Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs

Elette Boyle[*] Niv Gilboa[†] Yuval Ishai [‡] Ariel Nof [§]

November 26, 2019

Abstract

Secure multiparty computation enables a set of parties to securely carry out a joint computation on their private inputs without revealing anything but the output. A particularly motivated setting is that of three parties with a single corruption (hereafter denoted 3PC). This 3PC setting is particularly appealing for two main reasons: (1) it admits *more efficient* MPC protocols than in other standard settings; (2) it allows in principle to achieve *full security* (and fairness).

Highly efficient protocols exist within this setting with security against a *semi-honest* adversary; however, a significant gap remains between these and protocols with stronger security against a *malicious* adversary.

In this paper, we narrow this gap within concretely efficient protocols. More explicitly, we have the following contributions:

- **Concretely Efficient Malicious 3PC.** We present an optimized 3PC protocol for arithmetic circuits over rings with (amortized) communication of 1 ring element per multiplication gate per party, matching the best semi-honest protocols. The protocol applies also to Boolean circuits, significantly improving over previous protocols even for small circuits.

  Our protocol builds on recent techniques of Boneh et al. (Crypto 2019) for sublinear zero-knowledge proofs on distributed data, together with an efficient semi-honest protocol based on replicated secret sharing (Araki et al., CCS 2016).

  We present a concrete analysis of communication and computation costs, including several optimizations. For example, for 40-bit statistical security, and Boolean circuit with a million (nonlinear) gates, the overhead on top of the semi-honest protocol can involve less than 0.5KB of communication *for the entire circuit*, while the computational overhead is dominated by roughly 30 multiplications per gate in the field $\mathbb{F}_{2^{47}}$. In addition, we implemented and benchmarked the protocol for varied circuit sizes.

- **Full Security.** We augment the 3PC protocol to further provide *full security* (with guaranteed output delivery) while maintaining amortized 1 ring element communication per party per multiplication gate, and with hardly any impact on concrete efficiency. This is contrasted with the best previous 3PC protocols from the literature, which allow a corrupt party to mount a denial-of-service attack without being detected.

[*] IDC Herzliya, Israel email: eboyle@alum.mit.edu.

[†] Ben-Gurion University, Israel email: gilboan@bgu.ac.il.

[‡] Technion, Israel. email: yuvali@cs.technion.ac.il.

[§] Technion, Israel. email: ariel.nof@biu.ac.il.

1 Introduction

Protocols for secure computation [Yao86, GMW87, BGW88, CCD88] enable a set of parties with private inputs to compute a joint function of their inputs while revealing nothing but the output. Secure computation provides a general-purpose method for computing across sensitive data items, as well as for eliminating single points of failure. As a result, a major research effort has been undertaken within the applied cryptography and security community in improving the concrete efficiency of secure computation protocols.

Among the most studied settings for prospective practical applications is that of three-party secure computation tolerating a single corrupt party (henceforth referred to as 3PC). This setting is highly motivated by two significant features: As the minimal nontrivial setting with an honest majority, it seems to have the greatest potential for admitting fast, simple protocols. Indeed, 3PC protocols can make use of simple information-theoretic secret sharing, whereas two-party secure computation protocols rely on more complex public-key primitives such as oblivious transfer or homomorphic encryption that incur higher communication and computation costs. Further, it is the minimal setting in which one can achieve *full* security, where honest parties are guaranteed to learn the output at the conclusion of the protocol. This lies again in contrast to settings without honest majority, where one can only hope to provide a weaker notion of *security with abort*, leaving the protocol open to denial-of-service attacks from a corrupt party.

Within each setting, two classic adversary models are typically considered: *semi-honest* (where the adversary follows the protocol specification but may try to learn more than allowed from the protocol transcript) and *malicious* (where the adversary can run an arbitrary polynomial-time attack strategy). Highly efficient protocols for 3PC have been obtained in the semi-honest model; see [BLW08, AFL^{+}16, CGH^{+}18, CCPS19] and references therein. However, despite tremendous progress, there is still a significant gap in concrete complexity costs between solutions in the semi-honest and malicious settings, particularly for Boolean circuits.

Since concretely efficient 3PC protocols are computationally simple and only employ symmetric cryptography, their overall cost is typically dominated by communication rather than computation.[1] Consequently, the main efforts in this line focus on minimizing the ratio between the amortized *communication* costs of malicious and semi-honest solutions (while maintaining reasonable computation complexity). For the case of evaluating an arithmetic circuit over a ring, the best such semi-honest protocols have an amortized communication cost of only 1 ring element per multiplication gate per party [AFL^{+}16]. (In this work, the term "ring" refers to either a finite field or a ring of the form $\mathbb{Z}_{2^k}$; the term "amortized communication cost" refers to the ratio between the communication and the number of gates when the latter tends to infinity.) To date, the best such *malicious*-secure protocols that have been optimized and implemented communicate 2 field elements per multiplication gate per party over large fields (of size comparable to 2^σ for statistical security parameter σ) [CGH^{+}18, NV18], or alternatively 7 bits per AND/OR gate per party for Boolean circuits (where XOR/NOT gates are for free) [ABF^{+}17].

[1] More broadly, in the context of parallelizable tasks, computation is generally considered a cheaper resource, since buying more hardware is easier than improving bandwidth. For instance, quoting from the recent PSI implementation work of [IKN^{+}19]: "A rule of thumb we encountered is that doubling the communication cost of a solution is equivalent to increasing the computational cost by a factor of 20× - 40×."

1.1 Our Contributions

In this paper, we present an optimized high-throughput protocol for the malicious 3PC setting of three parties and an honest majority. Our protocol begins with the efficient semi-honest protocol of [AFL$^+$16] that relies on replicated secret-sharing.[2] Then, building on the recent work of Boneh et al. [BBC$^+$19], compiles it to be maliciously secure by adding a single step at the end of the protocol which verifies the correctness of behavior during the execution. Our protocol has very low communication: it requires each party to send only one field/ring element per multiplication gate (amortized over the circuit). This is exactly the amortized communication cost of the best known semi-honest protocols (that do not employ public-key cryptography). We measured the concrete efficiency of our protocol and report its practicality. Our findings indicate that the improved amortized communication cost does not conflict with concrete efficiency: the communication advantage over earlier protocols kicks in even for fairly small circuits, and the computational overhead is reasonable.

As mentioned, our starting point towards achieving malicious security is the recent work by Boneh et al. [BBC$^+$19], which introduced the new notion of zero-knowledge proofs over distributed data. Such proofs involve a single prover and multiple verifiers. The prover wishes to prove the correctness of a statement over some data which is known entirely to the prover but is distributed among the verifiers. The protocols from [BBC$^+$19] have sublinear proof size when applied to simple statements. The authors have also identified the potential of this tool for MPC in the presence of malicious adversaries, as in such protocols the information is secret-shared between the parties, and each party needs to prove that it behaved correctly to the other parties given that information. This can be useful in particular in the setting of three parties with one corruption, since in this case we are guaranteed that only the prover *or* one of the verifiers is corrupted, leading to much more efficient protocols. However, the asymptotically efficient protocols presented in [BBC$^+$19] use several layers of abstraction, and were not optimized or even analyzed for concrete efficiency.

In this paper, we take a step forward and show how to turn the protocols from [BBC$^+$19] into efficient zero-knowledge protocols for verifying that all the messages sent in a concretely efficient 3PC protocol were correct. Moreover, adapting the protocol to a specific setting enables us to introduce an optimized version and to prove that it securely computes an ideal functionality $\mathcal{F}_{\mathsf{vrfy}}$ that is used in the main protocol. The sublinear communication complexity of the zero-knowledge protocol ensures that, amortized over the circuit, the communication cost of the final 3PC protocol does not increase beyond the cost of the semi-honest baseline, and moreover the concrete overhead is small. Our initial protocol works only over finite fields, but then we show how to extend it to work also over the ring $\mathbb{Z}_{2^k}$ by using variants of techniques from [BBC$^+$19]. This requires working over a larger ring, which blows up the cost of the zero-knowledge proofs by a small factor. However, since the proofs are still sublinear in the size of the circuit, this does not increase the amortized communication cost. Finally, while the basic version of our protocol (as well as the protocol from [BBC$^+$19]) only achieves "security with abort," we show how to achieve *full security,* including fairness and guaranteed output delivery, at a minor additional cost.

We present a detailed concrete analysis of both the computational and the communication cost of our protocol and discuss multiple optimizations that improve the overall efficiency. We also ran experiments to measure the actual performance of our protocol. As the semi-honest protocol has been shown to achieve very high throughput in different settings [AFL$^+$16, ABF$^+$17, CGH$^+$18], we

[2] One could alternatively use other semi-honest 3PC protocols from the literature, such as the one from [KKW18]. See Remark 3.4 for discussion.

focus only on the verification step, which anyway can be executed in parallel and independently of the main semi-honest protocol. Our experiments demonstrate the computational effort (measured by its running time) required by each party in the verification step. The experiment was run in an AWS c5.9xlarge instance (Intel Xeon Platinum 8000 series with clock speed of up to 3.5 GHz) for different sizes of arithmetic circuits over a 31-bit Mersenne field. (These fields enable particularly fast multiplications and are big enough for most arithmetic computations.) In this setting, applying the verification protocol to all 1 million gates simultaneously requires 2.3 seconds. However, when splitting the gates into 8 groups of 2^{16} gates, where each group is being verified separately, it is possible to complete the verification of all the groups (without parallelizing the computation!) in *less than a second*, while the communication overhead is only 0.04 field elements per gate on average (thus adding less than 5% of the communication required by the semi-honest protocol). This shows a promising potential for usefulness in real-world applications where bandwidth is a bottleneck.

Finally, we present additional efficiency improvements (at the expense of additional rounds of interaction[3]) by optimizing and concretely analyzing a recursive proof composition technique suggested in [BBC+19]. For 40 bits of statistical security, the communication complexity of the recursive protocol for verifying m Boolean gates is roughly $4 \log m$ field elements, for a field $\mathbb{F}_{2^w}$ where $w \approx 42 + \log \log m$. For example, for a Boolean circuit with a million (nonlinear) gates, the verification protocol requires less than 0.5KB of communication *for the entire circuit.* The computational complexity of this protocol is dominated by roughly 30 field multiplications per gate. See Table 5 vs. Table 4 for comparing the concrete communication costs of the recursive vs. non-recursive protocol for Boolean circuits.

1.2 Comparison to Previous Work

The problem of 3PC with one malicious party has drawn a lot of attention in recent years. As the 3PC setting gives rise to the simplest and most efficient protocols, this setting was considered particularly attractive for applications such as financial data analysis [BLW08, BNTW12], protecting cryptographic keys [AFL+16], privacy-preserving machine learning [MR18], and more.

In this section, we compare the communication complexity and security of our protocol to the best concretely efficient 3PC protocols and to protocols for four parties with a single corruption. We restrict our attention to protocols that have been optimized for concrete efficiency and implemented, thus excluding the recent work of Boneh et al. [BBC+19] on which we build. In Table 1 we present a comparison between our results to other 3PC protocols for computing circuits that are defined over different domains: the Boolean domain $\mathbb{F}_2$, the field $\mathbb{F}_{2^8}$, which is commonly used in secure evaluations of AES, large finite fields, which are used emulate numerical computations over the integers or reals, and the ring $\mathbb{Z}_{2^k}$, which is used for integer arithmetic in CPUs with k-bit architectures, e.g. $k = 32$ or $k = 64$.

- The most efficient protocol for Boolean circuits, measured by total communication complexity, is [ABF+17], which achieves its low communication (7 bits per AND gate per party) by an efficient instantiation of the "cut–and–choose" method. The recent work of [CCPS19] has slightly higher overall communication but focuses on minimizing the online (input-dependent) cost. Indeed, the online communication cost of [CCPS19] per party is $\frac{4}{3}$ bits per AND gate.

[3] Alternatively, one can eliminate this extra interaction by using the Fiat-Shamir heuristic. However, for multi-round protocols such as ours, there are still gaps in our understanding of the soundness of this heuristic when analyzed in the random oracle model [PS00, BCS16].

- The protocol of [CGH$^+$18] (and also [NV18], but with higher computational cost) is the most efficient for "large" finite fields $\mathbb{F}$, i.e. fields such that $\mathbb{F}| \geq 2^{\sigma}$ for a statistical security parameter σ. For this field size, the protocol of [CGH$^+$18] requires each party to send 2 field elements per multiplication (i.e., doubling the communication cost of the semi-honest protocol). However, for small fields (e.g, $\mathbb{F}_2$ or $\mathbb{F}_{2^8}$) or non-field rings, their protocol requires many repetitions, making it inefficient.

- Recently, there has been considerable interest in MPC over the ring $\mathbb{Z}_{2^k}$ (see [CDE$^+$18, CRFG19, DOS18, GRW18, EOP$^+$19, CCPS19] and references therein). In the setting of three parties with honest majority, Eerikson et al. [EOP$^+$19] present two protocols, one that mixes operations on fields and on rings and one that relies on ring operations only. The latter protocol uses ideas from [CDE$^+$18], lifting each element to a larger ring $\mathbb{Z}_{2^{k+s}}$, and resulting in statistical error 2^{-s}. Since each party is required to send $3(k+s)$ bits per multiplication gate, then when $k = 64$, the number of ring elements sent is roughly 5. However, when the ring is small (in particular, when $k = 1$), the value of s is much larger than the bit-length of ring elements, since it determines the statistical error. This significantly increases the number of ring elements sent per gate.

To compare the security of our protocol to other protocols in Table 1, we remark that all of these protocols (including ours) use mild cryptographic assumptions, such as the existence of one-way functions or collision-resistant hash functions, to achieve their best efficiency. In fact, our protocol only makes a *black-box* use of any pseudorandom function (implemented in practice by a block cipher).

Our protocol is the only one in the table that provides full security rather than security with abort, which is a major qualitative advantage. It is well known that full security is achievable in our 3PC setting, assuming the availability of broadcast channels[4] or a PKI [RB89]. However, to the best of our knowledge, no other concretely efficient protocol in our setting achieves this, excluding protocols based on garbled circuits whose throughput is much worse than ours (see detailed comparison below).

It is also instructive to compare our results to what was achieved in the setting of two-thirds honest majority, i.e. the number of corrupt parties is less than a third of the total number of parties. Very recently, [FL19] presented a protocol (an improvement of [BHKL18]) which works over fields and requires each party to send $2\frac{2}{3}$ field elements per multiplication gate. The communication complexity of this protocol is higher than ours and the protocol only works over fields. However, the construction is applicable for any number of parties. Another low-communication protocol was proposed by [GRW18]. Their protocol is designed for the specific setting of 4 parties and one corruption and has communication cost of 1.5 ring elements per multiplication gate per party. While [FL19] guarantees security with abort only, in [GRW18] it is also shown how to achieve guaranteed output delivery but without concrete cost analysis. We stress that both of the above protocols do not apply to the case of 3PC; a minimal number of 4 parties is required to begin with.

