \\ & + (\mathsf{z}(X) - 1) \cdot L_1(X) \cdot \alpha^2\end{aligned}$$

$$\text{Then } \mathsf{t}(X) = \mathsf{Qt}\left(\frac{\mathsf{s}(X)}{\mathsf{Z}_H(X)}\right)$$

In the formula for $\mathsf{t}(x)$, a_8 is multiplied by $\alpha \cdot (\mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x))$. Now, consider for $\mathsf{s}'(X) = (\mathsf{a}''(X) \cdot \mathsf{b}''(X) \cdot \mathsf{c}''(X) + \varepsilon_5(X) \cdot \mathsf{c}''(X) + \varepsilon_6(X))$,

$$\begin{aligned}
\mathsf{s}''(X) &= \mathsf{s}(X) - \mathsf{s}'(X) \cdot \mathsf{z}(X\omega) \cdot \alpha \\
&= (\mathsf{a}(X) \cdot \mathsf{b}(X) + \varepsilon_1(X)) \cdot \mathsf{q}_M(X) \\
&\quad + \mathsf{a}(X) \cdot \mathsf{q}_L(X) + \mathsf{b}(X) \cdot \mathsf{q}_R(X) + \mathsf{c}(X) \cdot \mathsf{q}_O(X) + PI(X) + \mathsf{q}_C(X) \\
&\quad + (\mathsf{a}'(X) \cdot \mathsf{b}'(X) \cdot \mathsf{c}'(X) + \varepsilon_2(X) \cdot \mathsf{c}'(X) + \varepsilon_3(X)) \cdot \mathsf{z}(X) \cdot \alpha + \varepsilon_4(X) \cdot \alpha \\
&\quad + \varepsilon_7(X) \cdot \alpha \\
&\quad + (\mathsf{z}(X) - 1) \cdot L_1(X) \cdot \alpha^2,
\end{aligned}$$

and $\mathsf{z}_0(X) = \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(X)$, we have:

$$\begin{aligned}
\mathsf{z}(X) &= (b_7 \cdot X^2 + b_8 \cdot X + b_9) \cdot \mathsf{Z}_H(X) + \mathsf{z}_0(X) \\
\mathsf{s}(X) &= \mathsf{s}''(X) + \mathsf{s}'(X) \cdot \mathsf{z}(X\omega) \cdot \alpha \\
&= \mathsf{s}''(X) + \mathsf{s}'(X) \cdot \left((b_7 \cdot (X\omega)^2 + b_8 \cdot (X\omega) + b_9) \cdot \mathsf{Z}_H(X\omega) + \mathsf{z}_0(X\omega)\right) \cdot \alpha \\
&= \left(\mathsf{s}''(X) + \mathsf{s}'(X) \cdot \mathsf{z}_0(X\omega) \cdot \alpha\right) + \left(\mathsf{s}'(X) \cdot (b_7 \cdot (X\omega)^2 + b_8 \cdot (X\omega) + b_9) \cdot \mathsf{Z}_H(X) \cdot \alpha\right)
\end{aligned}$$

Note that $\mathsf{Z}_H(X\omega) = \mathsf{Z}_H(X) = X^n - 1$, therefore we have:

$$\begin{aligned}
\mathsf{t}(X) &= \mathsf{Qt}\left(\frac{\mathsf{s}(X)}{\mathsf{Z}_H(X)}\right) \\
&= \mathsf{Qt}\left(\frac{\mathsf{s}''(X) + \mathsf{s}'(X) \cdot \mathsf{z}_0(X\omega) \cdot \alpha}{\mathsf{Z}_H(X)}\right) + \left(\mathsf{s}'(X) \cdot (b_7 \cdot (X\omega)^2 + b_8 \cdot (X\omega) + b_9) \cdot \alpha\right) \\
\mathsf{t}(x) &= \mathsf{Qt}\left(\frac{\mathsf{s}''(X) + \mathsf{s}'(X) \cdot \mathsf{z}_0(X\omega) \cdot \alpha}{\mathsf{Z}_H(X)}\right)(x) + (\alpha \cdot \mathsf{s}'(x) \cdot a_8)
\end{aligned}$$

Note that a_8 is multiplied by $\alpha \cdot \mathsf{s}'(x)$.

Looking ahead, we will show that in the formula for $\mathsf{t}(x)$, the first term – $\mathsf{Qt}\left(\frac{\mathsf{s}''(X)+\mathsf{s}'(X)\cdot\mathsf{z}_0(X\omega)\cdot\alpha}{\mathsf{Z}_H(X)}\right)(x)$ – is independent of a_8, and in the second term, the coefficient $\alpha \cdot \mathsf{s}'(x)$ is non-zero and independent of a_8. Therefore, over the randomness of a_8, $\mathsf{t}(x)$ is uniformly random.

When we evaluate $\mathsf{t}(X)$ at $X = x$, then $\mathsf{z}(x\omega)$ is the only part of the expression that depends on a_8:

$$\mathsf{z}(x\omega) = a_8 \cdot \mathsf{Z}_H(x\omega) + \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(x\omega)$$

For any given $(\mathsf{PP}_1, \ldots, \mathsf{PP}_{n-1})$, $\mathsf{z}(x\omega)$ is uniformly random in $\mathbb{F}$, due to the randomness of a_8.

Furthermore, the adversary's error polynomials $(\varepsilon_1(X), \ldots, \varepsilon_7(X))$ are independent of a_8 because the perfect secrecy of the secret sharing hides the value of a_8 from the adversary.

If the CDH problem in $\mathbb{G}_1$ is hard for all PPT adversaries, then with overwhelming probability, the adversary will choose $\varepsilon_5(X)$ and $\varepsilon_6(X)$ such that

$$s'(x) = \mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x) \neq 0$$

See lemma 7.6 for the proof. In this case, $\mathsf{z}(x\omega)$ is multiplied by a non-zero value, so the value of $\mathsf{t}(x)$ will be uniformly random in $\mathbb{F}$, due to the randomness of a_8. Finally, $(\mathsf{t}_{\mathsf{lo}}(x), \mathsf{t}_{\mathsf{mid}}(x), \mathsf{t}_{\mathsf{hi}}(x))$ will be statistically close to independent and uniformly random in $\mathbb{F}$ due to the randomness of (a_8, a_{10}, a_{11}).

First, the protocol in round 3 derives three polynomials $(\mathsf{t}'_{\mathsf{lo}}(X), \mathsf{t}'_{\mathsf{mid}}(X), \mathsf{t}'_{\mathsf{hi}}(X))$ and computes:

$$\mathsf{t}_{\mathsf{lo}}(x) = a_{10} + \mathsf{t}'_{\mathsf{lo}}(x)$$
$$\mathsf{t}_{\mathsf{hi}}(x) = a_{11} + \mathsf{t}'_{\mathsf{hi}}(x)$$

For any values of $(\mathsf{t}'_{\mathsf{lo}}(x), \mathsf{t}'_{\mathsf{hi}}(x))$, the values of $(\mathsf{t}_{\mathsf{lo}}(x), \mathsf{t}_{\mathsf{hi}}(x))$ will be uniformly random and independent in $\mathbb{F}$, due to the randomness of (a_{10}, a_{11}).

Next, the protocol also computes $\mathsf{t}_{\mathsf{mid}}(x)$, which satisfies:

$$\mathsf{t}_{\mathsf{mid}}(x) = x^{-n} \cdot \left(\mathsf{t}(x) - \mathsf{t}_{\mathsf{lo}}(x) - x^{2n} \cdot \mathsf{t}_{\mathsf{hi}}(x)\right)$$

Then for any values of $(\mathsf{t}_{\mathsf{lo}}(x), \mathsf{t}_{\mathsf{hi}}(x))$, $\mathsf{t}_{\mathsf{mid}}(x)$ is uniformly random, due to the randomness of $\mathsf{t}(x)$ and a_8.

