# Best of Both Worlds in Secure Computation, with Low Communication Overhead

Daniel Genkin[2,3], S. Dov Gordon[1], and Samuel Ranellucci[1,2]

[1] George Mason University
[2] University of Maryland
[3] University of Pennsylvania

**Abstract.** When performing a secure multiparty computation with a few hundred parties, using the best protocols known today, bandwidth constraints are the primary bottleneck. A long line of work demonstrates that $n$ parties can compute a circuit $C$ of depth $d$ while communicating $O(|C| \log |C| + \mathsf{poly}(d, n)$ field elements per party, as long as a majority of parties are honest. However, in the malicious majority setting, a lot less is known. The work of Nielsen and Ranellucci is the first to provide constant-overhead in the communication complexity when a majority of parties are malicious; their result demonstrates feasibility, but is quite complex and impractical.

In this work, we construct a new MPC protocol in the pre-processing model. We introduce a new middle-ground: our protocol has low communication and provides robustness when a majority of parties are honest, and gives security with abort (possibly with higher communication cost) when a majority of players are malicious. Robustness is impossible when a majority of parties are malicious; viewing the increased communication complexity as a form of denial of service, similar to an abort, we view our result as providing the "best of both worlds".

## 1 Introduction

After a decade of improvements in the computational cost of secure multiparty computation, we have reached a point where the primary performance bottleneck is the communication complexity, even when computing with only a moderate number of parties Most constructions require that $n$ participants communicate a total of $O(Cn^2)$ field elements to compute a circuit of size $C$. The $n^2$ term stems from point-to-point communication at every multiplication gate, which, at first glance, seems hard to avoid. Amazingly, when a majority of parties are honest, there are several constructions that require communicating only $O(C)$ field elements.[4] Very broadly, these constructions make use of two ideas to lower communication cost. First, by using a randomly chosen dealer, they can reduce the communication channels from $O(n^2)$ to $O(n)$. This requires care, to ensure that a malicious dealer cannot corrupt the computation. Second, by using "packed secret sharing", the participants can communicate just one field element to compute $O(n)$ multiplication gates. In a bit more detail, multiple wire values are simultaneously encoded using a single threshold secret sharing scheme: to

[4] Additionally, they have an additive term that is polynomial in $n$.

encode $\ell$ wire values, $w_1, \cdots, w_\ell$, a random polynomial $p$ is chosen such that $p(-j) = w_j$. As usual, $p(1), \cdots, p(n)$ define the secret shares of the $n$ parties, and, for a degree $t + \ell$ polynomial, all $\ell$ secrets remain perfectly hidden against $t$ colluding parties. Since $t + \ell < n$, this provides a tradeoff between security and efficiency; as more values are packed into the secret sharing, the number of corruptions that can be tolerated decreases. With a small blowup in the circuit description, these polynomials can be used to compute $\ell$ multiplication gates at a time, cutting the communication cost by a factor of $\ell = O(n)$ [9].

In the malicious majority setting, a lot less is known about reducing the communication complexity. The recent work of Nielsen and Ranellucci is the first and only protocol with constant communication cost per circuit gate [16]. The result of their work is exciting, as it demonstrates feasibility for the first time. However, as the authors state, their protocol "is solely of theoretical interest"; it has constants that are large and difficult to compute, and, conceptually, it requires parsing a complex composition of player emulations and subprotocols.

In this work, we propose an optimistic approach to communication complexity. Our protocol has constant *expected* communication complexity if a majority of players are *honest*. However, unlike prior work in the honest majority setting, we stress that our protocol also remains secure when a majority of players are malicious, albeit with higher communication complexity. At a high level, the variation in communication complexity stems from the following feature of our approach. We choose a random dealer and hope that they are honest. If the dealer happens to be malicious, he can force a re-start of the protocol, and if $O(n)$ consecutive dealers are malicious, then they can force the communication complexity to blow up. Taking the view that this increased communication cost is simply a form of denial of service, we view our result as providing "the best of both worlds" with respect to denial of service; when a majority of parties are malicious, it is impossible to prevent a denial of service attack, as the adversary can always force an abort. While Nielsen and Ranellucci show that, technically, it is possible to achieve low communication when a majority of players are malicious, the benefit of our relaxation is that it allows us to construct a much simpler protocol, both in concept and in concrete complexity.

The phrase "best of both worlds" has been used before in the MPC literature, referring to the more common notion of denial of service: guaranteed output delivery [15, 13, 8]. With only a few exceptions, protocols for secure multiparty computation are usually designed with a particular corruption threshold in mind. They either provide security with guaranteed output delivery when a majority of parties are honest, but provide no security at all when a majority are malicious, or they provide security with abort when a majority of parties are corrupt, but allow a denial of service even if only a single party is corrupt. Our protocol provides the best of both worlds in this sense as well, giving security with guaranteed output delivery when the adversary fails to corrupt a majority of parties, and security with abort when a majority are corrupt.

Our construction relies on offline (data independent) preprocessing that, currently, we do not know how to compute with constant overhead (short of using

Nielsen and Ranellucci). While we hope this reliance can be removed in future work, we note that there are settings where it might be very reasonable to use such preprocessing. The obvious case is where the parties can afford to send a lot of data prior to the arrival of their inputs, but another setting in which preprocessing is available is where the parties have access to some trusted setup.