Finally, we remark that for the case of evaluating Boolean circuits using the garbled-circuit approach [Yao86], and in the same setting of honest-majority 3PC, [IKKP15, MRZ15] have shown that it is possible to achieve malicious security with abort with similar cost of semi-honest. This was later extended to full security in [BJPR18]. In general, protocols using the garbled-circuit approach have the advantage of being constant-round, but their throughput is worse than ours by

[4] As our protocol makes a minimal use of broadcast, any reasonable implementation of broadcast over PKI suffices for making the cost of broadcast dominated by other costs.

The protocol	# of elements sent per party per multiplication gate				Full security?
	Boolean Circuits	Circuits over $\mathbb{F}_{2^8}$	Circuits over the ring $\mathbb{Z}_{2^{64}}$	Circuits over large finite fields ($\lvert\mathbb{F}\rvert \geq 2^{40}$)	
Araki et al. [ABF+17]	7	7	7	7	No
Chaudhari et al. [CCPS19]	7(offline)+4/3(online)	-	7(offline)+4/3(online)	-	No
Chida et al. [CGH+18]	41	6	41	2	No
Eerikson et al. [EOP+19]	123	-	5	-	No
This work	1	1	1	1	Yes

Table 1: Comparison between concretely efficient 3PC protocols. Statistical security for all protocols in the table is 40-bit. The communication per gate, measured by number of elements sent by each party, is amortized over a large circuit.

roughly two orders of magnitude. Moreover, protocols based on garbled circuits do not natively extend to evaluating arithmetic circuits, which are useful for many applications.

2 Preliminaries

Notation. Let P_1, P_2, P_3 denote the three parties participating in the computation. In this work, we assume an honest majority, and thus one party may be corrupted. We denote by R a finite ring, by $\mathbb{F}$ a finite field and by $\mathbb{Z}_{2^k}$ the ring of integers modulo 2^k. Finally, $[n]$ is used to denote the set $\{1, \ldots, n\}$.

2.1 Replicated Secret Sharing

In this section, we present the "replicated secret sharing" scheme used in our protocol and its operations. Originally used for realizing general access structures [ISN89], replicated secret sharing turned out to be useful for simple and concretely efficient MPC protocols with a small number of parties [Mau06, CDI05, BLW08, AFL+16]. The description in this section is specialized to the 3PC case and is based on [AFL+16, FLNW17].

To share an element v in a ring R, the dealer chooses three random elements $v_1, v_2, v_3 \in R$ under the constraint that $v_1 + v_2 + v_3 = v$. (This can be done by picking v_1, v_2 at random and computing $v_3 = v - (v_1 + v_2)$.) Then, the dealer shares v so that P_1's share is (v_1, v_3), P_2's share is (v_2, v_1) and P_3's share is (v_3, v_2). We use $[\![v]\!]$ to denote a sharing of v and denote P_i's share by (v_i, v_{i-1}).

The $\mathsf{reconstruct}([\![v]\!], i)$ procedure. In this procedure, party P_{i+1} and party P_{i-1} send their shares to P_i. Then P_i checks that the shares are consistent by checking that the same v_{i+1} was sent by both parties. If this holds, then P_i computes $v_1 + v_2 + v_3 = v$. Otherwise, P_i aborts.

The $\mathsf{open}([\![v]\!])$ procedure. The parties run $\mathsf{reconstruct}([\![v]\!], i)$ for each i.

Local operators. Given two shared secrets $[\![u]\!]$ and $[\![v]\!]$, the parties can compute locally the sharing of $u+v$, i.e., $[\![u+v]\!]$. Whenever we write that the parties compute $[\![u]\!]+[\![v]\!]$ (likewise for subtraction), it means that each party P_i sets its share of $u+v$ to be $(u_i+v_i, u_{i-1}+v_{i-1})$ where (u_i, u_{i-1}) is its share of u and (v_i, v_{i-1}) is its share of v.

In addition, given a sharing $[\![v]\!]$ and an element $\sigma \in R$, we write that the parties compute $\sigma \cdot [\![v]\!]$ when each party P_i sets its share of $\sigma \cdot v$ to be $(\sigma \cdot v_i, \sigma \cdot v_{i-1})$, where (v_i, v_{i-1}) is its share of v.

Finally, given a sharing $[\![v]\!]$ and an element $\sigma \in R$, the parties can compute locally $[\![v+\sigma]\!]$ by having party P_1 set its new share to be $(v_1+\sigma, v_3)$, party P_2 sets its share to be $(v_2, v_1+\sigma)$ and the share of P_3 remains the same.

2.1.1 The Semi-Honest Multiplication Protocol

We now show the protocol (as in [AFL+16]) for computing a sharing of $v \cdot u$, given $[\![v]\!]$ and $[\![u]\!]$. Let (v_i, v_{i-1}) and (u_i, u_{i-1}) be the shares of v and u respectively held by P_i. We assume that the parties P_1, P_2, P_3 hold *correlated randomness* $\alpha_1, \alpha_2, \alpha_3$, respectively, picked uniformly at random subject to $\alpha_1+\alpha_2+\alpha_3=0$. The parties compute $\binom{3}{2}$-shares (namely, 2-out-of-3 shares) of $v_1 v_2$ as follows:

1. **Step 1 – compute $\binom{3}{3}$-sharing:** Each party P_i computes $z_i = u_i \cdot v_i + u_i \cdot v_{i-1} + u_{i-1} \cdot v_i + \alpha_i$ and sends it to P_{i+1}. These messages are computed and sent in parallel.
2. **Step 2 – compute $\binom{3}{2}$-sharing:** Holding z_{i-1} and z_i, each party P_i stores (z_i, z_{i-1}).

Generating correlated randomness. It is possible to securely generate correlated randomness with perfect security by having each party P_i simply choose a random $\rho_i \in R$ and send it to P_{i+1} (where P_3 sends to P_1). Then, each party takes its random element to be its element subtracted by the element it received: P_1 computes $\alpha_1 = \rho_1 - \rho_3$, P_2 computes $\alpha_2 = \rho_2 - \rho_1$ and P_3 computes $\alpha_3 = \rho_3 - \rho_2$. Observe that $\alpha_1+\alpha_2+\alpha_3=0$ as required. In addition, if P_1 is corrupted, then it knows nothing about α_2 and α_3 except that $\alpha_1 = -\alpha_2 - \alpha_3$.

As shown in [AFL+16] (following [GI99, CDI05]), there is a simple procedure for generating this type of correlated randomness with *computational* security and without any interaction beyond a short initial setup. Let κ be a computational security parameter and let $\mathcal{F} = \{F_k \mid k \in \{0,1\}^\kappa, F_k : \{0,1\}^\kappa \to R\}$ be a family of pseudo-random functions [GGM86]. To generate random elements $\alpha_1, \alpha_2, \alpha_3 \in R$ under the constraint that $\alpha_1+\alpha_2+\alpha_3=0$, the parties work as follows.

1. **Setup step:** each party P_i chooses a random key $k_i \in \{0,1\}^\kappa$ and sends it to P_{i+1}.
2. **Computing the correlated randomness:** Upon request, each party computes $\alpha_i = F_{k_{i-1}}(id) - F_{k_i}(id)$ using the two keys it holds (where id is some public counter that was agreed upon and is incremented each time).

This method allows the parties to generate all the correlated randomness needed at the cost of one exchange of keys.

2.2 Security Definition

We use the standard definition of security based on the ideal/real model paradigm [Can00, Gol04]. When we say that a protocol securely computes an ideal functionality with abort, then we consider non-unanimous abort (sometimes referred to as "selective abort"). This means that the adversary first receives the output, and then determines for each honest party whether they will receive abort or receive their correct output.

2.3 Ideal Functionalities

In this section, we define the building blocks used by our protocol, formalized by ideal functionalities. For simplicity, we only consider the case where exactly one party is malicious. The correctness of the protocol in the case where no parties are corrupted is easy to verify directly.

2.3.1 $\mathcal{F}_{\text{rand}}$ - Generating Random shares

We define the ideal functionality $\mathcal{F}_{\text{rand}}$ to generate a sharing of a random value unknown to the parties. A formal description appears in Functionality 2.1. The functionality lets the adversary choose the corrupted parties' shares, which together with the random secret chosen by the functionality, are used to compute the shares of the honest parties.

FUNCTIONALITY 2.1 ($\mathcal{F}_{\text{rand}}$- Generating Random Shares)

Upon receiving r_i, r_{i-1} from the ideal adversary $\mathcal{S}$ controlling P_i, the ideal functionality $\mathcal{F}_{\text{rand}}$ chooses a random $r \in R$, sets $r_{i+1} = r - r_i - r_{i+1}$ and hands each honest party P_j its share (r_j, r_{j-1}).

To generate a sharing of a random value, the parties work similarly to generating correlated randomness:

1. **Setup step:** each party P_i chooses an random key $k_i \in \{0,1\}^\kappa$ and sends it to P_{i+1}.
2. **Computing the correlated randomness:** Upon request, each party computes $\alpha_i = F_{k_i}(id)$ and $\alpha_{i-1} = F_{k_{i-1}}(id)$ (where id is some public counter that was agreed upon and is incremented each time). Then, it sets its share to be (α_i, α_{i-1}).

2.3.2 $\mathcal{F}_{\text{coin}}$ - Generating Random Coins

$\mathcal{F}_{\text{coin}}$ is an ideal functionality that chooses a random element from R and hands it to all parties. A simple way to compute $\mathcal{F}_{\text{coin}}$ is to use $\mathcal{F}_{\text{rand}}$ to generate a random sharing and then open it.

2.3.3 $\mathcal{F}_{\text{input}}$ – Secure Sharing of Inputs

In this section, we present our protocol for secure sharing of the parties' inputs. The protocol works exactly as in [CGH$^+$18]: for each input x belonging to a party P_j, the parties call $\mathcal{F}_{\text{rand}}$ to generate a random sharing $[\![r]\!]$; denote the share held by P_i by r_i. Then, r is reconstructed to P_j, who echo/broadcasts $x - r$ to all parties. Finally, each P_i outputs the share $[\![r + (x - r)]\!] = [\![x]\!]$. This is secure since $\mathcal{F}_{\text{rand}}$ guarantees that the sharing of r is correct, which in turn guarantees that the sharing of x is correct (since adding $x - r$ is a local operation only). In order to ensure that P_j sends the same value $x - r$ to all parties, a basic echo-broadcast is used. This is efficient since all inputs can be shared in parallel, utilizing a single echo broadcast. The formal definition of the ideal functionality for input sharing appears in Functionality 2.2.

FUNCTIONALITY 2.2 ($\mathcal{F}_{\text{input}}$- Sharing of Inputs)

Let $\mathcal{S}$ be the ideal world adversary and let P_i be the corrupted party controlled by $\mathcal{S}$.

1. Functionality $\mathcal{F}_{\text{input}}$ receives inputs $v_1, \ldots, v_s \in R$ from the parties. For every $j \in [s]$, $\mathcal{F}_{\text{input}}$ also receives from $\mathcal{S}$ the share of the corrupted party $(v_{j,i}, v_{j,i-1})$ for the jth input.
2. For every $i \in [M]$, $\mathcal{F}_{\text{input}}$ computes $v_{j,i+1} = v_j - v_{j,i} - v_{j,i-1}$.
3. $\mathcal{F}_{\text{input}}$ hands P_{i-1} the shares $\{(v_{j,i-1}, v_{j,i+1})\}_{j=1}^{s}$ and P_{i+1} the shares $\{(v_{j,i+1}, v_{j,i})\}_{j=1}^{s}$.

A formal description of the protocol appears in Protocol 2.3

PROTOCOL 2.3 (Secure Sharing of Inputs)

- **Inputs:** Let $v_1, \ldots, v_s \in \mathbb{Z}_q$ be the series of inputs; each v_k is held by some P_j.
- **The protocol:**
  1. The parties call $\mathcal{F}_{\text{rand}}$ s times to obtain sharings $[\![r_1]\!], \ldots, [\![r_s]\!]$.
  2. For $k \in [s]$, the parties run $\mathsf{reconstruct}([\![r_i]\!], j)$ for P_j to receive r_k, where P_j is the owner of the kth input. If P_k receives $\perp$, then it sends $\perp$ to all parties, outputs abort and halts.
  3. For $k \in [s]$, party P_j sends $w_k = v_k - r_k$ to all parties.
  4. All parties send $\vec{w} = (w_1, \ldots, w_s)$, or a collision-resistant hash of the vector, to all other parties. If any party receives a different vector to its own, then it outputs $\perp$ and halts.
  5. For each $k \in [s]$, the parties compute $[\![v_k]\!] = [\![r_k]\!] + w_k$.
- **Outputs:** The parties output $[\![v_1]\!], \ldots, [\![v_s]\!]$.

For completeness, we now prove that Protocol 2.3 securely computes $\mathcal{F}_{\text{input}}$ specified in Functionality 2.2.

Proposition 2.4 *Protocol 2.3 securely computes Functionality 2.2 with abort in the presence of one malicious party.*

Proof: Let $\mathcal{A}$ be the real adversary. We construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ receives the shares $(r_{k,i}, r_{k,i-1})$ (for $k = 1, \ldots, s$) that $\mathcal{A}$ sends to $\mathcal{F}_{\text{rand}}$ in the protocol.
2. For every $k \in [s]$, $\mathcal{S}$ follows the instructions of $\mathcal{F}_{\text{rand}}$ to define a sharing $[\![r_k]\!]$.
3. $\mathcal{S}$ simulates the honest parties in all $\mathsf{reconstruct}$ executions. If an honest party P_j receives $\perp$ in the reconstruction, then $\mathcal{S}$ simulates it sending $\perp$ to all parties. Then, $\mathcal{S}$ simulates all honest parties aborting.
4. $\mathcal{S}$ simulates the remainder of the execution, obtaining all w_k values from $\mathcal{A}$ associated with corrupted parties' inputs, and sending random w_k values for inputs associated with honest parties' inputs.
5. For every $k \in [s]$ for which the kth input is that of the corrupted party P_i, simulator $\mathcal{S}$ sends the trusted party computing $\mathcal{F}_{\text{input}}$ the input value $v_k = w_k + r_k$ and the corrupted party's share of v_k.