□

Lemma 7.6. *If CDH in $\mathbb{G}_1$ is hard for the adversary, then with overwhelming probability, the adversary will choose $\varepsilon_5(X), \varepsilon_6(X)$ such that*

$$\mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x) \neq 0$$

Proof. Let us assume toward contradiction that there is some adversary $\mathcal{A}$ such that with non-negligible probability in λ, $\mathcal{A}$ outputs polynomials $\varepsilon_5(X), \varepsilon_6(X)$ such that

$$\mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x) = 0 \tag{5}$$

Now we'll use $\mathcal{A}$ to construct an adversary that wins the computational Diffie-Hellman (CDH) game with non-negligible probability in λ:

1. The CDH adversary is given $([1]_1, [A]_1, [B]_1)$ where $(A, B) \xleftarrow{\$} \mathbb{F}^2$.

2. The CDH adversary samples $x \xleftarrow{\$} \mathbb{F}$ and $C \xleftarrow{\$} \mathbb{F}$ and uses x to prepare the common preprocessed input for Plonk. Then they run the simulator for round 1 of Plonk and they force the output of round 1 to be:

$$[\mathsf{a}(x)]_1 = [A]_1$$
$$[\mathsf{b}(x)]_1 = [B]_1$$
$$[\mathsf{c}(x)]_1 = [C]_1$$

3. In round 2 of Plonk, the CDH adversary computes β, γ. Then they sample $Z \xleftarrow{\$} \mathbb{F}$ and force the output of round 2 of Plonk to be:

$$[\mathsf{z}(x)]_1 = [Z]_1$$

4. In round 3 of Plonk, the Plonk adversary outputs polynomials $\varepsilon_5(X), \varepsilon_6(X)$.

5. The CDH adversary defines the following polynomial:

$$\mathsf{p}(X) := -\varepsilon_5(X) - \varepsilon_6(X) \cdot \frac{1}{\mathsf{c}(X) + \beta \mathsf{S}_{\sigma 3}(X) + \gamma}$$
$$- \mathsf{b}(X) \cdot (\beta \mathsf{S}_{\sigma 1}(X) + \gamma) - \mathsf{a}(X) \cdot (\beta \mathsf{S}_{\sigma 2}(X) + \gamma) - (\beta \mathsf{S}_{\sigma 1}(X) + \gamma) \cdot (\beta \mathsf{S}_{\sigma 2}(X) + \gamma)$$

   and computes $[\mathsf{p}(x)]_1$ using $x, C, [A]_1, [B]_1, \beta, \gamma$. Finally, the CDH adversary's output is $[\mathsf{p}(x)]_1$.

We will show that if eq. (5) is satisfied, then $[\mathsf{p}(x)]_1 = [A \cdot B]_1$. If eq. (5) is satisfied, then:

$$\mathsf{a}''(x) \cdot \mathsf{b}''(x) = -\varepsilon_5(x) - \varepsilon_6(x) \cdot \frac{1}{\mathsf{c}''(x)}$$

$$[\mathsf{a}(x) + \beta \mathsf{S}_{\sigma 1}(x) + \gamma] \cdot [\mathsf{b}(x) + \beta \mathsf{S}_{\sigma 2}(x) + \gamma] = -\varepsilon_5(x) - \varepsilon_6(x) \cdot \frac{1}{\mathsf{c}''(x)}$$

$$\mathsf{a}(x) \cdot \mathsf{b}(x) = -\varepsilon_5(x) - \varepsilon_6(x) \cdot \frac{1}{\mathsf{c}''(x)} - \mathsf{b}(x) \cdot (\beta \mathsf{S}_{\sigma 1}(x) + \gamma)$$

$$- \mathsf{a}(x) \cdot (\beta \mathsf{S}_{\sigma 2}(x) + \gamma) - (\beta \mathsf{S}_{\sigma 1}(x) + \gamma) \cdot (\beta \mathsf{S}_{\sigma 2}(x) + \gamma) = \mathsf{p}(x)$$

$$A \cdot B = \mathsf{p}(x)$$

Then $[\mathsf{p}(x)]_1 = [A \cdot B]_1$. We have shown that the CDH adversary wins the CDH game with non-negligible probability. This is a contradiction so the original assumption must have been false. In fact, with overwhelming probability, $\mathcal{A}$ will output polynomials $\varepsilon_5(X), \varepsilon_6(X)$ such that

$$\mathsf{a}''(x) \cdot \mathsf{b}''(x) \cdot \mathsf{c}''(x) + \varepsilon_5(x) \cdot \mathsf{c}''(x) + \varepsilon_6(x) \neq 0$$

□

Round 4. In round 4, the adversary cannot introduce additive errors because $\mathcal{F}_{\text{mult}}$ and $\mathcal{F}_{\text{polyMult}}$ are not called in this round. The simulated view of round 4 – $(\overline{a}, \overline{b}, \overline{c}, \overline{z}_\omega)$ – will be uniformly random in $\mathbb{F}$ and independent of all previous outputs, due to the randomness of (a_2, a_4, a_6, a_9) (lemma 7.1). $(\overline{a}, \overline{b}, \overline{c}, \overline{z}_\omega)$ satisfy the following equations:

$$\overline{a} = \mathsf{a}(\mathfrak{z}) = (b_1 \cdot \mathfrak{z} + b_2) \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_i \cdot L_i(\mathfrak{z}) = a_2 \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_i \cdot L_i(\mathfrak{z})$$

$$\overline{b} = \mathsf{b}(\mathfrak{z}) = (b_3 \cdot \mathfrak{z} + b_4) \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_{n+i} \cdot L_i(\mathfrak{z}) = a_4 \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_{n+i} \cdot L_i(\mathfrak{z})$$

$$\overline{c} = \mathsf{c}(\mathfrak{z}) = (b_5 \cdot \mathfrak{z} + b_6) \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_{2n+i} \cdot L_i(\mathfrak{z}) = a_6 \cdot Z_H(\mathfrak{z}) + \sum_{i=1}^{n} w_{2n+i} \cdot L_i(\mathfrak{z})$$

$$\overline{z}_\omega = \mathsf{z}(\mathfrak{z}\omega) = \left(b_7 \cdot (\mathfrak{z}\omega)^2 + b_8(\mathfrak{z}\omega) + b_9\right) \cdot \mathsf{Z}_H(\mathfrak{z}\omega) + \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(\mathfrak{z}\omega)$$

$$= a_9 \cdot \mathsf{Z}_H(\mathfrak{z}\omega) + \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(\mathfrak{z}\omega)$$

Due to the randomness of (a_2, a_4, a_6, a_9), $(\overline{a}, \overline{b}, \overline{c}, \overline{z}_\omega)$ are uniformly random and independent of the previous outputs of the protocol.
Finally, $\overline{\mathsf{s}}_{\sigma 1} = \mathsf{S}_{\sigma 1}(\mathfrak{z})$ and $\overline{\mathsf{s}}_{\sigma 2} = \mathsf{S}_{\sigma 2}(\mathfrak{z})$ are computed the same way in the real protocol as in the simulated protocol.

Round 5. In round 5, the protocol computes $V = \mathsf{Rd}\left(\frac{\mathsf{num}(X)}{X - \mathfrak{z}}\right)$, and the simulator handles this by setting $V = 0$ if and only if the following conditions are all satisfied:

Definition 7.1 (Conditions)**.**

1. In round 2, every call to $\mathcal{F}_{\text{mult}}$ has an additive error of $\varepsilon = 0$.
2. $0 = \varepsilon_2(X) = \varepsilon_3(X) = \varepsilon_5(X) = \varepsilon_6(X)$
3. $0 = \mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right)$

In lemma 7.7 (with the help of several observations formalized in lemma 7.8-lemma 7.12), we formally prove that the simulator handles this step correctly because in the real protocol, except with negligible probability, $V = 0$ if and only if the conditions of definition 7.1 are satisfied.
To complete the proof, in lemma 7.13, we finally show that if $\mathcal{F}_{\text{checkZero}}([V])$ accepts, then the simulator computes the correct values of $[\mathsf{W}_{\mathfrak{z}}(x)]_1$ and $[\mathsf{W}_{\mathfrak{z}\omega}(x)]_1$:

$$\mathsf{W}_{\mathfrak{z}}(x) = \frac{\mathsf{num}(x)}{x - \mathfrak{z}}; \quad \mathsf{W}_{\mathfrak{z}\omega}(x) = \frac{\mathsf{z}(x) - \overline{z}_{\omega}}{x - \mathfrak{z}\omega}$$

Lemma 7.7. *With overwhelming probability, $V = 0$ if and only if the conditions of definition 7.1 are satisfied.*