**Formal description of our result.** For privacy threshold $t_p$, and packing parameter $\ell$, our protocol enables $n$ players to compute any arithmetic circuit $C$, guaranteeing security with abort when fewer than $t < t_p$ players are corrupt. It achieves guaranteed output delivery (aka: robustness, full security) when $t < t_r$, where $t_r = (n - t_p - 2 \cdot \ell)/2$. In addition, if $t < t_r$ and $\ell \in \Omega(n)$, then for a circuit $C$ of size $|C|$ and depth $d$, our protocol has expected communication complexity of $O(|C| \log |C| + \mathsf{poly}(n, d))$.

**Related Work.** Our work follows from two lines of work. The first line of work focuses on achieving low overhead computation in the majority setting, this includes the work of [2, 14]. The paper of [3] achieved a sublinear overhead in the number of players, but only in the computational setting and with overhead in the security parameter that is not sublinear. The paper of [2] showed how by selecting $\ell \in \Omega(n)$, it was possible to construct a protocol for $n$ parties with communication overhead of $O(|C| \log |C| + \mathsf{poly}(n, d))$ for a circuit $C$ of size $|C|$ and depth $d$.

The second line of work [15, 13, 8] focuses on finding mpc protocols with tradeoffs between how many corruptions can be tolerated before privacy is compromised ($t_p$), and how many corruptions can be tolerated before the robustness guarantee is lost ($t_r$). Ishai et al. demonstrated that this is possible when there is some slack in the parameters: there exist $n$-party protocols where, for $t_p + t_r < n$, the protocol maintains security with guaranteed output delivery against $t_r$ malicious players and security with abort against $t_p$ malicious parties [13]. In the same work, they demonstrated that this slackness is inherent, by giving an example of a function that cannot be securely computed with these same guarantees if $t + s = n$.

In parallel to our work, the work of [12] used the assumption that a certain number of parties are honest to improve the efficiency of semi-honest GMW and BMR-style MPC protocols. Other approaches that use preprocessing (such as [6, 4, 7]) require each player to communicate one field element per multiplication since they do not use packing.

### 1.1 Technical Overview

In this section we present a high level overview of our protocol. We begin by describing a semi-honest version of our protocol, in order to provide insight into how we achieve low communication complexity. (Note that we never give a formal description of this semi-honest version, and it is meant purely for intuition.) Borrowing techniques from [2, 3, 5], we use a $t_p$-private packed Shamir secret sharing scheme with packing parameter $\ell$. These polynomials have degree $t_p + \ell$, and we will maintain this degree as we compute the circuit.

In order to compute multiplication gates, our protocol uses a special designated party (called the dealer), and Beaver triples $[\ \boldsymbol{a}\ ], [\ \boldsymbol{b}\ ], [\ \boldsymbol{c}\ ]$, which are secret sharings of values $a, b, c \in \mathbb{F}^\ell$, where $a, b$ are randomly sampled and $c = a \cdot b$ (introduced in [1]). These triples are shared using a $t_p$-private Shamir packed secret sharing scheme with packing parameter $\ell$. The packing parameter $\ell$ allows players to compute pointwise multiplication on vectors of field elements by having each player compute and send a constant number of field elements to the dealer.

Our protocol evaluates an arithmetic circuit $C$ in topological order from the input to the output gates. Since packed Shamir secret sharing is linear, the players can locally compute on their shares in order to evaluate the addition gates of $C$. To compute the product $[\ \boldsymbol{z}\ ] = [\ \boldsymbol{x}\ ] \cdot [\ \boldsymbol{y}\ ]$, the players execute the following steps, using a Beaver triple $[\ \boldsymbol{a}\ ], [\ \boldsymbol{b}\ ], [\ \boldsymbol{c}\ ]$. First, the players locally compute shares of $\boldsymbol{x - a}$ and $\boldsymbol{y - b}$ and send them to the dealer. The dealer reshares $\boldsymbol{x - a}$, $\boldsymbol{y - b}$ and $\boldsymbol{(x - a) \cdot (y - b)}$ using degree $\ell$ polynomials. By resharing and packing those values instead of sending them in the clear, we cut down the communication cost by a factor $\ell$; the secret sharing in this step has nothing to do with privacy. The players then compute shares of $\boldsymbol{w} \leftarrow \boldsymbol{y} \cdot \boldsymbol{(x - a)} + \boldsymbol{x} \cdot \boldsymbol{(y - b)} + r$, where the random mask $r$ is sampled and secret shared during preprocessing, using a degree $t_p + \ell$ polynomial. Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are of degree $t_p + \ell$, and $\boldsymbol{(y - b)}$ and $\boldsymbol{(x - a)}$ are of degree $\ell$, it follows that $\boldsymbol{w}$ is of degree $t_p + 2\ell$. The players send their shares of $\boldsymbol{w}$ to the dealer. The dealer re-shares $\boldsymbol{w}$ using a degree $\ell$ polynomial, and the players compute $\boldsymbol{z} = \boldsymbol{w} - r$. Since $r$ is shared using a degree $t_p + \ell$ polynomial, this results in shares of a degree $t_p + \ell$ polynomial, maintaining the invariant.