6. $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

The only difference between the simulation by $\mathcal{S}$ and a real execution is that $\mathcal{S}$ sends random values w_j for inputs associated with honest parties' inputs. However, by the perfect secrecy of secret sharing, this is distributed identically to a real execution. ■

2.3.4 $\mathcal{F}_{\mathsf{vrfy}}$- Verifying Correctness of Messages in the Multiplication Protocol

We now define an ideal functionality to verify that the messages sent in the execution of the multiplication protocol are the correct messages according to the protocol's instructions, given each party's inputs. Recall that in our multiplication protocol, each party P_i sends one message z_i to P_{i+1}. Given P_i's input shares (u_i, u_{i-1}) and (v_i, v_{i-1}) and its randomness α_i, let

$$c(u_i, u_{i-1}, v_i, v_{i-1}, \alpha_i, z_i) = u_i \cdot v_i + u_i \cdot v_{i-1} + u_{i-1} \cdot v_i + \alpha_i - z_i.$$

We need to ensure that

$$c(u_i, u_{i-1}, v_i, v_{i-1}, \alpha_i, z_i) = 0. \tag{1}$$

Observe that the input to c is distributed among P_{i-1} and P_{i+1}. Specifically, u_i, v_i, z_i are known to P_{i+1}, u_{i-1}, v_{i-1} are known to P_{i-1} and ρ_i, α_{i-1}, which are held by P_{i+1} and P_{i-1} respectively, determine deterministically the value of α_i, since $\alpha_i = \rho_i - \rho_{i-1}$ (see Section 2.1.1). We use this fact to define the ideal functionality $\mathcal{F}_{\mathsf{vrfy}}$, which receives the shares on the input wires, the randomness and the sent/received message from the honest parties, and checks that Eq. (1) holds. This is formalized in Functionality 2.5.

FUNCTIONALITY 2.5 ($\mathcal{F}_{\mathsf{vrfy}}$- Verify Correctness of Messages)

Let $\mathcal{S}$ be the ideal world adversary and let P_i be the corrupted party controlled by $\mathcal{S}$. $\mathcal{F}_{\mathsf{vrfy}}$ receives an index j and a parameter $m \in \mathbb{Z}$ from the honest parties. Then:

- If P_j is an honest party, then $\mathcal{F}_{\mathsf{vrfy}}$ receives from P_j its input $(u_{k,j}, u_{k,j-1}, v_{k,j}, v_{k,j-1}, \alpha_{k,j}, z_{k,j})$ for each $k \in [m]$.
  If $i = j + 1$, then $\mathcal{F}_{\mathsf{vrfy}}$ receives $\rho_{k,j-1}$ for each $k \in [m]$ from the honest P_{j-1} and hands $\mathcal{S}$ the input of P_i, i.e., $(u_{k,j}, v_{k,j}, \alpha_{k,j} + \rho_{k,j+1}, z_{k,j})$ for each $k \in [m]$.
  If $i = j - 1$, then $\mathcal{F}_{\mathsf{vrfy}}$ receives $\rho_{k,j}$ for each $k \in [m]$ from the honest P_{j+1} and hands $\mathcal{S}$ the input of P_i, i.e., $(u_{k,j-1}, v_{k,j-1}, \rho_{k,j} - \alpha_{k,j})$ for each $k \in [m]$.
  Then, $\mathcal{S}$ sends $\mathcal{F}_{\mathsf{vrfy}}$ the command abort or accept, which is handed by $\mathcal{F}_{\mathsf{vrfy}}$ to the honest parties.
- If P_j is the corrupted party (i.e., $i = j$) then $\mathcal{F}_{\mathsf{vrfy}}$ receives $(u_{k,i}, v_{k,i}, z_{k,i}, \rho_{k,i})$ from P_{i+1} and $(u_{k,i-1}, v_{k,i-1}, \rho_{k,i-1})$ from P_{i-1} for each $k \in [m]$.
  Then, $\mathcal{F}_{\mathsf{vrfy}}$ checks for each $k \in [m]$ that

  $$c(u_{k,i}, u_{k,i-1}, v_{k,i}, v_{k,i-1}, \alpha_{k,i}, z_{k,i}) = 0$$

  where $\alpha_{k,i} = \rho_{k,i} - \rho_{k,i-1}$.
  In addition, $\mathcal{F}_{\mathsf{vrfy}}$ hands $u_{k,i}, u_{k,i-1}, v_{k,i}, v_{k,i-1}, \rho_{k,i}, \rho_{k,i-1}, z_{k,i}$ to $\mathcal{S}$.
  If the equality does not hold for each $k \in [m]$, then $\mathcal{F}_{\mathsf{vrfy}}$ sends abort to the parties. Otherwise, upon receiving a command abort or accept from $\mathcal{S}$, $\mathcal{F}_{\mathsf{vrfy}}$ hands it to the honest parties and halts.

Observe that the functionality checks the correctness of many multiplications at the same time. Concretely efficient protocols for this functionality, with sublinear communication in the number of verified multiplications, will be presented in Sections 4 and 7.

3 The Main Protocol

In this section we present our framework to compute an arithmetic circuit over a finite ring R. The protocol utilizes the ideal functionalities $(\mathcal{F}_{\text{input}}, \mathcal{F}_{\mathsf{vrfy}})$ defined in the previous section and is described in Protocol 3.2. (We will later instantiate $\mathcal{F}_{\mathsf{vrfy}}$ for rings R that are either a finite field $\mathbb{F}$ or integer rings of the form $\mathbb{Z}_{2^k}$; however, the general framework applies over any finite ring R.) The protocol works naturally by having the parties run the semi-honest protocol and then, before reconstructing the outputs, calling the $\mathcal{F}_{\mathsf{vrfy}}$ functionality to detect any cheating during the execution.

We state the security features of our protocol in the following two theorems.

Theorem 3.1 (Information theoretic-security) *Let R be a finite ring and let f be a 3-party functionality computed by an arithmetic circuit C over R. Suppose that the correlated randomness for the multiplication protocol (as specified in Section 2.1.1) is generated perfectly. Then, Protocol 3.2 computes f with* perfect security with abort *in the $(\mathcal{F}_{\text{input}}, \mathcal{F}_{\mathsf{vrfy}})$-hybrid model in the presence of one malicious party.*

Proof: Let P_i be the corrupted party and let $\mathcal{S}$ be the ideal world adversary. The simulation works as follows:

1. By playing the role of $\mathcal{F}_{\text{input}}$ in the first step, $\mathcal{S}$ extracts the inputs of P_i and receives the shares of P_i on each input wire.

2. $\mathcal{S}$ simulates the circuit emulating step. For linear gates, it computes locally party P_i's share on the output wire. For multiplication gates, $\mathcal{S}$ runs the semi-honest multiplication protocol with the corrupted P_i in the following way: first, it chooses a random message $z_{i-1} \in R$ and plays the role of P_{i-1} sending z_{i-1} to P_i. Then, it receives the message z_i sent by P_i to party P_{i+1}. Note that $\mathcal{S}$ knows the shares held by P_i on each input wire and knows the randomness that P_i should use in the execution (since its randomness α_i is computed by taking $\rho_i - \rho_{i-1}$ where ρ_i is sent to P_{i+1} and ρ_{i-1} is chosen by P_{i-1} as described in Section 2.1.1). Thus, $\mathcal{S}$ knows whether z_i is the correct message or not and so $\mathcal{S}$ records whether cheating took place or not. Finally, $\mathcal{S}$ defines (z_i, z_{i-1}) to be the P_i's share on the output wire.

3. $\mathcal{S}$ simulates the verification step by following the instructions of $\mathcal{F}_{\mathsf{vrfy}}$. Specifically, for proving the correctness of message sent by honest parties, $\mathcal{S}$ outputs $\mathsf{abort}/\mathsf{accept}$ according to P_i's instructions. For proving correctness of messages sent by P_i, if $\mathcal{S}$ recorded that cheating took place in the previous step, then it plays the role of $\mathcal{F}_{\mathsf{vrfy}}$ outputing abort to the parties. Otherwise, it outputs $\mathsf{abort}/\mathsf{accept}$ according to P_i's instructions.
   In any case that $\mathcal{F}_{\mathsf{vrfy}}$ outputted abort, $\mathcal{S}$ simulates the honest parties aborting in the real world execution, outputs whatever P_i ouptuts and halts.

4. $\mathcal{S}$ sends P_i's inputs to the trusted party computing f to receive back the output that P_i should obtain.

PROTOCOL 3.2 (Computing an Arithmetic Circuit Over Finite Fields/Rings)

- **Inputs:** Each party P_j ($j \in \{1, 2, 3\}$) holds an input $x_j \in R^{\ell}$.
- **Auxiliary Input:** The parties hold a description of an arithmetic circuit C that computes f on inputs of length $\ell \cdot 3$. Let m be the number of multiplication gates in C.
- **Setup:** If the correlated randomness for the multiplication protocol is generated using a pseudorandom function F_k, then the parties run the setup step as specified in Section 2.1.1.
- **The protocol:**

  (a) *Sharing the inputs:* For each input x_k held by Party P_j, party P_j sends x_k to $\mathcal{F}_{\text{input}}$. Each party records its vector of shares of all inputs as received from $\mathcal{F}_{\text{input}}$. If any party received $\perp$ from $\mathcal{F}_{\text{input}}$ then it sends abort to the other parties and halts.

  (b) *Circuit emulation:* Let $G_1, ..., G_L$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, ..., L$ the parties work as follows:

  - *If G_k is an addition gate:* Given shares $\llbracket u \rrbracket$ and $\llbracket v \rrbracket$ on the input wires, the parties locally compute $\llbracket u + v \rrbracket$.
  - *If G_k is a multiplication-by-a-constant gate:* Given share $\llbracket u \rrbracket$ on the input wire and a public constant $\sigma \in R$, the parties locally compute $\llbracket \sigma \cdot u \rrbracket$.
  - *If G_k is a multiplication gate:* Given shares $\llbracket u \rrbracket$ and $\llbracket v \rrbracket$ on the input wires, the parties run the semi-honest protocol from Section 2.1.1 on $\llbracket u \rrbracket$ and $\llbracket v \rrbracket$, and define the result as their share on the output wire.

  (c) *Verification stage: Before* the secrets on the output wires are reconstructed, the parties verify that all the multiplications were carried out correctly, as follows.
  For each $j \in \{1, 2, 3\}$, the parties send j, m, the shares of P_j on the input wires, the messages sent by P_j and its randomness for each multiplication gate to $\mathcal{F}_{\mathsf{vrfy}}$.
  If a party received abort from $\mathcal{F}_{\mathsf{vrfy}}$, then it sends $\perp$ to the other parties and outputs $\perp$.

  (d) If any party received $\perp$ in any of the previous steps, then it outputs $\perp$ and halts.

  (e) *Output reconstruction:* For each output wire of the circuit, the parties run $\mathsf{reconstruct}(\llbracket v \rrbracket, j)$, where $\llbracket v \rrbracket$ is the sharing of the value on the output wire, and P_j is the party whose output is on the wire.

  (f) If a party received $\perp$ in any call to the reconstruct procedure, then it sends $\perp$ to the other parties, outputs $\perp$ and halts.

- **Output:** If a party did not output $\perp$, then it outputs the values it received on its output wires.

5. $\mathcal{S}$ simulates the reconstruction of outputs: recall that $\mathcal{S}$ knows P_i's shares on each output wire and that reconstruction of secretes towards party P_j involves each party sending their shares to P_j. Thus, for outputs that should be revealed to an honest party P_j, $\mathcal{S}$ receives the shares sent by P_i in the reconstruction procedure, and if they don't match the shares held by P_i, then it sends abort_j to the trusted party.
   For outputs that should be obtained by P_i, let v be such an output (as received from the trusted party in the previous step) and let (v_i, v_{i-1}) be the shares of v held by P_i. Then, $\mathcal{S}$ computes $v_{i+1} = v - v_i - v_{i-1}$ and send it to P_i.

Observe that until and not including the output reconstruction step, the only difference between the simulation and real execution is in P_i's view in the multiplication protocol. Specifically, in the simulation the message z_{i-1} sent to P_i is chosen uniformly over R, whereas in the real execution it is computed using P_{i-1}'s shares of the real values and its correlated randomness α_{i-1}. However, since α_{i-1} which is used as a masking is uniformly distributed over R, then so is z_{i-1}, exactly as in the simulation. It remains to show that the view of P_i in the output reconstruction step is distributed identically in both executions. However, this follows directly from the perfect secrecy of the secret sharing scheme. This complete the proof. ■

Theorem 3.3 (Computational security) *Let R be a finite ring and let f be a 3-party functionality computed by an arithmetic circuit C over R. Suppose that the correlated randomness for the multiplication protocol is generated using a pseudorandom function F_k as specified in Section 2.1.1. Then, Protocol 3.2 computes f with* computational security with abort *in the $(\mathcal{F}_{\text{input}}, \mathcal{F}_{\mathsf{vrfy}})$-hybrid model in the presence of one malicious party.*

Proof: The proof is identical to the the proof of Theorem 3.1 with the only exception being that the simulator runs first the initialization step for the correlated randomness generation protocol. Then, $\mathcal{S}$ knows the keys used by P_i to generate its randomness α_i in each multiplication gate and thus can compute it by himself and proceed exactly as in the proof of Theorem 3.1. The only difference between the simulation and the real world execution is the way the message z_{i-1} in each multiplication gate is computed. Specifically, in the simulation it is uniformly distributed over R, whereas in the real execution z_{i-1} is masked using α_{i-1} which is computed by taking $F_{k_{i-1}}(id) - F_{k_{i+1}}(id)$ where k_{i+1} is unknown to P_i. However, if the corrupted P_i can distinguish between its view in the two execution, then by a straight forward reduction we can show that it is possible to distinguish between a pseudo-random function and a truly random function. This completes the proof. ■

Remark 3.4 (On the choice of the protocol from [AFL+16]) *Protocol 3.2 takes advantage of the simplicity of the semi-honest 3PC protocol of Araki et al. [AFL+16], where the roles of the 3 parties are totally symmetric. This gives rise to an easy implementation that minimizes the overall communication and computation costs. However, it is possible to apply a similar methodology to other semi-honest 3PC protocols from the literature by adapting the implementation of $\mathcal{F}_{\mathsf{vrfy}}$ to their message structure. For instance, using the semi-honest 3PC protocol of Katz et al. [KKW18] as the semi-honest baseline, one would get a protocol that has similar overall costs, is slightly more complex to describe and implement, but has the advantage of better performance in an offline-online setting (pushing roughly 1/3 of the communication and computation to an input-independent offline phase).*

4 Instantiating $\mathcal{F}_{\mathsf{vrfy}}$ - Proving Honest Behavior

In this section, we show how to securely compute the $\mathcal{F}_{\mathsf{vrfy}}$ ideal functionality, building on the approach of Boneh et al. [BBC+19].