Proof. First, let us split $\mathsf{s}(X)$ into the terms $\mathsf{e}(X)$ that depend on the errors $(\varepsilon_1(X), \ldots, \varepsilon_7(X))$ and the terms $\tilde{\mathsf{s}}(X)$ that do not:

$$\begin{aligned}\tilde{\mathsf{s}}(X) &= \mathsf{a}(X) \cdot \mathsf{b}(X) \cdot \mathsf{q}_M(X) + \mathsf{a}(X) \cdot \mathsf{q}_L(X) + \mathsf{b}(X) \cdot \mathsf{q}_R(X) \\ &+ \mathsf{c}(X) \cdot \mathsf{q}_O(X) + PI(X) + \mathsf{q}_C(X) + \mathsf{a}'(X) \cdot \mathsf{b}'(X) \cdot \mathsf{c}'(X) \cdot \mathsf{z}(X) \cdot \alpha \\ &- \mathsf{a}''(X) \cdot \mathsf{b}''(X) \cdot \mathsf{c}''(X) \cdot \mathsf{z}(X\omega) \cdot \alpha + (\mathsf{z}(X) - 1) \cdot L_1(X) \cdot \alpha^2\end{aligned}$$

$$\begin{aligned}\mathsf{e}(X) &= \varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha \\ &+ (\varepsilon_2(X) \cdot \mathsf{c}'(X) + \varepsilon_3(X)) \cdot \mathsf{z}(X) \cdot \alpha \\ &- (\varepsilon_5(X) \cdot \mathsf{c}''(X) + \varepsilon_6(X)) \cdot \mathsf{z}(X\omega) \cdot \alpha\end{aligned}$$

Then, $\mathsf{s}(X) = \tilde{\mathsf{s}}(X) + \mathsf{e}(X)$ and $\mathsf{r}(\mathfrak{z}) = \tilde{\mathsf{s}}(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z}) \cdot \mathsf{t}(\mathfrak{z})$. We prove in lemma 7.8 that, with overwhelming probability, $V = 0$ if and only if

$$\begin{aligned}0 &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) \\ \text{and } 0 &= \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)\end{aligned}$$

Now consider the following exhaustive cases, which prove the lemma.

1. Case 1: Condition 1 is not satisfied (some call to $\mathcal{F}_{\text{mult}}$ in round 2 has an additive error of $\varepsilon \neq 0$). Then with overwhelming probability, $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) \neq 0$ (lemma 7.9), so $V \neq 0$.

2. Case 2: Condition 2 is not satisfied (at least one of $\varepsilon_2(X), \varepsilon_3(X), \varepsilon_5(X),$ or $\varepsilon_6(X)$ is non-zero). Then with overwhelming probability, $\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) \neq 0$ (lemma 7.11), so $V \neq 0$.

3. Case 3: Condition 2 is satisfied, but condition 3 is not satisfied:

$$\begin{aligned}0 &= \varepsilon_2(X) = \varepsilon_3(X) = \varepsilon_5(X) = \varepsilon_6(X) \\ 0 &\neq \mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right)\end{aligned}$$

   Then

$$\begin{aligned}\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) &= \mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right) \\ &\neq 0\end{aligned}$$

   by lemma 7.10. Then $V \neq 0$.

4. Case 4: Conditions 1, 2, and 3 are satisfied. Then $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) = 0$ (lemma 7.9), and

$$\begin{aligned}\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) &= \mathsf{Qt}\left(\frac{\varepsilon_1(X)\cdot \mathsf{q}_M(X) + \varepsilon_4(X)\cdot\alpha + \varepsilon_7(X)\cdot\alpha}{\mathsf{Z}_H(X)}\right)\\ &= 0\end{aligned}$$

by lemma 7.10. Then $V = 0$.

□

Now, we formally prove the equivalent conditions for $V = 0$, used in the proof above.

Lemma 7.8. *Except with negligible probability, $V = 0$ if and only if*

$$0 = \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)$$
$$\textit{and } 0 = \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)$$

Proof.

1. $V = \mathsf{num}(\mathfrak{z})$. Recall that $V = \mathsf{Rd}\left(\frac{\mathsf{num}(X)}{X-\mathfrak{z}}\right)$. The remainder is a polynomial of degree 0 because the divisor, $X - \mathfrak{z}$, is a polynomial of degree 1. Furthermore, $\mathsf{num}(X) - V$ is divisible by $X - \mathfrak{z}$. Therefore $V = \mathsf{num}(\mathfrak{z})$.

2. $V = \tilde{\mathsf{s}}(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z})\cdot\mathsf{t}(\mathfrak{z})$.

$$\begin{aligned}V &= \mathsf{num}(\mathfrak{z})\\ &= \mathsf{r}(\mathfrak{z}) + v(\mathsf{a}(\mathfrak{z}) - \overline{a}) + v^2(\mathsf{b}(\mathfrak{z}) - \overline{b}) + v^3(\mathsf{c}(\mathfrak{z}) - \overline{c})\\ &\quad + v^4(\mathsf{S}_{\sigma 1}(\mathfrak{z}) - \overline{\mathsf{s}}_{\sigma 1}) + v^5(\mathsf{S}_{\sigma 2}(\mathfrak{z}) - \overline{\mathsf{s}}_{\sigma 2})\\ &= \mathsf{r}(\mathfrak{z})\\ &= \tilde{\mathsf{s}}(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z})\cdot\mathsf{t}(\mathfrak{z})\end{aligned}$$

3. $V = \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z})\cdot\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z})$. This is because:

$$\begin{aligned}\mathsf{t}(X) &= \mathsf{Qt}\left(\frac{\mathsf{s}(X)}{\mathsf{Z}_H(X)}\right)\\ &= \mathsf{Qt}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) + \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)\end{aligned}$$

$$\begin{aligned}\tilde{\mathsf{s}}(X) - \mathsf{Z}_H(X)\cdot\mathsf{t}(X) &= \tilde{\mathsf{s}}(X) - \mathsf{Z}_H(X)\cdot\mathsf{Qt}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X)\cdot\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)\\ &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X)\cdot\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)\end{aligned}$$

$$\begin{aligned}V &= \tilde{\mathsf{s}}(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z})\cdot\mathsf{t}(\mathfrak{z})\\ &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z})\cdot\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z})\end{aligned}$$

4. With overwhelming probability over the randomness of $\mathfrak{z}$, $V = 0$ if and only if $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$.

   First, if $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$, then

$$\begin{aligned} V &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z}) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) \\ &= 0 \end{aligned}$$

   Second, if the polynomial $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) \neq 0$, then with overwhelming probability,

$$V = \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) - \mathsf{Z}_H(\mathfrak{z}) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) \neq 0$$

   This is because the functions $\tilde{\mathsf{s}}(X), \mathsf{Z}_H(X), \mathsf{e}(X)$ are determined by the end of round 3, but $\mathfrak{z}$ is not sampled until round 4, so $\mathfrak{z}$ is independently random from $\tilde{\mathsf{s}}(X), \mathsf{Z}_H(X), \mathsf{e}(X)$. Evaluating a non-zero polynomial at a random input will produce 0 with only negligible probability.

5. $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$ if and only if $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) = 0$ and $\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$.

   This is because if

$$\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$$

   then

$$\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) = \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)$$

   By definition, $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)$ is not divisible by $\mathsf{Z}_H(X)$ unless $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) = 0$. Since $\mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)$ is divisible by $\mathsf{Z}_H(X)$, then

$$\begin{aligned} 0 &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) \\ 0 &= \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) \\ 0 &= \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) \end{aligned}$$

   Conversely, if $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) = 0$ and $\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$, then $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) - \mathsf{Z}_H(X) \cdot \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = 0$.

6. In summary: with overwhelming probability over the randomness of $\mathfrak{z}$, $V = 0$ if and only if:

$$\begin{aligned} 0 &= \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) \\ \text{and } 0 &= \mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) \end{aligned}$$

□

Lemma 7.9. *With overwhelming probability,* $\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right) = 0$ *if and only if every call to* $\mathcal{F}_{mult}$ *in round 2 has an additive error of* $\varepsilon = 0$.

Proof.