The use of a dealer allows each user to send secret shares to one party, instead of $n$ parties, cutting the cost per gate from $O(n^2)$ to $O(n)$. The packed secret sharing further reduces the complexity from $O(n)$ to $O(n/\ell)$. However, note that this also forces us to increase the degree of the polynomial to $t_p + \ell$, which creates a tradeoff between privacy and efficiency: the closer $t_p$ is to $n$, the smaller $\ell$ must be.

**Attacks by Malicious adversaries.** The protocol above is only secure against a semi-honest adversary. At a high level, an active adversary, which instructs the players or the dealer to deviate from the protocol specification, can mount two types of attacks.

**Additive Attacks.** The first class of attacks occur either when a corrupt dealer re-shares the wrong value, or when malicious players send invalid shares to an honest dealer, thereby causing the dealer to reconstruct and re-share the wrong value. As we describe in our proof sketch in Section 4, these attacks are actually instances of additive attacks, in which an adversary can tamper with the evaluation of circuits by adding or subtracting values on individual wires, but cannot impact the computation in any other way. See the full version of this paper for more detail. By running the protocol on an additively secure circuit, obtained from the compiler of Genkin et al. [10], we are able to construct a protocol for MPC that renders such an attack ineffective. At a high level, the

compiler of Genkin et al. takes any circuit and transforms it into a new circuit that will output $\perp$ if the adversary applies an additive attack (i.e. tampers with the value of any wire). By showing that any attack on our protocol is equivalent to an additive attack, we can apply the protocol of [10] to make it secure. We note that [10] has a constant overhead.

**Divide and Conquer Attacks.** The second class of attacks can only be performed by a malicious dealer. At a high level, during the evaluation of multiplication gates, instead of re-sharing values using a degree-$\ell$ polynomial, the dealer can create two sets of shares, each consistent with a different degree-$\ell$ polynomial.

More formally, consider the following situation: let $n$ be the number of parties, let $M$ be a set of corrupted parties, and let $S_1, S_2$ be distinct sets of honest parties (not necessarily disjoint). The adversarial dealer sends shares to $S_1$ such that the secret recovered from those shares is $x - a$. He sends shares to $S_2$ such that the secret recovered from those shares is $x - a + 1$. Then, when the players try to compute shares of $(x - a) \cdot y + r$, where $r$ is a random mask, note that both $S_1 \cup M$ and $S_2 \cup M$ give the dealer enough shares to reconstruct the blinded secret: from $S_1 \cup M$, the dealer can recover $(x-a) \cdot y + r$ and from the shares of $S_2 \cup M$, the dealer can recover $(x-a+1) \cdot y + r$. By subtracting $(x-a+1) \cdot y + r$ from $(x - a) \cdot y + r$ the malicious dealer can recover $y$, even though the value of $(x - a) \cdot y$ is supposedly hidden by a random mask.

**The Degree-Test Protocol.** Dealing with this second type of attack is one of our main technical contributions. In Section 3 we present a novel *degree-test* protocol that takes secret shares from the dealer, and transmits them to the players if only if the shares of the honest players are consistent with a polynomial $p$ of degree-$d$. This degree test is also efficient, requiring each player to exchange only a constant number of field elements with the dealer. The main idea behind this protocol is as follows. During preprocessing, all parties learn a portion of a secret that is encoded in a degree $n-1$ polynomial, $w$. Additionally, they receive shares of a degree $n - d - 1$ polynomial, $v$, such that $v(0) = 0$. To prove that he shared a degree $d$ polynomial, the dealer collects $n$ shares of $z$, defined as $z \leftarrow p \cdot v + w$. If $p$ is of appropriate degree, this suffices to learn $w(0)$, revealing the secret value, while if the degree of $p$ is too high, $w(0)$ remains hidden and the dealer fails to prove that he acted honestly.

## 2 Best of Both Worlds Security

We prove our protocols secure under the ideal-world, real-world paradigm. We define $f_C$ as the ideal functionality that takes an input $x$ from the players and outputs $C(x)$. The functionality $f_C^A$ takes an input $x$ from the players, and a vector $A$ from the adversary. It also evaluates $C$ on $x$, but it allows an adversary to tamper with the evaluation by adding values on individual wires; the variable $A$ specifies the values that are added to each wire.

**Definition 1.** *Let $t_p \leq n$ be positive integers, let* SD *denote the statistical distance, and let $0 \leq \epsilon \leq 1$. We say that an $n$-party protocol $\pi$ $(t_p, \epsilon)$-securely realizes a functionality $\mathcal{F}$ if for every PPT real-world adversary $A$ which corrupts at most $t_p$ players, there exists a simulator $S$ such that*

$$\mathsf{SD}(Real_{\pi,A}, Ideal_{\mathcal{F},S}) \leq \epsilon.$$

*We naturally extend this definition to protocol in the $g$-hybrid model by replacing $Real_{\pi,A}$ above with $Real_{\pi,A,g}$. In this case we say that $\pi$ $(t_p, \epsilon)$-securely realizes $\mathcal{F}$ in the $g$-hybrid model.*