4.1 A Protocol for Any Finite Field $\mathbb{F}$

Recall that the goal of each party P_i is to prove for each multiplication gate that Eq. (1) holds. In our solution, we construct a "verification" circuit that contains m parallel copies of the small circuit c (recall that m is the number of multiplication gates). A naive construction of such a circuit will take $6m$ inputs $x_1, \ldots, x_{6m}$ (recall that each c circuit receives 6 inputs) and output the concatenation of the outputs of the small c circuits.

Our construction of the circuit is however slightly different. We first define a sub-circuit g that contains L small c circuits and so it takes $6L$ inputs and outputs the *random linear combination* of its c circuits' outputs. Thus,

$$g(x_1, \ldots, x_{6L}) = \sum_{k=1}^{L} \theta_k \cdot c(x_{(k-1)6+1}, \ldots, x_{(k-1)6+6}).$$

Letting $M = m/L$, we define the verification circuit G which outputs a random linear combination of the g circuits outputs. That is,

$$G(x_1, \ldots, x_{6m}) = \sum_{k=1}^{M} \beta_k \cdot g(x_{(k-1)6L+1}, \ldots, x_{(k-1)6L+6L})$$

where θ_k and β_k are uniformly distributed over $\mathbb{F}$ and will be randomly chosen jointly by the parties as explained below. Observe that the multiplicative depth of G is *constant*, regardless of the depth of the circuit being evaluated by the parties. The goal of a prover P_i is to prove that the output of G over the shared input is 0.

Our protocol has three rounds. In the first round, the prover P_i defines $6L$ random polynomials with degree M by setting $M+1$ points of each polynomial f_j in the following way: $f_j(0)$ is chosen randomly whereas $f_j(\ell)$ for $\ell \in \{1, \ldots, M\}$ is assigned by the jth *input* to the ℓth g gate. Next, the prover P_i defines a polynomial $p(\cdot)$ which is defined by taking $g(f_1, \ldots, f_{6L})$. This means that $p(\ell)$ for $\ell \in \{1, \ldots, M\}$, is the *output* of the ℓth g gate. Note that since the degree of each f is M and the degree of the g circuit is 2, it follows that p is a polynomial of degree $2M$. The proof sent by P_i to the other 2 parties at the end of the first round is essentially an additive secret sharing of $f_1(0), \ldots, f_{6L}(0)$ and the coefficients of p. This clearly does not reveal any information to the other 2 parties.

In the second and third rounds, parties P_{i+1} and P_{i-1} use the proof sent by P_i to validate that the circuit G indeed outputs 0 over the input that is shared between them. To be convinced, the parties need to verify that two properties hold:

1. The output of the circuit is 0. This can be verified by computing $\sum_{\ell=1}^{M} \beta_\ell \cdot p(\ell)$ where β_ℓ is randomly chosen *after* the proof has been sent. If p is defined correctly, then this is indeed the sum of the outputs of all g gates which yields the output of the circuit G. Note that each party has an additive share of the coefficients of p, but since all the operations are linear, the two parties can compute the random linear combination using their shares of the polynomial p and then send the result to each other.

2. For the previous check to work, the parties need to verify that p was correctly defined. This is done by sampling a random point r, and checking that $p(r) = g(f_1(r), \ldots, f_{6L}(r))$, where the left side is computed using the coefficients of p. For privacy to hold here, the point r must be sampled from $\mathbb{F} \setminus \{1, \ldots, M\}$. Otherwise, the parties will learn inputs to some g gate. If the prover P_i cheated and p is not defined correctly, then $q(x) = p(x) - g(x)$ is not the zero polynomial and so the probability to choose a root is bounded by $\frac{\mathsf{degree}(q)}{|\mathbb{F}|-M} = \frac{2M}{|\mathbb{F}|-M}$. Note again that the two parties hold an additive sharing of the points of the f polynomials and so by applying local linear operations on their shares, they obtain an additive sharing of $p(r)$ and $f_1(r), \ldots, f_{6L}(r)$, which can be exchanged to complete the check.

To reduce communication even further, we observe that since only one party is corrupted it suffices for one verifier (party P_{i+1} in Protocol 4.1) to perform the final check in Round 3 and decide whether to accept or abort. This follows from the fact that if the prover is corrupted then both verifiers are honest whereas if the deciding verifier (i.e., P_{i+1}) is the corrupted party, then all it can do is cause the parties to abort (but not learn any sensitive information). All the above is formalized in Protocol 4.1.

Setting the parameters and cost analysis. Observe that the proof sent by P_i at the end of Round 1 consists of $6L+2M+1$ field elements transmitted to both verifiers. However, observe that instead of choosing $f_1(0),\ldots,f_{6L}(0)$ randomly and secret sharing them to the verifiers, it is possible to first choose the shares of theses values and use them to compute the values of $f_1(0),\ldots,f_{6L}(0)$. Then, to reduce communication, we can use the well-known trick of letting P_i choose these shares pseudo-randomly and send only a seed to the verifiers. Similarly, P_i can also derive the shares of the remaining $2M+1$ elements from a single seed, but here only for *one* of the verifies. Thus, overall, P_i sends $2M+1$ elements in the first step.

In the second round, one verifier sends $6L+2$ field elements. As this proof is executed three times (each time one of three parties acts as the prover and the other two as the verifiers), we conclude that overall each party sends in the verification step roughly $6L+2M+3$ field elements. By setting $M=L=\sqrt{m}$, this is translated to $8\sqrt{m}+3$ field elements per party. The amount of communication in this step is thus sub-linear in the size of the circuit, and so when amortized over its size, each party sends $\frac{8}{\sqrt{m}}$ field elements per multiplication gate. The overall communication per party for computing an arithmetic circuit of m multiplication gates is thus roughly 1 field element per multiplication.

Security proof. We prove that our protocol securely realizes the ideal functionality $\mathcal{F}_{\mathsf{vrfy}}$ in the Theorem 4.2. For computing the statistical error, we take into consideration that the prover can succeed in cheating, if the random coefficients cause the output of G to be 0 (which happens with probability $\frac{1}{|\mathbb{F}|}$) or, as explained above, if the point r is the root of the polynomial $q(x)$ (which happens with probability $\frac{2M}{|\mathbb{F}|-M}$). In the proof of Theorem 4.2, we show that the two events yield an overall cheating probability bounded by $\frac{2M+1}{|\mathbb{F}|-M}$, as stated in the Theorem.

PROTOCOL 4.1 (Securely Computing $\mathcal{F}_{\mathsf{vrfy}}$ for Finite Fields)

- **Inputs:** Prover P_i holds the vector $(x^i_1, \ldots, x^i_{6m})$ such that $\forall k \in [m]$:

$$\left(x^i_{6(k-1)+1}, x^i_{6(k-1)+2}, x^i_{6(k-1)+3}, x^i_{6(k-1)+4}, x^i_{6(k-1)+5}, x^i_{6(k-1)+6}\right) = (u_{i,k}, u_{i-1,k}, v_{i,k}, v_{i-1,k}, \alpha_{i,k}, z_{i,k})$$

  P_{i+1} holds the vector $(x^{i+1}_1, \ldots, x^{i+1}_{6m})$ such that $\forall k \in [m]$:

$$\left(x^{i+1}_{6(k-1)+1}, x^{i+1}_{6(k-1)+2}, x^{i+1}_{6(k-1)+3}, x^{i+1}_{6(k-1)+4}, x^{i+1}_{6(k-1)+5}, x^{i+1}_{6(k-1)+6}\right) = (u_{i,k}, 0, v_{i,k}, 0, \rho_{i,k}, z_{i,k})$$

  P_{i-1} holds the vector $(x^{i-1}_1, \ldots, x^{i-1}_{6m})$ such that $\forall k \in [m]$:

$$\left(x^{i-1}_{6(k-1)+1}, x^{i-1}_{6(k-1)+2}, x^{i-1}_{6(k-1)+3}, x^{i-1}_{6(k-1)+4}, x^{i-1}_{6(k-1)+5}, x^{i-1}_{6(k-1)+6}\right) = (0, u_{i-1,k}, 0, v_{i-1,k}, -\rho_{i-1,k}, 0)$$

  where $\alpha_{i,k} = \rho_{i,k} - \rho_{i-1,k}$.

- **Auxiliary Input:** All parties hold public parameters L and M.
- **The protocol:**
  1. **Round 1:**
     (a) The parties call $\mathcal{F}_{\text{coin}}$ to receive random $\theta_1, \ldots, \theta_L \in \mathbb{F}$.
     (b) P_i chooses random $w_1, \ldots, w_{6L} \in \mathbb{F}$.
     (c) P_i defines $6L$ polynomials $f_1, \ldots, f_{6L} \in \mathbb{F}[x]$ of degree M such that for each $j \in [6L]$:
     $$f_j(0) = w_j \qquad \textsf{and} \qquad \forall \ell \in [M] : f_j(\ell) = x^i_{6L(\ell-1)+j}$$
     (Note that $f_j(\ell)$ is the jth input to the ℓth g gate)
     (d) P_i computes the coefficients of the $2M$-degree polynomial $p(x) \in \mathbb{F}[x]$ defined by $p = g(f_1, \ldots, f_{6L})$ (where g is as described in the text). Let $a_0, \ldots, a_{2M} \in \mathbb{F}$ be the obtained coefficients.
     (e) P_i defines $\vec{\pi} = (w_1, \ldots, w_{6L}, a_0, \ldots, a_{2M})$. Then, it chooses random $\vec{\pi}^{i+1} \in \mathbb{F}^{6L+2M+1}$ and defines $\vec{\pi}^{i-1} = \vec{\pi} - \vec{\pi}^{i+1}$.
     (f) P_i sends $\vec{\pi}^{i+1}$ to P_{i+1} and π_{i-1} to P_{i-1}.
  2. **Round 2:**
     (a) The parties call $\mathcal{F}_{\text{coin}}$ to receive random $\beta_1, \ldots, \beta_M \in \mathbb{F}$ and $r \in \mathbb{F} \setminus \{0, \ldots, M\}$.
     (b) Each party P_t where $t \in \{i-1, i+1\}$ does the following:
        i. Parse the message $\vec{\pi}^t$ as $(w^t_1, \ldots, w^t_{6L}, a^t_0, \ldots, a^t_{2M})$.
        ii. Define $6L$ polynomials $f^t_1, \ldots, f^t_{6L} \in \mathbb{F}[x]$ of degree M such that for each $j \in [6L]$:
        $$f^t_j(0) = w^t_j \qquad \textsf{and} \qquad \forall \ell \in [M] : f^t_j(\ell) = x^t_{6L(\ell-1)+j}$$
        iii. Compute $f^t_j(r)$ for each $j \in [6L]$ and $p^t_r = \sum_{j=0}^{2M} a^t_j \cdot r^j$.
        iv. Compute $b^t = \sum_{j=1}^{M} \beta_j \cdot \left(\sum_{k=0}^{2M} a^t_k \cdot j^k\right)$.
     (c) P_{i-1} sends $f^{i-1}_1(r), \ldots, f^{i-1}_{6L}(r), p^{i-1}_r, b^{i-1}$ to P_{i+1}
  3. **Round 3:**
     (a) Upon receiving the message sent from P_{i-1} in Round 2, P_{i+1} computes:
     $$\forall j \in [6L] : f'_j(r) = f^{i+1}_j(r) + f^{i-1}_j(r), \qquad p_r = p^{i+1}_r + p^{i-1}_r \qquad \textsf{and} \qquad b = b^{i+1} + b^{i-1}$$
     (b) P_{i+1} checks that: (1) $p_r = g(f'_1(r), \ldots, f'_{6L}(r))$ and (2) $b = 0$.
     If the two equalities did not hold, then it outputs abort. Otherwise, it outputs accept.

Theorem 4.2 *Protocol 4.1 securely computes $\mathcal{F}_{\mathsf{vrfy}}$ with abort in the presence of one malicious party, with statistical error $\frac{2M+1}{|\mathbb{F}|-M}$.*

Proof: We construct an ideal world simulator $\mathcal{S}$ for two different cases.

Case 1: the prover P_i is corrupted. In this case, $\mathcal{S}$ receives the inputs of P_i from $\mathcal{F}_{\mathsf{vrfy}}$ and so $\mathcal{S}$ knows the inputs of the honest parties. Thus, $\mathcal{S}$ can simulate exactly the role of the honest parties in the protocol. $\mathcal{S}$ invokes the real world adversary to receive the proof sent by P_i to the two other parties. $\mathcal{S}$ simulates $\mathcal{F}_{\text{coin}}$ handing the point r and random coefficients to the parties and follows the instructions of two verifiers.

If P_i was acting honestly in the execution of the main protocol and the output of the circuit c is 0 for every multiplication gate, then $\mathcal{S}$ sends $\mathcal{F}_{\mathsf{vrfy}}$ the output of the verifiers. Note that in this case, the simulation is perfect.

In contrast, if the output of the circuit c is not 0 for some multiplication gate (meaning that the prover has cheated in the execution), then $\mathcal{F}_{\mathsf{vrfy}}$ outputs abort to the parties. Thus, if the honest parties simulated by $\mathcal{S}$ output accept, then $\mathcal{S}$ outputs fail and halts.

The only difference between the simulation and the real execution is the event that $\mathcal{S}$ output fail. Observe that this happens if and only if one the following events happen: (1) the random coefficients $\theta_1, \ldots, \theta_L$ were chosen such that the output of the g gates is 0 or, (2) $p(r) = g(f_1(r), \ldots, f_{6L}(r))$ but $p(x) \neq g(f_1(x), \ldots, f_{6L}(x))$, or (3) if the random linear combination using $\beta_1, \ldots, \beta_M$ yields that the output of the circuit G is 0. In the first and third events, the check will pass with probability of at most $\frac{1}{|\mathbb{F}|}$ (as each random coefficient is uniformly distributed over $\mathbb{F}$). In the second event, the polynomial $q(x) = p(x) - g(f_1(x), \ldots, f_{6L}(x))$ is not the zero polynomial. Recall that the degree of $p(x)$ is $2M$ and thus so is the degree of $q(x)$. Since r is chosen uniformly from $\mathbb{F} \setminus \{0, \ldots, M\}$ and there are at most $2M$ roots for $q(x)$, the probability that the check passes and the fail event occurs in this case is bounded by $\frac{2M}{|\mathbb{F}|-M}$.