1.

$$\begin{aligned}\tilde{s}(X) = {} & a(X) \cdot b(X) \cdot q_M(X) + a(X) \cdot q_L(X) + b(X) \cdot q_R(X) + c(X) \cdot q_O(X) + PI(X) + q_C(X) \\ & + a'(X) \cdot b'(X) \cdot c'(X) \cdot z(X) \cdot \alpha \\ & - a''(X) \cdot b''(X) \cdot c''(X) \cdot z(X\omega) \cdot \alpha \\ & + (z(X) - 1) \cdot L_1(X) \cdot \alpha^2\end{aligned}$$

2. The first line of $\tilde{s}(X)$ does not contribute to $\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right)$. Since the witness is accepting:

$$0 = \mathsf{Rd}\left(\frac{a(X) \cdot b(X) \cdot q_M(X) + a(X) \cdot q_L(X) + b(X) \cdot q_R(X) + c(X) \cdot q_O(X) + PI(X) + q_C(X)}{Z_H(X)}\right)$$

3. The fourth line of $\tilde{s}(X)$ does not contribute to $\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right)$. Since $z(\omega^1) = 1$:

$$0 = \mathsf{Rd}\left(\frac{(z(X) - 1) \cdot L_1(X)}{Z_H(X)}\right)$$

4. Then $\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right)$ only depends on the second and third lines of $\tilde{s}(X)$.

$$\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right) = \mathsf{Rd}\left(\frac{a'(X) \cdot b'(X) \cdot c'(X) \cdot z(X) - a''(X) \cdot b''(X) \cdot c''(X) \cdot z(X\omega)}{Z_H(X)}\right) \cdot \alpha$$

5. If every call to $\mathcal{F}_{\text{mult}}$ in round 2 has an additive error of $\varepsilon = 0$, then $\mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right) = 0$. This is because for every $i \in [n-1]$:

$$\mathsf{PP}_i = \prod_{i' \in [i]} \frac{\left(w_{i'} + \beta\omega^{i'} + \gamma\right) \cdot \left(w_{n+i'} + \beta k_1 \omega^{i'} + \gamma\right) \cdot \left(w_{2n+i'} + \beta k_2 \omega^{i'} + \gamma\right)}{\left(w_{i'} + \sigma^*(i') \cdot \beta + \gamma\right) \cdot \left(w_{n+i'} + \sigma^*(n+i') \cdot \beta + \gamma\right) \cdot \left(w_{2n+i'} + \sigma^*(2n+i') \cdot \beta + \gamma\right)}$$

Then for every $i \in [n]$,

$$\begin{aligned}z(\omega^{i+1}) &= \frac{\left(w_i + \beta\omega^i + \gamma\right) \cdot \left(w_{n+i} + \beta k_1 \omega^i + \gamma\right) \cdot \left(w_{2n+i} + \beta k_2 \omega^i + \gamma\right)}{\left(w_i + \sigma^*(i) \cdot \beta + \gamma\right) \cdot \left(w_{n+i} + \sigma^*(n+i) \cdot \beta + \gamma\right) \cdot \left(w_{2n+i} + \sigma^*(2n+i) \cdot \beta + \gamma\right)} \cdot z(\omega^i) \\ z(\omega^{i+1}) &= \frac{a'(\omega^i) \cdot b'(\omega^i) \cdot c'(\omega^i)}{a''(\omega^i) \cdot b''(\omega^i) \cdot c''(\omega^i)} \cdot z(\omega^i)\end{aligned}$$

$$\begin{aligned}0 = {} & a'(\omega^i) \cdot b'(\omega^i) \cdot c'(\omega^i) \cdot z(\omega^i) \\ & - a''(\omega^i) \cdot b''(\omega^i) \cdot c''(\omega^i) \cdot z(\omega^{i+1})\end{aligned}$$

Therefore,

$$\begin{aligned}0 &= \mathsf{Rd}\left(\frac{a'(X) \cdot b'(X) \cdot c'(X) \cdot z(X) - a''(X) \cdot b''(X) \cdot c''(X) \cdot z(X\omega)}{Z_H(X)}\right) \\ &= \mathsf{Rd}\left(\frac{\tilde{s}(X)}{Z_H(X)}\right)\end{aligned}$$

6. It remains to show that if a non-zero additive error is introduced in round 2 during some call to $\mathcal{F}_{\text{mult}}$, then $\mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right) \neq 0$.

   If a non-zero additive error is introduced in round 2 during some call to $\mathcal{F}_{\text{mult}}$, then there is some $i \in [n-1]$, such that

$$\mathsf{PP}_i \neq \prod_{i' \in [i]} \frac{\left(w_{i'} + \beta\omega^{i'} + \gamma\right) \cdot \left(w_{n+i'} + \beta k_1 \omega^{i'} + \gamma\right) \cdot \left(w_{2n+i'} + \beta k_2 \omega^{i'} + \gamma\right)}{(w_{i'} + \sigma^*(i') \cdot \beta + \gamma) \cdot (w_{n+i'} + \sigma^*(n+i') \cdot \beta + \gamma) \cdot (w_{2n+i'} + \sigma^*(2n+i') \cdot \beta + \gamma)} \tag{6}$$

   with overwhelming probability (lemma 7.3).

7. Let us pick i to be the smallest value in $[n-1]$ for which eq. (6) holds. Then:

$$\mathsf{PP}_i \neq \frac{(w_i + \beta\omega^i + \gamma) \cdot (w_{n+i} + \beta k_1 \omega^i + \gamma) \cdot (w_{2n+i} + \beta k_2 \omega^i + \gamma)}{(w_i + \sigma^*(i) \cdot \beta + \gamma) \cdot (w_{n+i} + \sigma^*(n+i) \cdot \beta + \gamma) \cdot (w_{2n+i} + \sigma^*(2n+i) \cdot \beta + \gamma)} \cdot \mathsf{PP}_{i-1}$$

$$\mathsf{z}(\omega^{i+1}) = \frac{\mathsf{a}'(\omega^i) \cdot \mathsf{b}'(\omega^i) \cdot \mathsf{c}'(\omega^i)}{\mathsf{a}''(\omega^i) \cdot \mathsf{b}''(\omega^i) \cdot \mathsf{c}''(\omega^i)} \cdot \mathsf{z}(\omega^i)$$

$$\begin{aligned} 0 = {} & \mathsf{a}'(\omega^i) \cdot \mathsf{b}'(\omega^i) \cdot \mathsf{c}'(\omega^i) \cdot \mathsf{z}(\omega^i) \\ & - \mathsf{a}''(\omega^i) \cdot \mathsf{b}''(\omega^i) \cdot \mathsf{c}''(\omega^i) \cdot \mathsf{z}(\omega^{i+1}) \end{aligned}$$

8. Since ω^i is a root of $\mathsf{Z}_H(X)$, that means:

$$0 \neq \mathsf{Rd}\left(\frac{\mathsf{a}'(X) \cdot \mathsf{b}'(X) \cdot \mathsf{c}'(X) \cdot \mathsf{z}(X) - \mathsf{a}''(X) \cdot \mathsf{b}''(X) \cdot \mathsf{c}''(X) \cdot \mathsf{z}(X\omega)}{\mathsf{Z}_H(X)}\right)$$

$$0 \neq \mathsf{Rd}\left(\frac{\mathsf{a}'(X) \cdot \mathsf{b}'(X) \cdot \mathsf{c}'(X) \cdot \mathsf{z}(X) - \mathsf{a}''(X) \cdot \mathsf{b}''(X) \cdot \mathsf{c}''(X) \cdot \mathsf{z}(X\omega)}{\mathsf{Z}_H(X)}\right) \cdot \alpha$$

$$0 \neq \mathsf{Rd}\left(\frac{\tilde{\mathsf{s}}(X)}{\mathsf{Z}_H(X)}\right)$$

□

Lemma 7.10. *If* $0 = \varepsilon_2(X) = \varepsilon_3(X) = \varepsilon_5(X) = \varepsilon_6(X)$, *then*

$$\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = \mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right)$$

Proof.

$$\begin{aligned} \mathsf{e}(X) = {} & \varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha \\ & + (\varepsilon_2(X) \cdot \mathsf{c}'(X) + \varepsilon_3(X)) \cdot \mathsf{z}(X) \cdot \alpha \\ & - (\varepsilon_5(X) \cdot \mathsf{c}''(X) + \varepsilon_6(X)) \cdot \mathsf{z}(X\omega) \cdot \alpha \end{aligned}$$

If $0 = \varepsilon_2(X) = \varepsilon_3(X) = \varepsilon_5(X) = \varepsilon_6(X)$, then

$$\mathsf{e}(X) = \varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha$$

$$\mathsf{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right) = \mathsf{Qt}\left(\frac{\varepsilon_1(X) \cdot \mathsf{q}_M(X) + \varepsilon_4(X) \cdot \alpha + \varepsilon_7(X) \cdot \alpha}{\mathsf{Z}_H(X)}\right)$$

□

Lemma 7.11. *If at least one of $\varepsilon_2(X), \varepsilon_3(X), \varepsilon_5(X),$ or $\varepsilon_6(X)$ is non-zero, then with overwhelming probability, $0 \neq \mathrm{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)$.*

Proof.