**Definition 2.** *Let $n \geq t_p \geq t_r$ be positive integers and let $0 \leq \epsilon \leq 1$. We say that an $n$-party protocol $\pi$ $(t_r, t_p, \epsilon)$-robustly realizes $(f_C, f_C^A)$ if it meets the following two conditions.*

1. ***Security.*** *If $t_p > t \geq t_r$ then $\pi$ $(t_p, \epsilon)$-securely realizes $f_C^A$ as per Definition 1. This property does not guarantee that players receive outputs, because the adversary can cause the protocol to abort in the real world.*
2. ***Robustness.*** *If $t_r > t$ then $\pi$ $(t_p, \epsilon)$-securely realizes $f_C$, and it is guaranteed that the protocol will successfully terminate, with each honest player receiving output. More formally, if less than $t_r$ players are corrupt, the output generated in the real world is the same that is produced by the functionality $f_C$ in the ideal world where (i) each honest player $P_i$ provides input $x_i$ to the functionality, and (ii) the ideal functionality selects a default input for each corrupted player that does not provide an input $x_i$.*

## 3 Degree Test

Our degree test protocol is an interactive proof between a single prover (dealer) and multiple verifiers (players). The dealer sends a field element to each player, and proves that these elements are consistent with a polynomial $p$ of degree at most $d$. We construct a proof where the prover can only convince a given verifier with probability $2^{(-4s/n-t_p-\ell)}$. The aim is that at least $n - t_p - \ell$ verifiers will not be convinced by a cheating prover. The protocol proceeds as follows. The preprocessing functionality randomly samples a binary string of size $\frac{4s}{n-t_p-\ell}$, encodes it as $\texttt{secret} \in \mathbb{F}$, and sends a portion of `secret` to each player. After sharing the polynomial $p$, the players will interact with the dealer in a manner that allows the dealer to learn this `secret` if and only if $p$ is of degree $d$ or less. The dealer then proves that he learned `secret` by sending to each player the portion of the binary string that they received during the preprocessing. If some player does not receive the correct part of the secret that was given to him during preprocessing, the player complains about the dealer.

In more detail, the preprocessing phase will generate a random degree-$(n - d - 1)$ polynomial $v$ such that $v(0) = 0$. Additionally, the preprocessing phase generates a random string, encodes it in $\mathbb{F}$, and shares $\texttt{secret} \in \mathbb{F}$ using a random degree-$(n-1)$ polynomial $w$ (that is, $w(0) = \texttt{secret}$). Finally, the

individual bits of the binary string are distributed among the $n$ participating players (we assume a large enough field $\mathbb{F}$ to facilitate this). Upon receiving $p(i)$ from the dealer in the online phase, the player $P_i$ computes $z(i) \leftarrow p(i) \cdot v(i) + w(i)$ and sends it to the dealer. In case $(p(1), \cdots, p(n))$ are not consistent with any degree-$d$ polynomial, the dealer cannot reconstruct the value `secret` since the degree of $z$ is larger than $n-1$. As a result, the dealer would only be able to break soundness with a small number of players. The remaining players will complain, and conclude that the dealer is a cheater. On the other hand, if the dealer shared a low degree polynomial, the dealer can reconstruct `secret` by interpolating $z(1), \cdots, z(n)$, and can then use `secret` as a proof that indeed $(p(1), \cdots, p(n))$ define a degree-$d$ polynomial. This can be done by sending each $P_i$ its portion of the binary string encoded as `secret`.

**Attack on Shares by Corrupt Players.** Even if the dealer gives shares $p(1), \cdots, p(n)$ consistent with a low degree polynomial, it may be that corrupt players would send back bad shares to prevent the dealer from reconstructing the correct secret, or by refusing to send shares altogether. To solve this problem, we allow the dealer to verify shares and eliminate players that send bad shares.

We allow the dealer to verify that $P_i$ sent a share that equals $p(i) \cdot v(i) + w(i)$ by (1) having the preprocessing phase authenticate the shares $v(i), w(i)$ that it sends to each player $P_i$, (2) using the linearly homomorphic MAC from SPDZ, and (3) by giving the verification keys to the dealer. When a dealer complains about a player, the player will be eliminated and will no longer take part in any future degree tests with that dealer.[5] We use $E$ to denote the set of eliminated players.

**Properties about the Set of Eliminated Players.** We need certain guarantees about the set of eliminated players. First, if the dealer is honestly sharing a low degree polynomial, then no honest player will complain about the dealer. Second, if the dealer is malicious and does not share a low degree polynomial, then a large number of honest players will eliminate themselves. Third, we must ensure that every player has a consistent view of the set of eliminated players. We satisfy this last property using a secure broadcast anytime a player is eliminated. If a large number of players are eliminated, then the dealer is replaced and protocol restarts with a new dealer. We can safely remove the dealer in this case, because, either the dealer is corrupt, or there are enough corrupt players that we can give up on robustness.