Observe that the θ coefficients are known before the proof is sent, whereas the β coefficients are being chosen only after the proof is sent. Thus, if the first event occurs, then the prover knows that it succeeded in cheating before generating the proof, which means that it can act honestly from now on. On the other hand, when the first event does not occur, the prover needs to decide whether to cheat in the generation of the polynomial p or to hope that the third event will happen. Since $\frac{2M}{|\mathbb{F}|-M} \geq \frac{1}{|\mathbb{F}|}$, the prover will clearly prefer to cheat in the proof. Overall, we have that the success cheating probability is bounded by $\frac{1}{|\mathbb{F}|} + (1 - \frac{1}{|\mathbb{F}|})\frac{2M}{|\mathbb{F}|-M} < \frac{2M+1}{|\mathbb{F}|-M}$, which is exactly the statistical error of the protocol .

Case 2: the prover P_i is honest. In this case, the simulator $\mathcal{S}$ receives the inputs known to the corrupted verifier P_i but it needs to simulate the protocol execution without knowing the prover and the other verifier's inputs. Thus, $\mathcal{S}$ simulates $\mathcal{F}_{\text{coin}}$ handing $\theta_1, \ldots, \theta_L \in \mathbb{F}$ to the parties and then it chooses a random $\pi' \in \mathbb{F}^{6L+2M+1}$ and hands it to the real world adversary as the message sent by the prover P_i. Then, it simulate the ideal functionality $\mathcal{F}_{\text{coin}}$ handing a random point $r \in \mathbb{F} \setminus \{0, \ldots, M\}$ and coefficients $\beta_1, \ldots, \beta_M \in \mathbb{F}$ to the parties. Since $\mathcal{S}$ knows the corrupted party's inputs, it can compute now the message that should be sent by the corrupted party $f_1'(r), \ldots, f_{6L}'(r), p_r', b'$. If P_{i-1} is the corrupted party and so it only sends its message at the end of Round 2 to the honest P_{i+1} who decides whether to accept or not, then upon receiving the message, $\mathcal{S}$ can tell whether the honest P_{i+1} will abort or accept by comparing it to the message that should have been sent. We conclude that in this case the simulation is perfect.

Otherwise, P_{i+1} is the corrupted party and thus $\mathcal{S}$ needs to simulate the message sent by the honest verifier P_{i-1}. Thus, to compute the message sent by the P_{i-1}, it chooses random

$f''_1(r), \ldots, f''_{6L}(r) \in \mathbb{F}$, computes

$$g_r = g\left((f'_1(r) + f''_1(r)), \ldots, (f'_{6L}(r) + f''_{6L}(r))\right)$$

and set $p''_r = g_r - p'_r$. Finally, it sets $b'' = -b'$ and hands the message $f''_1(r), \ldots, f''_{6L}(r), p''_r, b''$ to adversary as the message sent by the honest verifier P_{i-1}. Then, it waits for the adversary controlling P_{i+1} to output accept or abort and hand its output to $\mathcal{F}_{\mathsf{vrfy}}$.

We claim that the view of the adversary in the simulation is distributed identically to its view in the real execution. First, observe that this follows immediately for the message sent by the prover at the end of Round 1 due to the perfect secrecy of additive secret sharing. Next, observe that $f_k(r)$ for each $k \in [6L]$ can be written using Lagrange coefficients $\lambda_0(r), \ldots, \lambda_M(r)$ in the following way:

$$f_k(r) = \lambda_0(r) \cdot f_k(0) + \sum_{\ell=1}^{M} \lambda_\ell(r) \cdot f_k(\ell).$$

Now, since $r \notin \{0, \ldots, M\}$ it follows that $\lambda_0(r) \neq 0$. Thus, using the fact that $f_k(0)$ is distributed uniformly and independently over $\mathbb{F}$, we obtain that $f_k(r)$ is also uniformly distributed over $\mathbb{F}$. Thus, the distribution of $f''_1(r), \ldots, f''_{6L}(r)$ is exactly as in the real execution. Since g_r is computed as in a real execution on $f'_1(r), \ldots, f'_{6L}(r)$ and $f''_1(r), \ldots, f''_{6L}(r)$, it is also distributed the same.

Finally, we claim that in both executions b'' is uniformly distributed over $\mathbb{F}$ under the constraint that $b' + b'' = 0$. This follows by definition for the simulation. In the real execution, recall that $b'' = \sum_{j=1}^{M} \beta_j \cdot \left(\sum_{k=0}^{2M} a''_k \cdot j^k\right)$ where all β_js are public random and all a''_ks are uniformly distributed over $\mathbb{F}$ as they are random shares of the the polynomial p's coefficients. Thus, the distribution is again the same. We conclude that in this case, the simulation is perfect. This concludes the proof. ■

We remark that when using the optimization described above to reduce communication, where the prover just sends random seeds from which pseudo-random shares are derived, we obtain *computational security*. The proof is identical to the proof of Theorem 4.2, with an additional step where it is shown by a reduction that if it is possible to distinguish between the transcript when true randomness is used and the transcript when pseudo-randomness is used with non-negligible probability, then this can be used to distinguish between random and pseudo-random functions with the same probability.

Remark 4.3 (Recomputable verification) *Note that Protocol 4.1 satisfies the following "recomputable verification" property. For prover P_i, each verifier party P_t, $t \in \{i-1, i+1\}$, sends a* single *message in the protocol, which is computed as a* deterministic *function $v_t = \mathsf{VMsg}_t(\pi_t, x_t, \mathsf{pub})$ of the messages π_t sent from P_i to P_t, the input x_t and values pub that are publicly known by all parties. More explicitly, the message of P_t (described in Round 2 (c)) depends deterministically on the prover's message $\vec{\pi}^t$, its inputs to the protocol $\vec{x}^t$, and the public random coins $\theta_1, \ldots, \theta_L, \beta_1, \ldots, \beta_M, r$ received from $\mathcal{F}_{\mathrm{coin}}$. Since the inputs of each verifier are known to the prover, this in particular means that the prover himself can recompute the expected message of each verifier. This property (and the notations $\mathsf{VMsg}_t, \pi_t, x_t, \mathsf{pub}$) will be leveraged later in Section 5, when obtaining* full *security.*

4.2 Extending the Protocol to the Ring $\mathbb{Z}_{2^k}$

In this section, we show how our protocol can work over the ring $\mathbb{Z}_{2^k}$. This is of importance since arithmetic in the hardware of modern computers is done modulo 2^k unlike field operations.

The main problem of extending our protocol to rings is that elements may not have an inverse, preventing polynomial interpolation. The idea behind our solution is to work over the ring $\mathbb{Z}_{2^k}[x]/f(x)$, i.e. the ring of all polynomials with coefficients in $\mathbb{Z}_{2^k}$ working modulo a polynomial f that is of the right degree and irreducible over $\mathbb{Z}_2$. As we will see, this will enable us to define enough points on our polynomials which allows interpolation.

Before proceeding we define some notations for this section. Given a polynomial $g(x) \in \mathbb{Z}_{2^k}[x]$, we denote by g_2 the polynomial obtained by reducing each coefficient of g modulo 2. We say that an element a in a ring R is a *zero divisor* if there exists $b \in R$ such that $a \cdot b = 0$. In contrast, a is called a *unit* if it has a multiplicative inverse. It is easy to see that in a finite ring any element is either a zero divisor or a unit.[5]

In the following claim we prove a useful property of the ring $\mathbb{Z}_{2^k}[x]/f(x)$.

Claim 4.4 *Let $f(x) \in \mathbb{Z}_{2^k}[x]$ be a polynomial of degree d such that $f_2(x)$ is irreducible over $\mathbb{F}_2$. Then, any $g(x) \in \mathbb{Z}_{2^k}[x]/f(x)$ is a unit if and only if $g_2(x) \neq 0$.*

The proof appears in Appendix A.

Recall that in our protocol we need to interpolate a polynomial $p(x)$ of degree $2M$. This means that we need to define $2M + 1$ points $(\alpha_0, p(\alpha_0)), \ldots, (\alpha_{2m+1}, p(\alpha_{2m+1}))$ satisfying the property that for each i, j such that $i \neq j$ it holds that $\alpha_i - \alpha_j$ is a unit. From Claim 4.4 it follows that we can choose the elements $\alpha_0, \ldots, \alpha_{2M+1}$ from the set of polynomials over $\mathbb{Z}_{2^k}$ with each coefficient being in $\{0, 1\}$. If $2^d > 2M + 1$, then we will have enough elements in this set. Note that each $\alpha_i - \alpha_j$ is a polynomial which when reduced modulo 2 is not 0 and so by Claim 4.4 is a unit and has an inverse.

The protocol. The protocol works as in Protocol 4.1 with two differences. First, the parties extend the input from elements in $\mathbb{Z}_{2^k}$ to a vector of coefficients in $\mathbb{Z}_{2^k}[x]/f(x)$. This can be done by setting the input to be the free coefficient and adding random d elements. Recall that each input is known by two of the three parties and thus they need to choose jointly these coefficients (for example, by agreeing on a seed from which all the randomness will be derived). The second difference is that the parties use the elements $\alpha_0, \ldots, \alpha_{2M+1}$ as defined above instead of the set $\{0, \ldots, 2M + 1\}$.

Security (sketch). Security is proved exactly as in the proof of Theorem 4.2. The only difference is that the cheating probability of the prover and hence the statistical error of the protocol is computed differently, since the number of roots of a polynomial defined over a ring, may be larger than its degree. To compute this we first prove the following claim.

Claim 4.5 *Let $f(x)$ be as in Claim 4.4. Then, the number of zero divisors in $\mathbb{Z}_{2^k}[x]/f(x)$ is at most $(2^{k-1})^d$.*

The proof appears in Appendix A.

We next show a general claim about the number of roots of a polynomial over a ring based on its number of zero divisors.

[5] if $ab \neq 1$ for any b then there exists two different elements x, y such that $ax = ay$ and so $a(x - y) = 0$ which means that a is a zero divisor

Claim 4.6 *Let R be a commutative ring with unity, such that there are z zero divisors in R and let $g(x)$ be a polynomial of degree δ over R. then, $f(x)$ has at most $z\delta + 1$ roots in R*

The proof appears in Appendix A.

Combining Claim 4.5 and Claim 4.6 we obtain that the number of roots of a degree-δ polynomial $g(x)$ defined over $\mathbb{Z}_{2^k}[x]/f(x)$ is $2^{(k-1)d}\delta + 1$.

We thus can say that the probability that a cheating prover P_i can cause the two other parties to output accept is bounded by $\frac{2^{(k-1)d} \cdot 2M+1}{2^{kd}-M}$ (recall that the two verifiers can query all points in $\mathbb{Z}_{2^k}[x]/f(x)$ except for the points $(\alpha_0, \ldots, \alpha_M)$).

Our result in this section is summarized in the following theorem.

Theorem 4.7 *Let d be such that $2^d > 2M + 1$. Then, our protocol (as described in the text above) securely computes $\mathcal{F}_{\mathsf{vrfy}}$ with abort in the presence of one malicious party with statistical error $\frac{2^{(k-1)d} \cdot 2M+2}{2^{kd}-M}$.*

Setting the parameters and concrete cost. To achieve statistical error that is sufficiently small, it is necessary that 2^d will be much larger than $2M + 1$. Let γ be the smallest number for which $2^\gamma \geq 2M$. Then, we can write

$$\frac{2^{(k-1)d} \cdot 2M + 1}{2^{kd} - M} \leq \frac{2^{(k-1)d} \cdot 2^\gamma + 1}{2^{kd} - M} \approx 2^{-(d-\gamma)}$$

Recall that our protocol yields sub-linear communication when we set $M = \sqrt{m}$, where m is the number of multiplication gates being verified together. This allows us to derive concrete values for d based on m. For example, if we verify $m = 2^{20}$ multiplication gates using our protocol, then $M = 2^{10}$ and $\gamma = 11$. Thus, to achieve statistical security of 40 bits, we can set $d = 51$.

In terms of communication, all we are doing in the protocol is lifting each element in $\mathbb{Z}_{2^k}$ to a d-degree polynomial in $\mathbb{Z}_{2^k}[x]/f(x)$. Thus, communication is blown up by a factor of d and so the number of bits sent in the protocol is $(6L + 2M + 3)k \cdot d$.

As in the protocol for fields, we can set $L = M = \sqrt{m}$. Since d grows logarithmically with m, we obtain that communication is sub-linear also for rings.

Remark 4.8 *As an alternative to the above, it is possible to use a constant d and repeat the protocol multiply times to achieve the desired level of security. Reducing the degree of the extension ring has the potential to improve computation complexity. On the negative side, this comes at the cost of increasing communication. Specifically, if d is constant and σ is the statistical security parameter, then the number of repetitions ℓ is the smallest integer that satisfied the condition $\ell \geq \frac{\sigma}{d-\gamma}$ (since it is required that $\frac{1}{2^{(d-\gamma)\ell}} \leq \frac{1}{2^\sigma}$). The resulted communication cost is $(6L + 2M + 3)k \cdot d\ell$ bits sent by each party.*

5 Achieving Full Security

In this section, we present an augmented version of the 3PC protocol from Section 3 that provides *full security* against 1 corruption: boosting from security with abort to achieve guaranteed output delivery and fairness, while maintaining (amortized) 1 ring element communication per party per multiplication gate.

Our protocol is in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\text{coin}})$-hybrid model: i.e., assuming access to a broadcast channel and (fully secure) coin-toss functionality. We remark that use of a broadcast channel is necessary to achieve full security within this setting, where broadcast is not possible without setup [PSL80]. Full security of both is achievable given PKI serup [RB89]. As our protocol makes minimal use of these functionalities—few calls, with small input size, independent of the evaluated circuit size—any reasonable implementations of $\mathcal{F}_{\mathsf{bc}}$ and $\mathcal{F}_{\text{coin}}$ over PKI will suffice for ensuring the cost of their execution is dominated by other costs.

While player elimination techniques can often be used for upgrading from security with abort to full security (see [IKP$^+$16] for a general result along these lines), these techniques are inefficient when applied to protocols for general circuits. An efficient general compiler was given in [DOS18] that converts a rich class of protocols (including ours) from security with abort to the stronger notion of *fairness*, where either all parties learn the output or none do. However, while the compiler provides small concrete overhead, it does not provide a solution for *full security*; i.e., guaranteed output delivery. This means that any single party can mount a denial-of-service attack without being identified.

To bridge the gap from fairness to full security, there must in particular be means for identifying an honest party in the case of premature abort. This is a not a straightforward task (even given broadcast), as the identification must be done *while preserving privacy*. As we show, this is precisely what our $\mathcal{F}_{\mathsf{vrfy}}$ instantiation achieves.