1. Next, let's define some random variables. As before, let

   - $a_6 := b_5 \cdot \mathfrak{z} + b_6$
   - $a_9 := b_7 \cdot (\mathfrak{z}\omega)^2 + b_8 \cdot (\mathfrak{z}\omega) + b_9$

   In addition, let

   - $a_{12} = b_7 \cdot \mathfrak{z}^2 + b_8 \cdot \mathfrak{z} + b_9$

   We claim that over the randomness of $(b_5, \ldots, b_9, \mathfrak{z})$, (a_6, a_9, a_{12}) are statistically close to independent and uniformly random over $\mathbb{F}$. This is because for any $\mathfrak{z} \neq 0$ the function that maps $(b_5, \ldots, b_9)$ to (a_6, a_9, a_{12}) is a rank-3 linear function. Since $(b_5, \ldots, b_9)$ are uniformly random and independent, (a_6, a_9, a_{12}) are uniformly random and independent as well.

2. The round-3 error polynomials $(\varepsilon_1, \ldots, \varepsilon_7)$ are independent of $(a_6, a_9, a_{12}, \mathfrak{z})$. This is because the adversary must specify $(\varepsilon_1, \ldots, \varepsilon_7)$ during round 3 based on the values they have seen so far, and none of the values output so far depend on $(a_6, a_9, a_{12}, \mathfrak{z})$.

3. $\mathsf{z}(\mathfrak{z})$ is uniformly random due to the randomness of a_{12}, and $\mathsf{z}(\mathfrak{z}\omega)$ is uniformly random due to the randomness of a_9.

$$
\begin{aligned}
\text{Let } \mathsf{z}_0(X) &= \sum_{i=1}^{n} \mathsf{PP}_{i-1} \cdot \mathsf{L}_i(X) \\
\mathsf{z}(X) &= (b_7 \cdot X^2 + b_8 \cdot X + b_9) \cdot \mathsf{Z}_H(X) + \mathsf{z}_0(X) \\
\mathsf{z}(\mathfrak{z}) &= a_{12} \cdot \mathsf{Z}_H(\mathfrak{z}) + \mathsf{z}_0(\mathfrak{z}) \\
\mathsf{z}(\mathfrak{z}\omega) &= a_9 \cdot \mathsf{Z}_H(\mathfrak{z}\omega) + \mathsf{z}_0(\mathfrak{z}\omega)
\end{aligned}
$$

   Note that $\mathfrak{z}$, $\mathsf{Z}_H(\mathfrak{z})$, and $\mathsf{z}_0(\mathfrak{z})$ are independent of (a_9, a_{12}), so $\mathsf{z}(\mathfrak{z})$ and $\mathsf{z}(\mathfrak{z}\omega)$ are uniformly random due to the randomness of a_{12} and a_9 respectively.

4. If at least one of $\varepsilon_2(X), \varepsilon_3(X), \varepsilon_5(X),$ or $\varepsilon_6(X)$ is non-zero, then $\mathsf{e}(\mathfrak{z})$ is statistically close to uniformly random, due to the randomness of $(a_6, a_9, a_{12}, \mathfrak{z})$.

   First, let us expand the expression of $\mathsf{e}(\mathfrak{z})$ to show a_9 and a_{12}:

$$
\begin{aligned}
\mathsf{e}(\mathfrak{z}) &= \varepsilon_1(\mathfrak{z}) \cdot \mathsf{q}_M(\mathfrak{z}) + \varepsilon_4(\mathfrak{z}) \cdot \alpha + \varepsilon_7(\mathfrak{z}) \cdot \alpha \\
&\quad + (\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot \mathsf{z}(\mathfrak{z}) \cdot \alpha \\
&\quad - (\varepsilon_5(\mathfrak{z}) \cdot \mathsf{c}''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot \mathsf{z}(\mathfrak{z}\omega) \cdot \alpha \\
&= \varepsilon_1(\mathfrak{z}) \cdot \mathsf{q}_M(\mathfrak{z}) + \varepsilon_4(\mathfrak{z}) \cdot \alpha + \varepsilon_7(\mathfrak{z}) \cdot \alpha \\
&\quad + (\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot (a_{12} \cdot \mathsf{Z}_H(\mathfrak{z}) + \mathsf{z}_0(\mathfrak{z})) \cdot \alpha \\
&\quad - (\varepsilon_5(\mathfrak{z}) \cdot \mathsf{c}''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot (a_9 \cdot \mathsf{Z}_H(\mathfrak{z}\omega) + \mathsf{z}_0(\mathfrak{z}\omega)) \cdot \alpha \\
&= \varepsilon_1(\mathfrak{z}) \cdot \mathsf{q}_M(\mathfrak{z}) + \varepsilon_4(\mathfrak{z}) \cdot \alpha + \varepsilon_7(\mathfrak{z}) \cdot \alpha \\
&\quad + (\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot \mathsf{Z}_H(\mathfrak{z}) \cdot \alpha \cdot a_{12} + (\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot \mathsf{z}_0(\mathfrak{z}) \cdot \alpha \\
&\quad - (\varepsilon_5(\mathfrak{z}) \cdot \mathsf{c}''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot \mathsf{Z}_H(\mathfrak{z}\omega) \cdot \alpha \cdot a_9 - (\varepsilon_5(\mathfrak{z}) \cdot \mathsf{c}''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot \mathsf{z}_0(\mathfrak{z}\omega) \cdot \alpha
\end{aligned}
$$

Second, the only term that depends on a_{12} is

$$(\varepsilon_2(\mathfrak{z}) \cdot c'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot Z_H(\mathfrak{z}) \cdot \alpha \cdot a_{12}$$

Furthermore, if at least one of $\varepsilon_2(X)$ or $\varepsilon_3(X)$ is non-zero, then

$$(\varepsilon_2(\mathfrak{z}) \cdot c'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z})) \cdot Z_H(\mathfrak{z}) \cdot \alpha \neq 0$$

with overwhelming probability over the randomness of a_6 and $\mathfrak{z}$. This follows from lemma 7.12. In this case, a_{12} has a non-zero coefficient, so $e(\mathfrak{z})$ is uniformly random due to the randomness of a_{12}.

Third, the only term that depends on a_9 is

$$-(\varepsilon_5(\mathfrak{z}) \cdot c''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot Z_H(\mathfrak{z}\omega) \cdot \alpha \cdot a_9$$

Furthermore, if at least one of $\varepsilon_5(X)$ or $\varepsilon_6(X)$ is non-zero, then

$$-(\varepsilon_5(\mathfrak{z}) \cdot c''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z})) \cdot Z_H(\mathfrak{z}\omega) \cdot \alpha \neq 0$$

with overwhelming probability over the randomness of a_6 and $\mathfrak{z}$. This follows from lemma 7.12. In this case, a_9 has a non-zero coefficient, so $e(\mathfrak{z})$ is uniformly random due to the randomness of a_9.

5. $\mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right)$ is independent of $(a_6, a_9, a_{12}, \mathfrak{z})$.

$$\mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right) = \sum_{i \in [n]} L_i(X) \cdot e(\omega^i)$$

For any $i \in \{0, \ldots, n-1\}$,

$$\begin{aligned} e(\omega^i) = {} & \varepsilon_1(\omega^i) \cdot q_M(\omega^i) + \varepsilon_4(\omega^i) \cdot \alpha + \varepsilon_7(\omega^i) \cdot \alpha \\ & + \left(\varepsilon_2(\omega^i) \cdot (w_{2n+i} + \beta \cdot k_2 \cdot \omega^i + \gamma) + \varepsilon_3(\omega^i)\right) \cdot \mathsf{PP}_{i-1} \cdot \alpha \\ & - \left(\varepsilon_5(\omega^i) \cdot (w_{2n+i} + \beta \cdot S_{\sigma 3}(\omega^i) + \gamma) + \varepsilon_6(\omega^i)\right) \cdot \mathsf{PP}_i \cdot \alpha \end{aligned}$$

Here, we used the fact that $z(\omega^i) = \mathsf{PP}_{i-1}$.