**Recovering From Eliminated Players.** The fact that the dealer can eliminate players creates a new problem: how does the dealer reconstruct `secret` when a few players have been eliminated? Recall that `secret` is shared using a degree-$n-1$ polynomial, and that eliminated players no longer provide shares to the dealer. In order to replace the eliminated players, we have the non-eliminated players send additional information that will allow the dealer to recover the missing shares of $z$. A natural approach for this is to have each remaining player

[5] If the protocol re-starts because the dealer is thrown out, the party will rejoin the computation.

send the dealer a share (generated during the preprocessing phase) of the eliminated player's share. While this solution works, it is too costly, as it introduces a quadratic overhead in the number of players. This overhead stems from two facts: first, a linear number of players could be eliminated, and second, for each execution of the degree test, for each eliminated player, each non-eliminated player would have to send one share to the dealer.

**Reducing Recovery Overhead.** We employ a couple of strategies to reduce the communication required of the honest players when they help the dealer to reconstruct the shares of eliminated players. First, we will reuse the same $v$ for each execution of the degree test. Now, when a player $P_i$ is eliminated by the dealer, each player will only need to send a share of $v_i$ to the dealer once, ensuring that the dealer learns $v_i$ for all further executions of the degree test protocol. Next, we notice that the dealer recovers `secret` from the shares of $z$ by performing Lagrange interpolation, which is a linear operation. That is, the dealer computes $\texttt{secret} = \sum_{i=1}^{n} \alpha_i z_i = \sum_{i=1}^{n} \alpha_i (p(i) \cdot v_i + w_i)$ where $\alpha_1, \cdots, \alpha_n$ are the Lagrange interpolation coefficients. Rewriting the above equation,

$$\begin{aligned}\texttt{secret} &= \sum_{i=1}^{n} \alpha_i z_i = \sum_{P_i \notin E} \alpha_i z_i + \sum_{P_i \in E} \alpha_i z_i = \sum_{P_i \notin E} \alpha_i z_i + \sum_{P_i \in E} \alpha_i (p(i) \cdot v_i + w_i) \\ &= \sum_{P_i \notin E} \alpha_i z_i + \sum_{P_i \in E} \alpha_i p(i) \cdot v_i + \sum_{P_i \in E} \alpha_i w_i.\end{aligned}$$

Since the dealer knows $p(i)$ for all players, knows $v_i$ for all eliminated players, and has the shares $z_i$ for all non-eliminated players, he only needs to learn $\bar{c} = \sum_{i \in E} \alpha_i w_i$. Thus, each non-eliminated player can locally compute a single share of $\bar{c}$, using a share of $w_i$ for each $P_i \in E$. Sending just this single share to the dealer, instead of one share for every eliminated player, allows us to avoid the linear overhead that arose in the naive approach previously suggested.

### 3.1 Formal Description of The Degree Test Protocol

In this section we formally present and analyze our degree test protocol. Let $H$ be the set of honest players and let $E$ denote a global, shared variable, indicating the set of eliminated players. We denote the inputs to the degree test protocol by $p(1), \cdots, p(n)$. For some honest player, $P_i$, we let $\eta$ denote the probability that a malicious dealer wrongly convinces $P_i$ that $p$ is of degree less than or equal to $d$. Finally, we denote the set of parties complaining about the dealer by $C$. The ideal functionality, $\mathcal{F}_{\text{dt}}$, is formally described in Fig. 1, and the degree test protocol realizing this functionality is described in Fig. 2. Consider the following theorem.

**Theorem 1.** *$\pi_{dt}$ securely realizes $\mathcal{F}_{dt}$ in the preprocessing-hybrid model.*

In order to prove Theorem 1, we provide two simulators, one simulator for the case when the dealer is honest, and a second simulator for the case when the dealer is corrupt. In each case the simulator simply follows the description

We denote $E$ to be a global variable which denotes the set of eliminated players.

**Input**

1. Dealer: $\{p(i)\}_{P_i \notin E}$. In case the dealer is honest we require that $(p(1), \cdots, p(n))_{P_i \notin E}$ is the evaluation of a polynomial of degree at most $d$.

**Compute**

1. Initialize a set of complaining players $C \leftarrow \emptyset$.
2. Let $p$ be the polynomial that results from interpolating the shares $\{p(i)\}_{P_i \in H}$
3. On receipt of $(\text{bad\_share\_complaint}, i)$ from the adversary set $C \leftarrow C \cup \{ P_i \}$, unless both the dealer and $P_i$ are honest.
4. If $C \neq \emptyset$ , set $E \leftarrow E \cup C$ and terminate.
5. If $\mathsf{degree}(p) > d$, each honest party is added to $C$ with probability $1 - \eta$.
6. On receipt of $(\text{bad\_proof\_complaint}, i)$ from the adversary set $C \leftarrow C \cup \{ P_i \}$ unless both the dealer and $P_i$ are honest.
7. If $C \neq \emptyset$, set $E \leftarrow E \cup C$ and terminate.

**Output**

1. Party $P_i$ receives $p(i)$

**Fig. 1.** Degree Test Functionality $\mathcal{F}_{\text{dt}}$

of the protocol, determines if players or the dealer needs to complain, and adds players to the set of eliminated players $E$ that would be eliminated. We recall that $H$ denotes the set of honest players. $E$ denotes the set of eliminated parties. The point $v_i \in \mathbb{F}$ is a share of a degree $n - t_p - \ell$ polynomial that evaluates to zero at zero. The point $w_i \in \mathbb{F}$ is a share of a polynomial that evaluates to a random value $\texttt{secret}$. The share $v_{i,j}$ is a resharing of $v_i$ that will help the dealer to reconstruct $v_i$ if $P_i$ is eliminated.