In what follows, we present the fully secure protocol directly with respect to the particular protocol Π_f as in Protocol 3.2, and protocol Π_{vrfy} implementing $\mathcal{F}_{\mathsf{vrfy}}$ as in Section 4. However, the blueprint can be extended to comparable building blocks, when the following properties are satisfied:

- The protocol Π_{vrfy} realizing $\mathcal{F}_{\mathsf{vrfy}}$ satisfies *recomputable verification*: namely, each verifier sends a single message, calculated as a deterministic function $\mathsf{VMsg}_t(\pi_t, \mathsf{pub})$ of messages from the prover π_t and public values pub. In particular, this means that the prover himself can recompute the expected message of each verifier. As observed, this property holds for our protocol Π_{vrfy} (see Section 4.1, Remark "Recomputable verification.")

- The protocol Π_f for computing f with security with abort has the property that the final messages in the protocol are *robust*, in the sense that each party P_i expects to receive identical messages from P_{i-1} and P_{i+1}. In particular, this means the final output can be reconstructed from just the final message of one honest party.
  Note that robustness holds for our Protocol 3.2, as the final-round messages are parallel executions of the $\mathsf{reconstruct}([\![w]\!], i)$ procedure on replicated secret shares, wherein parties P_{j-1}, P_{j+1} send their (identical) share values w_{j+1} to P_j (see Section 2.1).

High-level protocol description. Recall the challenge is in guaranteeing output delivery in cases where adversarial behavior would cause an abort in the protocol Π_f. As common, we devise means for parties in the face of adversarial behavior to identify and agree upon a party who must necessarily be *honest* (and thus can act as trusted party). In particular, our primary task is to provide an arbitration procedure for the case when a party P_i's proof is rejected by a verifier P_{i-1}, P_{i+1} in the verification procedure Π_{vrfy}. Such rejection can be caused by an invalid proof from corrupted P_i, or malicious acts of a corrupt verifier. Note that in the circuit-emulation phase no abort can occur; nevertheless, an abort during this phase could be resolved by jumping to this verification step, requiring the parties to justify their behavior (in zero knowledge) up to this point.

We observe that the *recomputable verification* property of Π_{vrfy} allows for a simple procedure for an honest prover P_i to identify and accuse a malicious verifier. The verifiers now each broadcast their respective message from Π_{vrfy}; the prover P_i can then recompute and check whether each message is as expected. The length of the messages that the verifier broadcasts is sublinear in the circuit size, $o(|C|)$, and one can further compress for concrete optimization via a few modifications and collision-resistant hashing.

The final challenge is to reconstruct the output in the case that all 3 parties' proofs are accepted (if any is rejected we are in the case addressed above), but where a corrupt party withholds or sends a malformed final message. Here we rely on the *robustness* of the final messages in Π_f. Since each honest party will send his final messages correctly, this guarantees that each party will in fact be able to reconstruct the correct output; however, this may be indistinguishable from a malicious output derived from a malicious final message. To identify which output is correct, we add a MAC for each party, and run the underlying protocol on an augmented authenticated functionality f' that accepts the original party inputs x_i as well as MAC keys s_i, and computes f as well as MACs on the output value with respect to the 3 corresponding keys.

A complete description is given as Protocol 5.2. Leveraging pseudorandomness for share compression as in Section 3, we obtain the following theorem.

Theorem 5.1 (Fully Secure 3PC) *Let R be a finite field, or ring $R = \mathbb{Z}_{2^k}$. Then, for any R-arithmetic circuit C with $o(|C|)$ output wires, there exists a 3-party protocol for computing C in the $(\mathcal{F}_{\mathsf{bc}}, \mathcal{F}_{\mathrm{coin}})$-hybrid model, with the following features:*

- *The protocol makes black-box use of any pseudorandom generator.*
- *The protocol is computationally secure, with guaranteed output delivery against one malicious party.*
- *The communication complexity is $|C| + o(|C|)$ elements of R per party, in addition to a constant number of calls to $\mathcal{F}_{\mathrm{coin}}$ and $\mathcal{F}_{\mathsf{bc}}$ on messages of length $o(|C|)$, where $|C|$ denotes the number of multiplication gates in C.*

A proof for the above theorem can be found in Appendix B.

PROTOCOL 5.2 (Fully Secure 3PC)

- **Inputs:** Each party P_j ($j \in \{1,2,3\}$) holds an input $x_j \in R^\ell$.
- **Auxiliary Input:** The parties hold a description of an arithmetic circuit C that computes f on inputs of length $\ell \cdot 3$.
  Let $\mathsf{MAC} = (\mathsf{MAC.Gen}, \mathsf{MAC.Tag}, \mathsf{MAC.Verify})$ be an information theoretic one-time MAC.
  We denote by Π_f the underlying protocol with security with abort (Protocol 3.2) executed for function f.
- **The base protocol:**
  1. *Input sharing & circuit emulation:* Each party samples a random MAC key, $s_i \leftarrow \mathsf{MAC.Gen}(1^\lambda)$, for desired (statistical) security parameter λ. Execute as in $\Pi_{f'}$, for the modified function $f'((x_1, s_1), (x_2, s_2), (x_3, s_3)) = (y := f(x_1, x_2, x_3), (\mathsf{MAC.Tag}(y, s_i))_{i\in\{1,2,3\}})$, where each party P_i uses input pair (x_i, s_i). If at any point during this phase a party P_i reaches abort, he instead broadcasts via $\mathcal{F}_{\mathsf{bc}}$ "inconsistent," and all parties skip to the Verification stage.
  2. *Verification stage:*
     (a) Begin as in $\Pi_{f'}$: For each prover $j \in \{1,2,3\}$, the parties execute protocol Π_{vrfy} realizing $\mathcal{F}_{\mathsf{vrfy}}$, with input j, the shares of P_j on the input wires, the messages sent by P_j and its randomness for each multiplication gate.
     For each prover $j \in \{1,2,3\}$ and verifier $t \in \{j-1, j+1\}$, denote P_t's verifier message by $v_{t,j}$ (see notation from the text, or Remark 4.3), and P_t's output bit by $\mathsf{Accept}_{t,j}$ (indicating whether P_t accepted the proof of P_j).
     (b) Each party P_t, $t \in \{1,2,3\}$, broadcasts the tuple $(v_{t,t-1}, \mathsf{Accept}_{t,t-1}, v_{t,t+1}, \mathsf{Accept}_{t,t+1})$.
- **Determining outputs:**
  1. *Output reconstruction, if $\exists$ rejection:*
     Let $i \in \{1,2,3\}$ be the minimal rejected prover index, i.e. for which $(\mathsf{Accept}_{i-1,i} \wedge \mathsf{Accept}_{i+1,i}) = 0$.
     (a) P_i accuses: Party P_i identifies whether a party acted maliciously during verification by recomputing the expected verifier messages. If for $t \in \{i-1, i+1\}$ the received message $v_{t,i}$ is not equal to the expected value $v_{t,i} \neq \mathsf{VMsg}_t(\pi_t, x_t, \mathsf{pub})$ (see notation from the text, or Remark 4.3) then P_i selects one such t and broadcasts "Accuse P_t."
     (b) Send inputs to honest party: Let $j^* \in \{i-1, i+1\}$ be the minimal index that was *not* accused in the previous step. (Note that all parties agree on j^* since P_i's accusation is broadcast.) Each party P_j, $j \in \{1,2,3\}$ sends his input x_j directly to party P_{j^*}.
     (c) Honest party computes: Party j^* evaluates f on the received inputs, and sends the output y to all parties.
  2. *Output reconstruction, if $\nexists$ rejection:*
     (a) Each party sends the final reconstruction message as per $\Pi_{f'}$. Denote the message from P_i to P_j, for $i \neq j \in \{1, 2.3\}$, by $\mathsf{msg}_{i,j}$.
     (b) Each P_i, $i \in \{1,2,3\}$: Perform the following to output.
     – Reconstruct the final output values y'_t for $t \in \{i-1, i+1\}$, induced by $\mathsf{msg}_{t,i}$ (by the assumed final-round robustness property of $\Pi_{f'}$; see text). Parse each as $y'_t = (y_t, \sigma_t^1, \sigma_t^2, \sigma_t^3)$.
     – Verify tag: For $t \in \{i-1, i+1\}$, if $1 = \mathsf{Verify}(y_t, \sigma_t^i, s_i)$, then output y_t.
     Otherwise, output fail.

Concrete complexity. Consider the sources of communication and computation overhead of the fully secure protocol, with respect to the underlying protocol Π_f for security with abort:

1. The overhead of computing authenticated f' in the place of f. For arithmetic circuits over a finite field $R = \mathbb{F}$, the information theoretic MAC can be instantiated by taking $\mathsf{MAC.Tag}(y, (a, b)) = y \cdot a + b$ over $\mathbb{F}$, providing small soundness error $|\mathbb{F}|^{-1}$. In such case, the augmented functionality f' requires 6 additional input gates (2 MAC key elements per party) and 3 additional multiplication gates over $\mathbb{F}$ per output gate. For arithmetic computations over a ring $R = \mathbb{Z}_{2^k}$, this MAC provides only soundness error 1/2, and thus must be amplified with a statistical security parameter many independent copies λ. This in turn increases communication by 3λ $\mathbb{Z}_{2^k}$-elements per party, accounting for the extra $\mathbb{Z}_{2^k}$ multiplications to be securely computed. However, we emphasize that the MAC must be securely computed and communicated only once per party per output wire, in contrast to prior approaches in this setting that require MAC values exchanged for every multiplication (e.g., [CGH+18]). In particular, for circuits with small output size, the amortized per-party communication per multiplication gate is not affected.

2. Broadcast of verifier messages $v_{t,t'}$ within the Verification stage (as opposed to sending over a private channel). We describe the protocol with this direct broadcast step for simplicity of description and security analysis. (Since the size of $v_{t,t'}$ is sublinear in $|C|$, this does not affect the amortized communication cost.) In practice, however, one may replace the broadcast of $v_{t,t'}$ with a collision-resistant *hash* $h(v_{t,t'})$, and communicate the full $v_{t,t'}$ only to the other verifier via point-to-point channels (as in the underlying protocol Π_f). Arbitrating a dispute in this case requires an additional step, wherein an honest verifier can accuse the other verifier of acting maliciously. It will always be the case that (a) if a verifier party acts maliciously he will be accused (either by prover or second verifier), and (b) if an accusation is made, either the accuser or the accused must be corrupted.

3. In the case of prover rejection: An additional broadcast of "Accuse P_t" message, plus communication of parties' inputs to the identified honest party P_{j^*}. However, this communication is minimal, and is in anyway in place of the final messages of Π_f.

6 Performance Evaluation

In this section, we provide a full evaluation of our protocol, including benchmark results for the computational effort in the verification step. For simplicity, we consider in this section only the initial protocol, which is only secure with abort. Throughout the section we assume that $M = L = \sqrt{m}$ (where m is the number of multiplication gates in the circuit), in order to have minimal communication overhead.

Recall that our protocol consists of two steps. First, the parties compute the circuit using a semi-honest protocol and then the parties run the verification step.

Communication cost. In the circuit evaluation each party needs to send one ring/field element per multiplication gate. Recall that the verification protocol is executed three times in parallel, each with one of the parties being the prover. Thus, when computing a circuit with m multiplication gates over a large field the overall communication per party is

$$m + 8\sqrt{m} + 3 \text{ field elements.}$$

In contrast, when the computation is carried out over the ring $\mathbb{Z}_{2^k}$ the overall communication per party is

$$m \cdot k + (8\sqrt{m} + 3) \cdot k \cdot d \text{ bits}$$

where d is the degree of each element in the extension ring, and is determined by the statistical security required.

Thus, per one gate the amortized cost is roughly *one ring/field element*, exactly as in the semi-honest protocol.

Computation cost. The semi-honest protocol requires only few simple operations per gate. As shown in [AFL+16, ABF+17, CGH+18] it has the ability to process more than a million multiplication gates per second (for Boolean circuits this increases to over one billion gates per second), thus achieving a very high throughput. The main question is therefore how will the verification step perform.

We now take a deeper look into the verification protocol. Recall that in the first step, the proving party defines $6\sqrt{m}$ polynomials $f_1, \dots, f_{6\sqrt{m}}$ of degree $\sqrt{m}$ and then uses them to define a polynomial $p(x)$ of degree $2\sqrt{m}$ defined by taking $p = g(f_1, \dots, f_{6\sqrt{m}})$. We observe that only interpolation of p is actually required. This follows from the fact that coefficients of p are then sent to the other parties, whereas the f polynomials are only used to compute points on p. Since $\sqrt{m}+1$ points on p are known (these include the points $1, \dots, \sqrt{m}$ which correspond to the outputs of the g gates in the verification circuit), it is required to compute $\sqrt{m}$ additional points to enable interpolation. This can be done by evaluating each f_i on the points $\sqrt{m}+1, \dots, 2\sqrt{m}$ and then computing directly $p(x) = g(f_1(x), \dots, f_{6\sqrt{m}}(x))$. Thus, we can use precomputed Lagrange coefficients to evaluate each f_i on these points without actually interpolating it. Since we have $\sqrt{m}+1$ points on $\sqrt{m}$ polynomials that are used to compute $\sqrt{m}$ new points on these polynomials, this computation is reduced to performing multiplication of two matrices, one that holds the known points on the polynomials and one that consists of constant Lagrange coefficients, where both matrices have roughly $\sqrt{m}$ rows and $\sqrt{m}$ columns. Clearly this is the main computational effort of the prover in the first round, as the remaining work includes interpolation of one polynomial of $2\sqrt{m}+1$ points and choosing an additive sharing of $2\sqrt{m}+1$ elements.

From the two verifiers' point of view in the second step, the main effort is evaluating each of the f polynomials (which are "shared" between the verifiers) on a random point r. This again can be done using Lagrange coefficients, by multiplying a vector of the known points with a vector of Lagrange coefficients. The computational cost here is much lower than what was required in the first round, as evaluation is done on one point only. The remaining task of each verifier includes checking that the output of the circuit is 0, which is done by computing additive shares of the output of each g gate and then taking a random linear combination of these. This again can be viewed as multiplication of a vector of $2\sqrt{m}+1$ polynomial coefficients with the vector $(1, x, x^2, \dots, x^{2\sqrt{m}})$ (which can be precomputed) for each $x \in [\sqrt{m}]$.

The computational cost of the protocol is dominated by the number of multiplication operations, and therefore, asymptotically, the cost is $O(m\sqrt{m})$ operations.