Note that every component of the formula for $e(\omega^i)$ is fixed by the end of round 3, before $\mathfrak{z}$ is chosen, and is independent of $(a_6, a_9, a_{12}, \mathfrak{z})$. Therefore, $e(\omega^i)$ is independent of $(a_6, a_9, a_{12}, \mathfrak{z})$ and so is $\mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right)$.

6. $\mathsf{Qt}\left(\frac{e(X)}{Z_H(X)}\right)(\mathfrak{z})$ is statistically close to uniformly random due to the randomness of $(a_6, a_9, a_{12}, \mathfrak{z})$.

$$e(X) = \mathsf{Qt}\left(\frac{e(X)}{Z_H(X)}\right) \cdot Z_H(X) + \mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right)$$

$$\mathsf{Qt}\left(\frac{e(X)}{Z_H(X)}\right)(\mathfrak{z}) = \frac{1}{Z_H(\mathfrak{z})} \cdot \left(e(\mathfrak{z}) - \mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right)(\mathfrak{z})\right)$$

We know that $e(\mathfrak{z})$ is statistically close to uniformly random due to the randomness of $(a_6, a_9, a_{12}, \mathfrak{z})$, and $\mathsf{Rd}\left(\frac{e(X)}{Z_H(X)}\right)(\mathfrak{z})$ is independent of $(a_6, a_9, a_{12}, \mathfrak{z})$. Therefore, $\mathsf{Qt}\left(\frac{e(X)}{Z_H(X)}\right)(\mathfrak{z})$ is statistically close to uniformly random due to the randomness of $(a_6, a_9, a_{12}, \mathfrak{z})$.

7. With overwhelming probability, $\mathrm{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(\mathfrak{z}) \neq 0$. That implies that

$$\mathrm{Qt}\left(\frac{\mathsf{e}(X)}{\mathsf{Z}_H(X)}\right)(X) \neq 0$$

□

Lemma 7.12.

1. *If at least one of $\varepsilon_2(X)$ or $\varepsilon_3(X)$ is non-zero, then*

$$\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z}) \neq 0$$

   with overwhelming probability over the randomness of a_6 and $\mathfrak{z}$.

2. *Likewise, if at least one of $\varepsilon_5(X)$ or $\varepsilon_6(X)$ is non-zero, then*

$$\varepsilon_5(\mathfrak{z}) \cdot \mathsf{c}''(\mathfrak{z}) + \varepsilon_6(\mathfrak{z}) \neq 0$$

   with overwhelming probability over the randomness of a_6 and $\mathfrak{z}$.

Proof.

1. We will just prove the first item because the proof of the second item is similar.
2. If $\varepsilon_2(X) = 0$ and $\varepsilon_3(X) \neq 0$, then

$$\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z}) = \varepsilon_3(\mathfrak{z})$$

   With overwhelming probability over the randomness of $\mathfrak{z}$, $\varepsilon_3(\mathfrak{z}) \neq 0$.
3. Next, if $\varepsilon_2(X) \neq 0$, then with overwhelming probability over the randomness of $\mathfrak{z}$, $\varepsilon_2(\mathfrak{z}) \neq 0$. Then the only way that $\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z}) = 0$ is if

$$\mathsf{c}'(\mathfrak{z}) = -\frac{\varepsilon_3(\mathfrak{z})}{\varepsilon_2(\mathfrak{z})}$$

4. $\mathsf{c}'(\mathfrak{z})$ is uniformly random due to the randomness of a_6.

$$\begin{aligned}
\mathsf{c}'(\mathfrak{z}) &= \mathsf{c}(\mathfrak{z}) + \beta k_2 \mathfrak{z} + \gamma \\
&= (b_5 \cdot \mathfrak{z} + b_6) \cdot \mathsf{Z}_H(\mathfrak{z}) + \sum_{i \in [n]} w_{2n+i} \cdot \mathsf{L}_i(\mathfrak{z}) + \beta k_2 \mathfrak{z} + \gamma \\
&= a_6 \cdot \mathsf{Z}_H(\mathfrak{z}) + \sum_{i \in [n]} w_{2n+i} \cdot \mathsf{L}_i(\mathfrak{z}) + \beta k_2 \mathfrak{z} + \gamma
\end{aligned}$$

   Since $\mathsf{Z}_H(\mathfrak{z}) \neq 0$, $\mathsf{c}'(\mathfrak{z})$ is uniformly random, over the randomness of a_6.
5. Over the randomness of a_6,

$$\Pr\left[\mathsf{c}'(\mathfrak{z}) = -\frac{\varepsilon_3(\mathfrak{z})}{\varepsilon_2(\mathfrak{z})}\right] = \frac{1}{|\mathbb{F}|} = \mathsf{negl}(\lambda)$$

6. In summary, if $\varepsilon_2(X) \neq 0$ or $\varepsilon_3(X) \neq 0$, then with overwhelming probability, $\varepsilon_2(\mathfrak{z}) \cdot \mathsf{c}'(\mathfrak{z}) + \varepsilon_3(\mathfrak{z}) \neq 0$.

□

Lemma 7.13. *In the real protocol, if $V = 0$, then:*

$$\mathsf{W}_{\mathfrak{z}}(x) = \frac{\mathsf{num}(x)}{x - \mathfrak{z}}$$
$$\mathsf{W}_{\mathfrak{z}\omega}(x) = \frac{\mathsf{z}(x) - \overline{z}_\omega}{x - \mathfrak{z}\omega}$$

Proof. We know that $V = \mathsf{num}(\mathfrak{z})$ (from the proof of lemma 7.8). If $V = 0$, then $\mathsf{num}(\mathfrak{z}) = 0$, so $\mathsf{num}(X)$ is divisible by $(X - \mathfrak{z})$. Therefore:

$$\mathsf{W}_{\mathfrak{z}}(X) = \mathsf{Qt}\left(\frac{\mathsf{num}(X)}{X - \mathfrak{z}}\right) = \frac{\mathsf{num}(X)}{X - \mathfrak{z}}$$
$$\mathsf{W}_{\mathfrak{z}}(x) = \frac{\mathsf{num}(x)}{x - \mathfrak{z}}$$

Next, since $\mathsf{z}(\mathfrak{z}\omega) = \overline{z}_\omega$, $\mathsf{z}(X) - \overline{z}_\omega$ is divisible by $(X - \mathfrak{z}\omega)$, so

$$\mathsf{W}_{\mathfrak{z}\omega}(X) = \mathsf{Qt}\left(\frac{\mathsf{z}(X) - \overline{z}_\omega}{X - \mathfrak{z}\omega}\right) = \frac{\mathsf{z}(X) - \overline{z}_\omega}{X - \mathfrak{z}\omega}$$
$$\mathsf{W}_{\mathfrak{z}\omega}(x) = \frac{\mathsf{z}(x) - \overline{z}_\omega}{x - \mathfrak{z}\omega}$$

□

This completes the proof of t-zero-knowledge against $t < N/2$ malicious provers.

Acknowledgements

S. Garg was supported in part by the AFOSR Award FA9550-24-1-0156 and research grants from the Bakar Fund, J. P. Morgan Faculty Research Award, Supra Inc., Sui Foundation, and the Stellar Development Foundation. A. Jain was supported in part by NSF CAREER 1942789, Johns Hopkins University Catalyst award, JP Morgan Faculty Award, and research gifts from Ethereum Foundation, Stellar Development Foundation, and Cisco.

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. pages 315–334, 2018.

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. pages 67–97, 2019.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptol.*, 30(4):989–1066, 2017.

[BDO14] Carsten Baum, Ivan Damgård, and Claudio Orlandi. Publicly auditable secure multi-party computation. pages 175–196, 2014.

[BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. pages 869–886, 2019.

[BGIN20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. pages 244–276, 2020.

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. pages 457–485, 2021.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). pages 1–10, 1988.

[BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. pages 201–209, 1989.

[BKa] Christopher Bender and Joseph Kraut. Implementation of renegade.

[BKb] Christopher Bender and Joseph Kraut. Renegade whitepaper.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. pages 499–530, 2023.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). page 462, 1988.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. pages 471–488, 2008.

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. pages 34–64, 2018.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. pages 738–768, 2020.

[DEN24] Anders P. K. Dalskov, Daniel Escudero, and Ariel Nof. Fully secure MPC and zk-FLIOP over rings: New constructions, improvements and extensions. pages 136–169, 2024.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. pages 572–590, 2007.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. pages 643–662, 2012.