**Degree Test Simulation Honest Case.** The simulator queries the ideal functionality and receives $p(i)$ for each $P_i \in \bar{H}$.

1. The simulator simulates the preprocessing by following its description.
2. The simulator sends $p(i)$ to each non-eliminated corrupt player $P_i \in \bar{E} \cup \bar{H}$
3. The simulator await that corrupted non-eliminated player $P_i \in \bar{E} \cup \bar{H}$ sends $(z_i, \mathsf{m}(z_i))$ and $a_i$ to the dealer.
4. The simulator computes $\mathsf{k}(z_i) \leftarrow p(i) \cdot \mathsf{k}(v_i) + \mathsf{k}(w_i)$, assigns to $S$ the subset of corrupted non-eliminated players $P_i$ who either did not send a $z_i$ with a valid mac tag or who did not send an $a_i$. All players in $S$ are added to $E$. If any players were added to $E$, he runs the player elimination simulation (below).
5. Send $\texttt{secret}_{m\cdot(i-1)+1,\cdots,m\cdot i}$ to each non-eliminated corrupt player$P_i$
6. For each player $P_i$ who complains about the dealer, the simulator sends $(\text{bad\_proof\_complaint}, i)$ to the functionality and then add $P_i$ to $E$. The simulator then runs the Player elimination simulation (below).

**Honest Dealer Elimination Simulation.** Whenever a player is eliminated, we require that the simulator do the following. After a set $S$ of players are added to $E$, the simulator awaits $(i, j, v'_{j,i})$ from each non-eliminated corrupt player for each $P_j \in S$. The, for each $P_j \in S$, the simulator tries to reconstruct $v_j$ from the $v'_{j,i}$ that come from non-eliminated corrupt players,

The degree test protocol is parameterized by security parameter $s$, $n$ players, threshold $t$, degree $d$, and field $\mathbb{F}$ where $|\mathbb{F}| \geq \frac{2ns}{n-t}$. Each player gets a point $p(i) \in \mathbb{F}$ from the dealer. The global variable $E$ stores the set of eliminated players and may be updated. If $E$ is updated, all parties will halt and discard their shares. Players in $E$ do not send shares to dealer. Otherwise, they know that $\{p(i)\}_{P_i \in H}$ lies on a polynomial of degree at most $d$. We define $\alpha_1, \cdots, \alpha_n$ as the lagrange coefficients. We use a homomorphic mac scheme $(\mathsf{setup}, \mathsf{keygen}, \mathsf{mac}, \mathsf{verify})$

**One-time preprocessing**

1. $(v_1, \cdots, v_n) \leftarrow \mathsf{share}_{n-d}(0)$
2. Send $\gamma \leftarrow \mathsf{setup}(1^s, \mathbb{F})$ to the dealer. (global key for mac scheme).
3. Send $k_i \leftarrow \mathsf{keygen}(1^s, \mathbb{F})$ to the dealer. (key used to authenticate $p(i) \cdot v_i$)
4. Send $v_i, t_i \leftarrow \mathsf{mac}(\gamma, k_i, v_i)$ to $P_i$
5. Generate shares to reconstruct values in case players are eliminated.
   $v_{i,j} \leftarrow \mathsf{share}_t(v_i)$ and send $v_{1,j}, \cdots, v_{n,j}$ to $P_j$

**Preprocessing (per instance)**

1. Set $m = \frac{2ns}{n-t-\ell}$ and sample a random $\mathtt{secret} \in \{0,1\}^m$.
2. $(w_1, \cdots, w_n) \leftarrow \mathsf{share}_n(\mathtt{secret})$ (Hides the secret.)
3. Send $\delta_i \leftarrow \mathsf{keygen}(1^s, \mathbb{F})$ to the dealer.
4. $\tau_i \leftarrow \mathsf{mac}(\gamma, \delta_i, w_i)$ ($\tau_i + p(i) \cdot t_i$ produces a mac tag for $p(i) \cdot v_i + w_i$)
5. Send $(w_i, \tau_i)$ and $\mathtt{secret}_{m/n\cdot(i-1)+1,\cdots,m/n\cdot i}$ to $P_i$.
6. Generate shares to recover in case players are eliminated.
   $w_{i,1}, \cdots, w_{i,n} \leftarrow \mathsf{share}_{t_p}(w_i)$ and send $w_{1,j}, \cdots, w_{n,j}$ to $P_j$.