Benchmark results. To estimate the concrete efficiency of our protocol, we ran experiments measuring the running time of each party in the verification protocol, for different numbers of multiplication gates: $2^{10}, 2^{11}, \dots, 2^{20}$. The field we used for our experiment was the 31-bit Mersenne field, i.e. the finite field with $2^{31}-1$ elements, and so we repeated the protocol twice to achieve more than 40-bit security. The experiment was run in an AWS c5.9xlarge instance (Intel Xeon Platinum

m	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Time(msec)	1.0	1.4	2.6	4.8	9.8	20.7	46.4	110.1	303	809.3	2,352.1
Statistical security	2^{-50}	2^{-49}	2^{-48}	2^{-47}	2^{-46}	2^{-45}	2^{-44}	2^{-43}	2^{-42}	2^{-41}	2^{-40}

Table 2: Computation time, measured in milliseconds, of the verification protocol for a *single* party as a function of m - the number of multiplication gates being verified together. The field used is 31-bit Mersenne.

s	**Time (msec)**	**Overall Com. (field elem.)**	**Com. per gate (field elem.)**
1024	1037.3	530,432	0.51
512	716.8	373,800	0.36
256	665.6	263,680	0.25
128	614.4	186,132	0.18
64	627.2	131,456	0.13
32	662.4	92,874	0.09
16	742.4	65,632	0.06
8	880.8	46,389	0.04
4	1,212	32,792	0.03
2	1,618.6	23,182	0.02
1	2,352.1	16,390	0.02

Table 3: Overall computation time and number of field elements sent when verifying $m = 2^{20}$ gates for a *single* party, as a function of s - number of calls to the verification protocol. The field used is 31-bit Mersenne and the statistical security guaranteed is at least 40-bit.

8000 series with clock speed of up to 3.5 GHz). The results appear in Table 2. For each circuit size we indicate the actual statistical security obtained, which is computed by taking $\left(\frac{2\sqrt{m}}{2^{31}}\right)^2$.

Discussion. The verification protocol can be modified to allow a tradeoff between computation and communication, which may be useful for large circuits. As an example, consider the circuit with 2^{20} gates in Table 2, for which the computation time of the verification protocol is more than two seconds.

The idea is to split the circuit into several sub-circuits and verify each sub-circuit separately. Specifically, assume that m gates are divided into s groups of the same size. Then, the number of gates that are verified in each sub-circuit is $\frac{m}{s}$ and so the overall communication cost *of the proof* increases to

$$(8\sqrt{\frac{m}{s}}+3)\cdot s \text{ field elements,}$$

but if the communication complexity of the whole protocol is dominated by the semi-honest part then the larger proof may have a small impact on the overall communication. In contrast, the computation time is reduced by a factor $O(\sqrt{s})$ to $O(\frac{m}{s}\cdot\sqrt{\frac{m}{s}})\cdot s = O(\frac{m\sqrt{m}}{\sqrt{s}})$ operations.

To demonstrate the concrete effect of this idea, we provide a detailed analysis for a circuit consisting of 2^{20} gates in Table 3. In the table, we provide the overall computation time of a single

s	Extension Degree (d)	Overall Com. (bits)	Com. per gate (bits)
1024	46	12,199,936	11.63
512	47	8,784,291	8.38
256	47	6,196,480	5.91
128	48	4,467,163	4.26
64	48	3,154,944	3.01
32	49	2,275,411	2.17
16	49	1,607,984	1.53
8	50	1,159,724	1.11
4	50	819,800	0.78
2	51	591,153	0.56
1	51	417,945	0.4

Table 4: Number of bits sent when verifying $m = 2^{20}$ gates over a Boolean circuit for a *single* party, as a function of s - number of calls to the verification protocol. The verification can be carried out over the extension field $\mathbb{F}_{2^d}$. The statistical security guaranteed is at least 40-bit. Assuming that matrix multiplication over $\mathbb{F}_{2^d}$ is c times slower than matrix multiplication over $\mathbb{F}_p$ for a 31-bit Mersenne prime p (or $c < 1$ if it is faster), the running time for each row of this table is roughly $c/2$ times the running time in the corresponding row of Table 3.

party, overall communication (in field elements) and number of field elements per gate sent by each party, for different sizes of s. As can be seen, we can verify a circuit of 2^{20} gates in less than a second by introducing an additional communication overhead of *less than* 5% of the cost when considering only semi-honest security (this is achieved for example by dividing the gates into 8 groups, i.e., $s = 8$. In this case, computation time is 880.8msec and it requires sending 0.04 field elements per multiplication gate, which is translated to 4% of the semi-honest cost). Observe that the running times do not grow monotonically as one would expect. This is caused most likely since running times of matrix/vector multiplication on modern processors do not scale as predicted, due to cache misses and other architecture dependent issues. We remark that the best known result for this setting is the protocol of [CGH$^+$18] which requires adding 2 field elements beyond the one field element required by the semi-honest protocol. Thus, our protocol outperforms that result even when using this optimization.

We did not implement the version of our verification protocol for the ring $\mathbb{Z}_{2^k}$. Nevertheless, we examined the effect of doing group verification on the communication cost of the protocol. Recall that in this protocol, each element is being lifted to an extension ring and so communication is blown up by a factor which is the degree of the extension ring. In Table 4 we show the communication cost measured by bits, when evaluating a Boolean circuit of 2^{20} multiplication gates, as a function of the parameter s. Since the statistical error of this protocol is $\frac{2\sqrt{m/s}}{2^d}$ where d is the extension degree, we indicate in the second column the value of d for which exactly 40 bit of security is achieved. As mentioned at the end of Section 4.2, as an alternative, it is possible to make d constant so that all computation will be carried out on a smaller ring (for example, the field $\mathbb{F}_{2^{32}}$), and repeat the execution several times to obtain the desired statistical security. We remark that the most efficient protocol for Boolean circuits [ABF$^+$17] requires sending 6 bits in addition to the one bit of the semi-honest protocol. As can be seen in Table 4, when dividing the gates into not-too-many groups, our protocol stays more communication-efficient than [ABF+17] up to $s = 256$; this implies a breakeven point of $m/s \approx 4,000$ gates. The question of finding the optimal balance between computation time (which is determined by (1) the size of the batch of gates that is being verified together and (2) the extension ring that was chosen to work with) and communication cost is highly dependent on the underlying architecture. We thus do not attempt to give a general answer to this question in this work.

Matrix Multiplication. As mentioned above, the main computational bottleneck is the matrix multiplication operation performed by the prover in the first step. The size of the two matrices is roughly $\sqrt{m/s} \times \sqrt{m/s}$. Thus, the largest matrix that we had to multiply in our experiment is of size 1000×1000. We used a naive algorithm for this multiplication but while utilizing Intel's instructions for vectorization of operations to enhance efficiency. It is possible that more advanced algorithms for matrix multiplication would further improve the results when considering large matrices.

7 Optimizing $\mathcal{F}_{\mathsf{vrfy}}$ through recursion

In Protocol 4.1 each party acts as a prover and sends its proof in a single round to the other two parties who act as verifiers, and require an additional round of communication between them to verify the proof. In this section we present a multiple-round verification protocol, which is a fine-tuning of the recursive protocol of [BBC+19] and is more efficient in communication and computation than the protocol of Section 4 (but requires more rounds).

Theorem 7.1 *There is a protocol that securely computes $\mathcal{F}_{\mathsf{vrfy}}$ with abort in the presence of one malicious party, for any Boolean or arithmetic circuit over a finite field or a ring $\mathbb{Z}_{2^k}$ with m multiplication gates, that for statistical error ϵ has $\log m$ communication rounds, requires 27 ring multiplications for each gate in the circuit and*

- *in the case of Boolean circuits or circuits over a finite field, has communication complexity $1 + 4\log m$ field elements in a field $\mathbb{F}$ such that $|\mathbb{F}| \geq 2 + \frac{5\log m+1}{\epsilon}$.*
- *in the case of circuits over a ring $\mathbb{Z}_{2^k}$, has communication complexity $\delta(1 + 4\log m)$ ring elements for $\delta \geq \log\left(2 + \frac{5\log m+1}{\epsilon}\right)$.*

Proof: The proof of Protocol 4.1 consists of shares of a description of p and of the values of $f_i(0)$ for all i. The verifiers then check that the output of the circuit is zero and that $p(r) = g(f_1(r), \ldots, f_L(r))$ for some random field element r.

The high level idea of the recursive proof is that the verifiers only check that the output of the circuit is 0 and then outsource the test that $p(r) = g(f_1(r), \ldots, f_L(r))$ to the prover . Therefore, in the first round, the prover only transmits a description of p and the verifiers respond with a random element r. In the next round, the prover proves that $p(r) - g(f_1(r), \ldots, f_L(r)) = 0$. We show that the statement that is proved in the second round has the same structure as the statement in the first round, but the circuit in the second round has half the number of gates. Furthermore, the verifiers hold additive sharing of the inputs to both circuits. These facts taken together allow the proof system to continue recursively. The details follow.

The proof is defined over a ring R which is sufficiently large to ensure soundness error ϵ. If the original circuit is Boolean or over a finite field then R is a finite field $\mathbb{F}$ while if the original circuit is over a ring $\mathbb{Z}_{2^k}$ then $R = \mathbb{Z}_{2^k}/f(x)$ for an irreducible polynomial $f(x)$ of degree δ over $\mathbb{Z}_2$.

For the i-th multiplication gate in the circuit we defined a degree-2 relation $c : R^6 \to R$ on the six variables $x_{6(i-1)+1}$, …,$x_{6(i-1)+6}$ such that $c(x_{6(i-1)+1}, \ldots, x_{6(i-1)+6}) = 0$ in an honest proof. Define

$$G(x_1, \ldots, x_{6m}) = \sum_{i=1}^{m} \beta_i c(x_{6(i-1)+1}, \ldots, x_{6(i-1)+6}),$$

for random $\beta_i \in R$ that are derived from the shared random source of the two verifiers. If the prover is honest then $G(x_1, \ldots, x_{6m}) = 0$. If $c(x_{6(i-1)+1}, \ldots, x_{6(i-1)+6}) \neq 0$ for some i then $G(x_1, \ldots, x_{6m})$ can be viewed as a (non-zero) degree-1 polynomial in the variable β_i. The number of roots for such a polynomial is at most 1 in $\mathbb{F}$ and $2^{(k-1)\delta}$ in $\mathbb{Z}_{2^k}/f(x)$ (Claim 4.5), and therefore the probability that $G(x_1, \ldots, x_{6m}) \neq 0$ is $1/|\mathbb{F}|$ in the first case and $1/2^{\delta}$ in the second.

proving that $G(x_1, \ldots, x_{6m}) = 0$ can be replaced by proving that

$$\sum_{i=1}^{m} c(A_i(x_1, \ldots, x_{6m})) = A(x_1, \ldots, x_{6m})$$

for $m+1$ appropriate affine functions $A_i : R^{6m} \to R^6$ and $A : R^{6m} \to R$ (here A is simply identically zero).

The prover proceeds by setting $M = 2$, $L = 3m$ and by defining a gate $g(x_1, \ldots, x_{3m}) = \sum_{i=1}^{m/2} c(A_i(x_{6(i-1)+1}, \ldots, x_{6(i-1)+6}))$. The statement to be proved can be viewed as a statement on the sum of two g gates: $g(x_1, \ldots, x_{3m}) + g(x_{3m+1}, \ldots, x_{6m}) = A(x_1, \ldots, x_{6m})$.

The prover constructs $f_1, \ldots, f_L$ by setting f_i to be the lowest degree polynomial such that $f_i(\ell)$ is the $i - th$ input to the $\ell - th$ g gate for $\ell = 1, 2$. Therefore, the degree of all f_i is 1. The prover defines $p = g(f_1, \ldots, f_L)$ and sends additive shares of p as the proof to the other two parties.

The verifiers use shared randomness to send a random ring element r to the prover in response to the proof. Since f_i is of degree at most $M - 1$, computing $f_i(r)$ is possible given the M values $f_i(\ell)$, $\ell = 1, \ldots, M$. Furthermore, the computation is an affine mapping of $f_i(1), \ldots, f_i(M)$, with coefficients that are only a function of r. Therefore, since the verifiers hold an additive sharing of $f_i(\ell)$ for every ℓ, and both know r, they can locally compute additive shares of $f_i(r)$ by mapping their shares with the same affine mapping. The verifiers can similarly compute an additive sharing of $p(r)$.

At this point, the verifiers can check that $p(M) = 0$ by both parties computing a share of $p(M)$ and one of the parties sending its share to the other. The verifiers can optimize this step by deferring the check to the last round. In the last round, each verifier has a sequence of $\log m$ shares: $\alpha_1, \ldots, \alpha_{\log m}$ for the first and $\gamma_1, \ldots, \gamma_{\log m}$ for the second, such that purportedly $\alpha_j + \gamma_j = 0$ for every j. The verifiers can then use their shared randomness source to generate random $r_1, \ldots, r_{\log m}$ and test that $\sum_{j=1}^{\log m} r_j(\alpha_j + \gamma_j) = 0$ by exchanging a single field element. If the sum of every pair of shares is indeed zero then the test passes. If there exists a pair such that their sum is not zero then the probability of the test passing is $1/|\mathbb{F}|$ for $R = \mathbb{F}$ and $1/2^{\delta}$ for $R = \mathbb{Z}_{2^k}/f(x)$.

In the next round, the prover proves that $g(f_1(r), \ldots, f_L(r)) - p(r) = 0$. Since the values $f_1(r), \ldots, f_L(r), p(r)$ are affine functions of the inputs, this statement is exactly of the form

$$\sum_{i=1}^{m/2} c(B_i(x_1, \ldots, x_{6m})) = B(x_1, \ldots, x_{6m})$$

for appropriate affine functions $B_1, \ldots, B_{m/2}, B$. Therefore, the proof protocol of the first round can be executed again, further compressing the circuit for the next round.

After $\log m$ rounds the circuit has a constant number of gates and the protocol of Section 4 is executed completing the proof on the small circuit.

For $R = \mathbb{F}$, in each round of the protocol, the soundness error is $\frac{2M+1}{|\mathbb{F}|-M}$ due to the same argument used to bound the soundness error of the non-recursive protocol. Taking into account the additive $1/|\mathbb{F}|$ terms due to ensuring that $p(M) = 0$, the soundness error of the recursive protocol is at most $\frac{5\log m+1}{|\mathbb{F}|-2}$. Therefore to ensure soundness error at most ϵ the field size must satisfy $|\mathbb{F}| \geq 2 + \frac{5\log m+1}{\epsilon}$.

For $R = \mathbb{Z}_{2^k}/f(x)$, in each round of the protocol the soundness error is $\frac{2M+1}{2^\delta - M}$. Taking into account the additive $1/2^\delta$ term, the soundness error of the recursive protocol is at most $\frac{5\log m+1}{2^\delta-2}$. Therefore to ensure soundness error at most ϵ the polynomial degree δ must satisfy $\delta \geq \log\big(2 + \frac{5\log m+1}{\epsilon}\big)$.