[FL19] Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. pages 1557–1571, 2019.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194, 1987.

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. pages 4427–4444, 2023.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. pages 495–504, 2014.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. pages 218–229, 1987.

[Gol01] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. pages 305–326, 2016.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. pages 618–646, 2020.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). pages 723–732, 1992.

[LN17] Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. pages 259–276, 2017.

[LZW+24a] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. Scalable collaborative zk-SNARK and its application to efficient proof outsourcing. Cryptology ePrint Archive, Paper 2024/940, 2024.

[LZW+24b] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-SNARK: Fully distributed proof generation and malicious security. Cryptology ePrint Archive, Paper 2024/143, 2024.

[Mic94] Silvio Micali. CS proofs (extended abstracts). pages 436–453, 1994.

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority MPC by batchwise multiplication verification. pages 321–339, 2018.

[OB] Alexander Ozdemir and Dan Boneh. Implementation of collaborative zksnarks.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. pages 4291–4308, 2022.

[ST19] Nigel P. Smart and Younes Talibi Alaoui. Distributing any elliptic curve based protocol. pages 342–366, 2019.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. pages 733–764, 2019.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). pages 162–167, 1986.

A Multiparty Computation Functionalities and Sub-Protocols

We now give a complete description of the MPC functionalities described in section 3.1.

A.1 Standard Honest-Majority MPC Functionalities

In this section, we give a formal definition of some standard MPC ideal functionalities from [CGH+18], which also gave protocols that securely compute these functionalities with abort in the presence of malicious adversaries in the honest-majority setting.

$\mathcal{F}_{\text{input}}$: This functionality allows the parties to secret-share M inputs. It is formally described below.

Functionality $\mathcal{F}_{\text{input}}$

For each input $i \in [M]$:

1: $\mathcal{F}_{\text{input}}$ receives $v_i \in \mathbb{F}$ from one of the parties.
2: $\mathcal{F}_{\text{input}}$ receives from the adversary the values $\{\alpha_i^j\}_{\forall j \in C}$, which represent the corrupted parties' shares of $[v_i]$.
3: $\mathcal{F}_{\text{input}}$ sets $[v_i]_C = \{\alpha_i^j\}_{\forall j \in C}$ and computes:

$$[v_i] = (v_i^1, \ldots, v_i^n) = \mathsf{share}\,(v_i, [v_i]_C)$$

4: $\mathcal{F}_{\text{input}}$ sends to each party $j \in [n]$ its share v_i^j.

$\mathcal{F}_{\text{rand}}$: This functionality generates secret shares of a random value. It is formally described below.

Functionality $\mathcal{F}_{\text{rand}}$

1: The adversary sends to $\mathcal{F}_{\text{rand}}$ the values $\{\alpha_j\}_{\forall j \in C}$.
2: $\mathcal{F}_{\text{rand}}$ samples $r \xleftarrow{\$} \mathbb{F}$.
3: $\mathcal{F}_{\text{rand}}$ sets $[r]_C = \{\alpha_j\}_{\forall j \in C}$ and computes:

$$[r] = \mathsf{share}\,(r, [r]_C)$$

4: $\mathcal{F}_{\text{rand}}$ sends each honest party $j \in \mathcal{H}$ their share r_j.

Figure 1: Functionality $\mathcal{F}_{\text{rand}}$

$\mathcal{F}_{\text{coin}}$: This functionality generates a random field element. It is formally described below.

Functionality $\mathcal{F}_{\text{coin}}$

1: $\mathcal{F}_{\text{coin}}$ samples $r \xleftarrow{\$} \mathbb{F}$ and sends r to all parties.

$\mathcal{F}_{\text{checkZero}}$: This functionality is given $[x]$ and determines if $x = 0$. False negatives and false positives are possible.

Functionality $\mathcal{F}_{\text{checkZero}}$

1: $\mathcal{F}_{\text{checkZero}}$ receives $[x]_{\mathcal{H}}$ from the honest parties, and uses this to reconstruct x.
2: If $x = 0$, then $\mathcal{F}_{\text{checkZero}}$ sends 0 to the adversary. The adversary responds with accept or reject, and $\mathcal{F}_{\text{checkZero}}$ forwards the adversary's response to the honest parties.
3: If $x \neq 0$, then:

1. With probability $\frac{1}{|\mathbb{F}|}$, $\mathcal{F}_{\text{checkZero}}$ sends accept to all parties.

2. With probability $1 - \frac{1}{|\mathbb{F}|}$, $\mathcal{F}_{\text{checkZero}}$ sends reject to all parties.

Figure 2: Functionality $\mathcal{F}_{\text{checkZero}}$

$\mathcal{F}_{\text{mult}}$: This functionality is used to multiply two secret-shared values $[x]$ and $[y]$. However, a malicious adversary is allowed to specify an additive error ε. In the end, $\mathcal{F}_{\text{mult}}$ actually computes $[x \cdot y + \varepsilon]$. Although $\mathcal{F}_{\text{mult}}$ is defined for malicious adversaries, the protocol that computes $\mathcal{F}_{\text{mult}}$ in [CGH+18] is a standard semi-honest-secure protocol for multiplication.

Functionality $\mathcal{F}_{\text{mult}}$

1: $\mathcal{F}_{\text{mult}}$ receives $[x]_{\mathcal{H}}$ and $[y]_{\mathcal{H}}$ from the honest parties and uses the shares to reconstruct x and y.
2: $\mathcal{F}_{\text{mult}}$ computes the shares for all parties:

$$[x] = \text{share}(x, [x]_{\mathcal{H}})$$
$$[y] = \text{share}(y, [y]_{\mathcal{H}})$$

and sends $[x]_C$ and $[y]_C$ to the adversary.
3: The adversary responds with an additive error $\varepsilon \in \mathbb{F}$ and the corrupted parties' shares of the output, $\{\alpha_j\}_{\forall j \in C}$.
4: $\mathcal{F}_{\text{mult}}$ sets $[z]_C = \{\alpha_j\}_{\forall j \in C}$ and computes:

$$z = x \cdot y + \varepsilon$$
$$[z] = \text{share}\,(z, [z]_C)$$

5: $\mathcal{F}_{\text{mult}}$ sends to each honest party $j \in \mathcal{H}$ its share z_j.

Figure 3: Functionality $\mathcal{F}_{\text{mult}}$

A.2 Multiplying two secret shared polynomials

We first formally define the functionality $\mathcal{F}_{\text{polyMult}}$, which takes two secret-shared polynomials $[\mathrm{a}(X)]$ and $[\mathrm{b}(X)]$ of degree d_a and d_b, respectively, and outputs the shares of their product (up to additive attacks), i.e., $[\mathrm{a}(X) \cdot \mathrm{b}(X) + \varepsilon(X)]$.

Functionality $\mathcal{F}_{\text{polyMult}}$

1: $\mathcal{F}_{\text{polyMult}}$ receives $[\mathrm{a}(X)]_{\mathcal{H}}$ and $[\mathrm{b}(X)]_{\mathcal{H}}$ from the honest parties and uses the shares to reconstruct $\mathrm{a}(X)$ and $\mathrm{b}(X)$.
2: $\mathcal{F}_{\text{polyMult}}$ computes the corrupted parties' shares – $[\mathrm{a}(X)]_C$ and $[\mathrm{b}(X)]_C$ – and sends them to $\mathcal{S}$.
3: $\mathcal{S}$ responds with a polynomial $\varepsilon(X) \in \mathbb{F}[X]$ for which $\deg(\varepsilon(X)) \leq d_a + d_b$, as well as the corrupted parties' shares of the output, $[\mathrm{c}(X)]_C$.
4: $\mathcal{F}_{\text{polyMult}}$ computes:

$$\mathrm{c}(X) = \mathrm{a}(X) \cdot \mathrm{b}(X) + \varepsilon(X)$$
$$[\mathrm{c}(X)] = \text{share}\,(\mathrm{c}(X), [\mathrm{c}(X)]_C)$$

5: $\mathcal{F}_{\text{polyMult}}$ sends to each honest party $j \in \mathcal{H}$ its share of $[\mathrm{c}(X)]$.