**Protocol online phase**

1. **Dealer**
   Send $p(i)$ to $P_i$ for each $P_i \notin E$.
2. **Players**
   i. $z_i \leftarrow p(i) \cdot v_i + w_i$ ($z_i$ is a share of $z$ such that $z(0) = \mathtt{secret}$)
   ii. Send $z_i, \sigma_i \leftarrow p(i) \cdot t_i + \tau_i$ to the dealer.
   iii. Generate shares when players are eliminated.
   Send $a_i \leftarrow \sum_{P_j \in E} \alpha_j \cdot w_{j,i}$ to the dealer. (local shares of $\sum_{P_j \in E} \alpha_j w_j$)
3. **Dealer**
   i. Dealer complains about each non-eliminated player $P_i$ who either: did not send a $z_i$ with a tag $\sigma_i$ such that $\mathsf{verify}(\gamma, p(i) \cdot k_i + \delta_i, z_i, \sigma_i) = \mathtt{accept}$; or did not send an $a_i$ when players have been eliminated. If the dealer complained, $E$ is updated to include all the players that he complained about, and the dealer skips to 5.i.
   ii. Generate shares when players are eliminated.
   – $\bar{c} \leftarrow \mathsf{recover}([\, \boldsymbol{a_i} \mid \boldsymbol{P_i} \notin \boldsymbol{E} \,])$ ($\bar{c} = \sum_{P_j \in E} \alpha_j \cdot w_j$)
   – $\bar{c} \leftarrow \bar{c} + \sum_{P_i \in E} \alpha_i (v_i \cdot p(i))$
   iii. $\mathtt{secret} \leftarrow \left(\sum_{P_i \notin E} \alpha_i \cdot z_i\right) + \bar{c}$ and send $\mathtt{secret}_{m\cdot(i-1)+1,\cdots,m\cdot i}$ to $P_i$
4. **Players**
   i. If dealer did not complain, and $P_i$ received a different value for $\mathtt{secret}$ from the dealer then what he got in the preprocessing, then he complains. $P_i$ is added to $E$. Each player $P_i$ who received a complaint from the dealer (resp. $P_j$) about $P_j$ (resp. Dealer) sends $(i, j, v_{j,i})$ to the dealer.
5. **Dealer**
   i. The dealer reconstructs $v_j$ from the $v_{j,i}$ he receives using Reed-Solomon decoding. If the dealer cannot reconstruct $v_j$, the dealer broadcasts failure, and then $E$ is set to be the set of all players.

**Fig. 2.** Degree Test $\pi_{\mathrm{dt}}$

and the $v_{j,i}$ that were generated in the preprocessing for the honest players. If the simulator does not reconstruct a valid share $v_j$, the dealer broadcasts failure, and the full set of players is added $E$. The simulation then halts.

**Description of the simulator when dealer is corrupt.**

1. The simulator simulates the preprocessing by following its description.
2. The simulator await that the dealer send $p(i)$ to each non-eliminated honest player $P_i$.
3. The simulator computes $z_i \leftarrow p(i) \cdot v_i + w_i$, $a_i \leftarrow \sum_{P_j \in E} \alpha_j \cdot w_{j,i}$ (local shares of $\sum_{P_j \in E} \alpha_j w_j$), $\mathsf{m}(z_i) \leftarrow p(i) \cdot \mathsf{m}(v_i) + \mathsf{m}(w_i)$ and sends $(z_i, \mathsf{m}(z_i))$ and $a_i$ to the dealer.
4. For each non-eliminated player $P_i$ that the dealer complains about, the simulator send $(\text{bad\_proof\_complaint}, i)$ to the functionality. The simulator executes the player elimination simulation.
5. The simulator await $\texttt{secret}_{m\cdot(i-1)+1,\cdots,m\cdot i}$ from the dealer for each honest non-eliminated player $P_i$.
6. For each non-eliminated honest player $P_i \in \bar{E} \cup H$, if the dealer did not send the same value of `secret` that would have been to $P_i$, the simulator sends $(\text{bad\_proof\_complaint}, i)$ to the functionality. The simulator executes the player elimination simulation.

**Corrupt Dealer Elimination Simulation.** Whenever a player is eliminated players we require that the simulator do the following. After a set $S$ of players are added to $E$, the simulator sends $(i, j, v'_{j,i})$ for each eliminated player $P_j \in S$ and non-eliminated honest player $P_i$. If the corrupt dealer broadcasts failure, then $E$ is set to be the set of all players. The simulation then halts.

### 3.2 Properties of the degree-test protocol

We already know that the degree test protocol securely realizes the degree test functionality. Within the context of our main protocol, we want to show that our degree test protocol has more features than what is directly provided by the functionality. The first is that the online cost of the degree test is low, namely that if we use the protocol many times, the overhead of the degree test per player will be small. The second condition that we want is that if the dealer is honest, and less than some threshold of players are dishonest, then in each execution of the degree test, either it succeeds, or some malicious party is eliminated.

The third condition that we are interested in is that if a corrupt dealer cheats by sharing a high degree polynomial, and does not complain about the shares and tags given to him by honest players, then less than half of the honest player's will accept the secret. (Recall, if he does complain about some of the shares and tags that he was given, then all parties are eliminated and the protocol re-starts with a new dealer.)

**Lemma 1.** *The total communication cost of running $m$ executions of the degree test with the same dealer is $O(s \cdot n \cdot m + \mathsf{poly}(n))$ bits.*

*Proof.* We enumerate over each item that is communicated and compute its associated communication overhead. A player will broadcast a complaint about the dealer at most once ($O(n^2)$). The dealer will broadcast a complaint about a player at most once. ($O(n^2)$). Each player will send a constant number of field elements to the dealer per execution of the degree test ($O(s \cdot n \cdot m)$). The dealer will send a constant number of field elements to each player per execution of the degree test ($O(s \cdot n \cdot m)$).