Communication cost. The verification protocol has $\log m$ rounds. In each round, the prover P_i sends a sharing of the coefficients of the polynomial p to the two verifiers and they return a random element r. Since the degree of p is equal to 2 in all the rounds, it can be represented by three elements. One verifier sends another element at the end. Put together, there are $1 + 4\log m$ elements sent in this implementation of $\mathcal{F}_{\mathsf{vrfy}}$.

Computation cost. We count the number of ring multiplications, which is the dominant factor in the computation of both the prover and the verifier.

The main effort of the prover is to compute a new polynomial p in each round. Recall that p is of degree 2 and that it is defined by taking $p = g(f_1, \ldots, f_{6m_j/2})$ (where m_j is the number of gates in round j). This can be carried out by evaluating each of the polynomials f_i on one more point, which requires only ring additions when the three points are carefully chosen. The prover also computes the polynomial p, which requires evaluating $g(f_1, \ldots, f_{3m_j})$. The gate g is made up of $m_j/2$ invocations of the circuit c, and each c has three multiplication gates. Therefore, computing one point of p requires $3m_j/2$ multiplications, and computing the three points that enable interpolation requires $9m_j/2$ multiplications. At the end of the round, the prover needs also to evaluate each f on the challenge point r which requires one ring multiplication per f_i and $3m_j$ altogether. In total, the number of ring multiplications the server computes in the j-th round is $15m_j/2$. Therefore, the prover computation across the $\log m$ rounds in which $m_0 = m$ and $m_j = m_{j-1}/2$ is $15m$ multiplications.

For the two verifiers, the main effort in each round is to evaluate the "sharing" of each polynomial f_i on the challenge point r which requires $3m_j$ ring multiplications. Across all rounds the verifier computation is $6m$ ring multiplications.

Since in the protocol each party plays the role of the prover once and the role of the verifier twice, the total number of ring multiplications is $(2 \cdot 6 + 15)m = 27m$. ■

Remark 7.2 *The recursive protocol as described is an interactive, multi-round protocol. The proof can be heuristically reduced to a single round protocol via a variant of the Fiat-Shamir heuristic [FS86] (see Section 6.2.3 of the full version of [BBC+19]). However, for protocols with logarithmically many rounds, there are still significant gaps in our understanding of the soundness of this heuristic when analyzed in the random oracle model [PS00, BCS16].*

Circuit size	**Comm. (Field el.)**	**Comm. (bytes)**	**Comm. (Ring ele.)**
m	$1 + 4 \log m$		$\delta(1 + 4m)$
2^{15}	61	360	2867
2^{20}	81	480	3807
2^{25}	101	600	4747
2^{30}	121	720	5808

Table 5: Overall communication for the recursion based implementation of $\mathcal{F}_{\mathsf{vrfy}}$ as a function of m, the number of multiplication gates in a given (Boolean or arithmetic) circuit. The statistical security guaranteed is at least 40-bits, which requires a field of size 2^{47} (or ring $\mathbb{Z}_{2^k}/f(x)$ with $f(x)$ of degree $\delta = 47$) for a circuit with $2^{15} - 2^{25}$ multiplication gates and a field of size 2^{48} (or ring $\mathbb{Z}_{2^k}/f(x)$ with $f(x)$ of degree $\delta = 48$) for a circuit with 2^{30} multiplication gates. Communication is measured in field elements for arithmetic circuits over all finite fields, in ring elements for arithmetic circuits over all rings $\mathbb{Z}_{2^k}$ and in bytes for Boolean circuits and arithmetic circuits over fields no larger than 2^{47}.

Acknowledgments

This work is supported in part by ERC Project NTSC (742754). E. Boyle additionally supported by ISF grant 1861/16 and AFOSR Award FA9550-17-1-0069. N. Gilboa additionally supported by ISF grant 1638/15 and ERC grant (876110). Y. Ishai additionally supported by ISF grant (1709/14), NSF-BSF grant (2015782), and a grant from the Ministry of Science and Technology, Israel and Department of Science and Technology, Government of India. A. Nof additionally supported by the BIU Cyber Center.

References

[ABF+17] Toshinori Araki, Assi Barak, Jun Furukawa, Tamar Lichter, Yehuda Lindell, Ariel Nof, Kazuma Ohara, Adi Watzman, and Or Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 843–862, 2017.

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 805–817, 2016.

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 67–97, 2019. Full version: ePrint report 2019/188.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, pages 31–60, 2016.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10, 1988.

[BHKL18] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale P2P mpc-as-a-service and low-bandwidth MPC for weak participants. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 695–712, 2018.

[BJPR18] Megha Byali, Arun Joseph, Arpita Patra, and Divya Ravi. Fast secure computation for small population over the internet. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 677–694, 2018.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 192–206, 2008.

[BNTW12] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19, 1988.

[CCPS19] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. ASTRA: high throughput 3pc over rings with application to secure prediction. *IACR Cryptology ePrint Archive*, 2019:429, 2019.

[CDE$^+$18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. Spdz2k: Efficient MPC mod 2k for dishonest majority. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 769–798, 2018.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 342–362, 2005.

[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. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, pages 34–64, 2018.

[CRFG19] Dario Catalano, Mario Di Raimondo, Dario Fiore, and Irene Giacomelli. Monz2ka: Fast maliciously secure two party computation on z2k. *IACR Cryptology ePrint Archive*, 2019:211, 2019.

[DOS18] Ivan Damgård, Claudio Orlandi, and Mark Simkin. Yet another compiler for active security or: Efficient MPC over arbitrary rings. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, pages 799–829, 2018.

[EOP$^+$19] Hendrik Eerikson, Claudio Orlandi, Pille Pullonen, Joonas Puura, and Mark Simkin. Use your brain! arithmetic 3pc for any modulus with active security. *IACR Cryptology ePrint Archive*, 2019:164, 2019.

[FL19] Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. *IACR Cryptology ePrint Archive*, 2019:658, 2019. To appear in CCS 2019.

[FLNW17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 225–255, 2017.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GI99] Niv Gilboa and Yuval Ishai. Compressing cryptographic resources. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 591–608, 1999.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229, 1987.

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

[GRW18] S. Dov Gordon, Samuel Ranellucci, and Xiao Wang. Secure computation with low communication from cross-checking. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part III*, pages 59–85, 2018.

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 359–378, 2015.

[IKN$^+$19] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. *IACR Cryptology ePrint Archive*, 2019:723, 2019.

[IKP$^+$16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference*, pages 430–458, 2016.

[ISN89] Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 525–537, 2018.

[Mau06] Ueli M. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006. Earlier version in SCN 2002.

[MR18] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52, 2018.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 591–602, 2015.

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. Minimising communication in honest-majority mpc by batchwise multiplication verification. In *International Conference on Applied Cryptography and Network Security*, pages 321–339. Springer, 2018.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85, 1989.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

A Proofs for the Ring Protocol

In this section, we provide detailed proof for three claims presented in Section 4.2.

Claim A.1 (Claim 4.4 – restated) *Let $f(x) \in \mathbb{Z}_{2^k}[x]$ be a polynomial of degree d such that $f_2(x)$ is irreducible over $\mathbb{F}_2$. Then, any $g(x) \in \mathbb{Z}_{2^k}[x]/f(x)$ is a unit if and only if $g_2(x) \neq 0$.*

Proof: Assume that $g_2(x) \neq 0$. Since f is irreducible modulo 2, there exists $a(x), b(x) \in \mathbb{F}_2(x)$ such that $a(x)f_2(x) + b(x)g_2(x) = 1$ over $\mathbb{F}_2$. Therefore, there exists a polynomial $h(x)$ such that $a(x)f(x) + b(x)g(x) = 1 + 2h(x)$ over $\mathbb{Z}_{2^k}$. Taking

$$A(x) = a(x)\sum_{j=0}^{k-1}(-2h(x))^j \quad \text{and} \quad B(x) = b(x)\sum_{j=0}^{k-1}(-2h(x))^j$$

we obtain that

$$\begin{aligned} A(x)f(x) + B(x)g(x) &= \textstyle\sum_{j=0}^{k-1}(-2h(x))^j(a(x)f(x) + b(x)g(x)) \\ &= \textstyle\sum_{j=0}^{k-1}(-2h(x))^j - (-2h(x))^{j+1} \\ &= 1 - (-2h(x))^k \equiv 1 \bmod 2^k \end{aligned}$$

which means that $g(x)$ is invertible in $\mathbb{Z}_{2^k}[x]/f(x)$ and so is a unit as required.

Next, assume that $g_2(x) = 0$. Then, $g(x) = 2h(x)$ for some polynomial $h(x)$ and so clearly it is a zero divisor and not a unit. ■

Claim A.2 (Claim 4.5 – restated) *Let $f(x)$ be as in Claim 4.4. Then, the number of zero divisors in $\mathbb{Z}_{2^k}[x]/f(x)$ is at most $(2^{k-1})^d$.*

Proof: From Claim A.1, $g(x) \in \mathbb{Z}_{2^k}[x]/f(x)$ is a zero divisor if $g_2(x) = 0$. This means that each of the d coefficients of $g(x)$ is a multiple of 2. Thus, there are at most $(2^{k-1})^d$ polynomials which are zero divisors as required. ■

Claim A.3 (Claim 4.6 – restated) *Let R be a commutative ring with unity, such that there are z zero divisors in R and let $g(x)$ be a polynomial of degree δ over R. then, $f(x)$ has at most $z\delta$ roots in R*

Proof: This is proved by induction. For $\delta = 1$, let $g(x) = a + bx$ and let $u_1, \dots u_n$ be the roots of $g(x)$. It follows that for each $j = 2, \dots, n$ it holds that $b(u_j - u_1) = 0$ which means that $u_j - u_1$ is a zero divisor. Since there are z zero divisors in R, it implies that $n \leq z$.

Next, assume that the claim holds $\delta - 1$ and let $g(x)$ a polynomial of degree δ over R. If all the roots of g are zero divisors then the claim holds. Otherwise, there is a ring element r which is not a zero divisor such that $g(r) = 0$. This implies that we can write $g(x) = (x - r)h(x)$ where $h(x)$ is a polynomial of degree $\delta - 1$. Thus, by the induction hypothesis it follows that $h(x)$ has $z(\delta - 1)$ roots. Let u be a root of $g(x)$ which is not a root of $h(x)$. This means that $0 = (u - r)h(u)$ and so $u - r$ is a zero divisor. There are at most z roots which implies that $g(x)$ has at most $z(\delta - 1) + z = z\delta$ as required. ■

B Proof for the Full Security Protocol

In this section, we sketch the proof of Theorem 5.1.

At a high level, simulation takes place exactly via the simulator $\mathcal{S}_{f'}$ of the underlying protocol $\Pi_{f'}$, together with direct emulation of computation and communication steps of non-sensitive information.

More formally, let P_i be the corrupted party and let $\mathcal{S}$ be the ideal world adversary. The simulation works as follows:

1. $\mathcal{S}$ extracts the inputs of P_i (including MAC key s_i), and simulates the circuit emulating step, and verification stage as in $\mathcal{S}_{f'}$. (If the simulation leads to any party aborting during the circuit emulation phase, $\mathcal{S}$ broadcasts "inconsistent" if relevant, and skips directly to the verification stage on the truncated circuit.)

   - For P_i prover: The messages $v_{j,i}$ and output bits $\mathsf{Accept}_{j,i}$ for honest verifiers $j \in \{i-1, i+1\}$ are directly computable given the proof information sent by P_i and public (already simulated) information.
   - For P_i verifier, P_j prover for some $j \in \{i-1, i+1\}$: The simulator $\mathcal{S}_{f'}$ produces (among other things) messages $v_{j',j}$ and output $\mathsf{Accept}_{j',j}$ on behalf of the honest verifier party $P_{j'}$ (where $j' \in \{1,2,3\} \setminus \{i,j\}$), as sent to corrupt verifier P_i.

2. $\mathcal{S}$ simulates broadcasting all honest party values $(v_{j,\ell}, \mathsf{Accept}_{j,\ell})$, for each $j \in \{i-1, i+1\}$ and $\ell \in \{1,2,3\} \setminus \{j\}$.

3. If any of the provers is rejected, simulate as follows.

   - If P_i is the minimal rejected prover: $\mathcal{S}$ receives a message "Accuse P_t" for $t \in \{i-1, i+1\}$, and then receives a message x'_i from P_i, directed to the third party P_ℓ.
   - If P_j is the minimal rejected prover, for honest $j \in \{i-1, i+1\}$: $\mathcal{S}$ simulates broadcasting "Accuse P_i," and receives a message x'_i from P_i directed to the third party P_ℓ.

   $\mathcal{S}$ invokes the ideal f trusted party functionality on corrupt input x'_i, and receives answer y. $\mathcal{S}$ then simulates sending y to P_i on behalf of the selected third honest party.

4. If no prover was rejected, simulate as follows.

- $\mathcal{S}$ invokes the ideal f trusted party functionality on the corrupt input x_i extracted in Step 1, and receives answer y (corresponding to the unauthenticated function output). $\mathcal{S}$ honestly computes MACs on y: σ_i on behalf of corrupt P_i given the key s_i extracted in Step 1, and σ_j using fresh key s_j for each of $j \in \{i-1, i+1\}$.
- Finally, $\mathcal{S}$ uses the output $y' = (y, \sigma_1, \sigma_2, \sigma_3)$ together with the simulation procedure of $\mathcal{S}_{f'}$ to simulate the final-message values $\mathsf{msg}'_{j,i}$ of honest parties $j \in \{i-1, i+1\}$ sent to P_i.

Correctness of simulation up to the Determining Outputs phase follows directly. In the case where no prover was rejected, correct simulation follows by robustness of the final messages in $\Pi_{f'}$ together with unforgeability of the MAC. For the case where a prover was rejected, it remains to show that the party P_{j^*} selected within the protocol will necessarily be an *honest* party. Let P_j be the minimal-index rejected prover within the simulation.

- Case 1: P_j is corrupt ($j = i$). Then j^* is honest, since the rejected prover is never selected as j^*.
- Case 2: P_j is honest ($j \neq i$). Then P_j's messages within the circuit emulation phase are honestly consistent (i.e., he is proving a true statement), and he honestly generates and communicates the proof to the two verifier parties. In order for such proof to be rejected, (by completeness) the corrupt verifier must have sent an incorrect verifier message, which will be identified by the prover P_j in the arbitration phase. In turn, the honest prover will Accuse P_i; thus, j^* will be the remaining party, who is necessarily honest.