The protocol that realizes $\mathcal{F}_{\text{polyMult}}$ is defined in fig. 4. The protocol calls $\mathcal{F}_{\text{mult}}$ $(d_a + d_b + 1)$ times, and each time, the adversary can specify an additive error. The combined effect is that the adversary adds an error polynomial $\varepsilon(X)$ of degree $\leq d_a + d_b$, and the output is $[\mathrm{a}(X) \cdot \mathrm{b}(X) + \varepsilon(X)]$.

Protocol PolyMult

1: *Inputs:* $[\mathrm{a}(X)] = \left([\alpha_0], \ldots, [\alpha_{d_a}]\right)$ and $[\mathrm{b}(X)] = \left([\beta_0], \ldots, [\beta_{d_b}]\right)$

2: Let $R = \{1, \rho, \rho^2, \ldots, \rho^{d_a+d_b}\}$ be the $(d_a + d_b + 1)$-th roots of unity in $\mathbb{F}$. Use the FFT to compute the evaluations of $\mathrm{a}(X)$ and $\mathrm{b}(X)$ on R. For each $j \in \{0, \ldots, (d_a + d_b)\}$,

$$[\mathrm{a}(\rho^j)] = \sum_{i \in \{0,\ldots,d_a\}} [\alpha_i] \cdot \rho^{i \cdot j} \tag{7}$$

$$[\mathrm{b}(\rho^j)] = \sum_{i \in \{0,\ldots,d_b\}} [\beta_i] \cdot \rho^{i \cdot j} \tag{8}$$

3: For each $j \in \{0, \ldots, (d_a + d_b)\}$, compute:

$$[\mathrm{c}(\rho^j)] = \mathcal{F}_{\text{mult}}([\mathrm{a}(\rho^j)], [\mathrm{b}(\rho^j)])$$

4: Use the inverse FFT to compute the coefficients of $[\mathrm{c}(X)]$: $([\gamma_0], \ldots, [\gamma_{d_a+d_b}])$. For each $k \in \{0, \ldots, d_a + d_b\}$:

$$[\gamma_k] = \sum_{j \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot [\mathrm{c}(\rho^j)] \cdot \rho^{-k \cdot j} \tag{9}$$

5: Output $[\mathrm{c}(X)]$.

Figure 4: Protocol PolyMult

Theorem A.1. *The protocol in fig. 4 securely computes the functionality $\mathcal{F}_{polyMult}$ with abort in the $\mathcal{F}_{mult}$-hybrid model in the presence of malicious adversaries who control $t < N/2$ parties.*

Proof. Let $\mathcal{A}$ be the real-world adversary, and let us construct the ideal-world simulator $\mathcal{S}$, which runs $\mathcal{A}$ internally, as follows:

Simulator $\mathcal{S}$

1. $\mathcal{S}$ receives $[\mathrm{a}(X)]_C$ and $[\mathrm{b}(X)]_C$ from $\mathcal{F}_{\text{polyMult}}$.
2. $\mathcal{F}_{\text{mult}}$: For each $j \in \{0, \ldots, (d_a + d_b)\}$, $\mathcal{S}$ simulates $\mathcal{F}_{\text{mult}}([\mathrm{a}(\rho^j)], [\mathrm{b}(\rho^j)])$ as follows:
   (a) $\mathcal{S}$ computes

$$[\mathrm{a}(\rho^j)]_C = \sum_{i \in \{0,\ldots,d_a\}} [\alpha_i]_C \cdot \rho^{i \cdot j}$$

$$[\mathrm{b}(\rho^j)]_C = \sum_{i \in \{0,\ldots,d_b\}} [\beta_i]_C \cdot \rho^{i \cdot j}$$

   (b) $\mathcal{S}$ sends $[\mathrm{a}(\rho^j)]_C$ and $[\mathrm{b}(\rho^j)]_C$ to $\mathcal{A}$.
   (c) $\mathcal{S}$ receives from $\mathcal{A}$ the error ε_j and the corrupted parties' shares of the output $[\mathrm{c}(\rho^j)]_C$.

3. $\mathcal{S}$ computes:

$$\varepsilon(X) = \sum_{j,k \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot \varepsilon_j \cdot \rho^{-k \cdot j} \cdot X^k$$

4. For each $k \in \{0, \ldots, (d_a + d_b)\}$, $\mathcal{S}$ computes:

$$[\gamma_k]_C = \sum_{j \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot [\mathrm{c}(\rho^j)]_C \cdot \rho^{-k \cdot j}$$

and sets $[\mathrm{c}(X)]_C = \left([\gamma_0]_C, \ldots, [\gamma_{d_a+d_b}]_C\right)$.

5. $\mathcal{S}$ sends $\varepsilon(X)$ and $[\mathrm{c}(X)]_C$ to $\mathcal{F}_{\text{polyMult}}$.

$\mathcal{S}$ sends the correct values of $[\mathrm{a}(\rho^j)]_C$ and $[\mathrm{b}(\rho^j)]_C$ to $\mathcal{A}$ because it simply computes $[\mathrm{a}(\rho^j)]_C$ and $[\mathrm{b}(\rho^j)]_C$ based on the formulas (eqs. (7) and (8)) used in the real-world protocol.

Next, the real and simulated protocols compute the same $\mathrm{c}(X)$. In fig. 4, let ε_j be the additive error introduced during the computation of $\mathcal{F}_{\text{mult}}([\mathrm{a}(\rho^j)], [\mathrm{b}(\rho^j)])$, and let

$$\varepsilon(X) = \sum_{j,k \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot \varepsilon_j \cdot \rho^{-k \cdot j} \cdot X^k$$

Claim A.1. *In the protocol given in fig. 4,* $\mathrm{c}(X) = \mathrm{a}(X) \cdot \mathrm{b}(X) + \varepsilon(X)$

Proof.

$$\begin{aligned}
\mathrm{c}(X) &= \sum_{k \in \{0,\ldots,(d_a+d_b)\}} \gamma_k \cdot X^k = \sum_{k \in \{0,\ldots,(d_a+d_b)\}} \left(\sum_{j \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot \mathrm{c}(\rho^j) \cdot \rho^{-k \cdot j} \right) \cdot X^k \\
&= \sum_{j,k} |R|^{-1} \cdot \left(\mathrm{a}(\rho^j) \cdot \mathrm{b}(\rho^j) + \varepsilon_j\right) \cdot \rho^{-k \cdot j} \cdot X^k \\
&= \sum_{j,k} |R|^{-1} \cdot \left(\left(\sum_{i \in \{0,\ldots,d_a\}} \sum_{i' \in \{0,\ldots,d_b\}} \alpha_i \cdot \rho^{i \cdot j} \cdot \beta_{i'} \cdot \rho^{i' \cdot j} \right) + \varepsilon_j \right) \cdot \rho^{-k \cdot j} \cdot X^k \\
&= \left(\sum_{i,i',j,k} |R|^{-1} \cdot \alpha_i \cdot \beta_{i'} \cdot \rho^{(i+i'-k) \cdot j} \cdot X^k \right) + \left(\sum_{j,k} |R|^{-1} \cdot \varepsilon_j \cdot \rho^{-k \cdot j} \cdot X^k \right) \\
&= \left(\sum_{i,i',k} \alpha_i \cdot \beta_{i'} \cdot X^k \cdot \left(\sum_j |R|^{-1} \cdot \rho^{(i+i'-k) \cdot j} \right) \right) + \varepsilon(X) \\
&= \sum_{i,i'} \alpha_i \cdot \beta_{i'} \cdot X^{i+i'} + \varepsilon(X) = \left(\sum_i \alpha_i \cdot X^i \right) \cdot \left(\sum_{i'} \beta_{i'} \cdot X^{i'} \right) + \varepsilon(X) \\
&= \mathrm{a}(X) \cdot \mathrm{b}(X) + \varepsilon(X)
\end{aligned}$$

We used the fact that

$$\sum_{j \in \{0,\ldots,(d_a+d_b)\}} |R|^{-1} \cdot \rho^{(i+i'-k) \cdot j} = \begin{cases} 1, & k = i + i' \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

□

Finally, simulator $\mathcal{S}$ computes the shares $[\mathrm{c}(X)]_C$ that the corrupted parties would hold if they computed eq. (9) honestly. Then $\mathcal{F}_{\text{polyMult}}$ chooses $[\mathrm{c}(X)]_{\mathcal{H}}$ to be consistent with $\mathrm{c}(X)$ and $[\mathrm{c}(X)]_C$. This distribution of $[\mathrm{c}(X)]_{\mathcal{H}}$ is the same as in the real world. □