The communication complexity of all these items is $O(s \cdot n \cdot m + \mathsf{poly}(n))$. This completes the proof of this lemma

**Lemma 2.** *If the dealer is honest, and less than $\frac{(n-t_p-2\ell)}{2}$ players are corrupt, then both of the following conditions will be met (except with negligible probability)*

1. *No honest player will be eliminated.*
2. *The degree test will succeed, or at least one corrupt player will be eliminated.*

*Proof.* First, we show that an honest player will not be eliminated by an honest dealer except with negligible probability. Since honest players always send correct shares, the dealer would only eliminate an honest player if he reconstructs an incorrect value for the secret in step 3.iii. This can only occur if the adversary is able to successfully forge a mac tag in step 2.ii. Otherwise, the dealer would complain about a corrupt player and the degree test would terminate. Since forging a mac tag only succeeds with negligible probability, this completes the first part of the proof.

Next we proceed to show that either the degree test will succeed, or at least one corrupt player will be eliminated. If the dealer reconstructs the correct secret, the dealer will send the correct part of the secret to each honest player in step 3.iii, and each honest player will accept the secret in step 4.i. This leaves only two strategies for the adversary to prevent the degree test from succeeding: he can either send bad shares, or not send shares at all. In either case, the dealer will complain about corrupt players in step 3.i and the dealer will eliminate corrupt players. This completes the second part of the proof.

**Lemma 3.** *If less than $t_p$ players are corrupt and the following conditions all hold, then more than $\frac{n-t_p-2\ell}{2}$ honest players will be eliminated.*

1. *The dealer is malicious and does not complain about a player in step 3.i.*
2. *The degree of the polynomial $p$ is greater than $\ell$.*

*Proof.* First, we show that if all the above conditions hold then the dealer cannot learn any information about the secret. Since by condition 2, the dealer shares a polynomial $p$ of degree higher than $d$, then the degree $p \cdot v$ is greater than $n$. Since $w$ was selected at random, and less than $t_p$ players are corrupt, then the dealer cannot recover the secret from $(p \cdot v + w)(0)$.

By the first condition of the lemma, the dealer did not complain in step 3.i. This means that the dealer, to convince an honest player that he is honest must correctly guess the part of the secret given to that player. Since the probability

of correctly guessing the secret for a given player is $2^{-\frac{4s}{n-t_p-\ell}}$, we can finally show that more than half the honest players will abort.

By combining the following two statements with the lemma below, we have what we want: (1) the probability of correctly guessing a player's secret is $p = 1 - 2^{-4s/(n-t_p-\ell)}$ and (2) the random variables associated to the dealer successfully guessing players' part of the secret are independent.

**Lemma 4.** *Given $s, m \in \mathbb{N}$, let $X_1, \ldots, X_m$ be independant Bernoulli variables with success probability $p = 1 - 2^{-4s/m}$ and let $X = \sum_{i=1}^{m} X_i$ then*

$$\Pr\left[X < \tfrac{m}{2}\right] \leq 2^{-\theta(s)}$$

*Proof.* If $m \geq s$, we can directly apply Chernoff's bound to get this result. We have that $\mu = m \cdot (1 - 2^{-4s/m})$ and let $\delta = \frac{1}{2(1-2^{-4s/m})}$. We note that $\delta \geq \frac{1}{2}$ and that $\mu \cdot \delta = \frac{m}{2}$ and thus we have that

$$\Pr\left[X \leq \tfrac{m}{2}\right] = \Pr\left[X < (1-\delta)\cdot\mu\right] \leq e^{\frac{-\delta^2\mu}{3}} \leq e^{-\frac{m}{12}} \in 2^{-\theta(s)}$$

This leaves only the case where $m < s$ and we show that this also holds by using the following combinatorial argument.

$$\begin{aligned}\Pr\left[X < \tfrac{m}{2}\right] &= \sum_{i=0}^{m/2} \binom{m}{i} \left(1 - 2^{-4s/m}\right)^i \left(2^{-4s/m}\right)^{m-i} \leq \sum_{i=0}^{m/2} \binom{m}{i} \left(2^{-4s/m}\right)^{m-i} \\ &\leq \sum_{i=0}^{m/2} 2^m \left(2^{-4s/m}\right)^{m/2} \leq 2^{-2s+m+\log m} \in 2^{-\theta(s)}\end{aligned}$$

## 4 Additively-secure protocol

We now construct a protocol which is secure in the preprocessing-hybrid model, aside from allowing additive attacks. (Recall, these are then handled using the compiler of Genkin et al. [10].) The players will randomly elect (without repetition) a dealer that will be used to run the computation. If at some point too many players claim the dealer is cheating, the protocol will be restarted with a new dealer. During the evaluation phases (routing, multiplication and addition), the players will add, multiply and subtract shares locally, and they will also send and receive shares to and from the dealer. The dealer will be responsible for receiving, reconstructing values and resharing them. In particular, the dealer will be responsible for reconstructing values when less than $t_r$ shares are corrupted. This will be done by having the dealer apply Reed-Solomon decoding to the shares he receives. If at any point, the dealer fails to reconstruct the secret, the dealer will eliminate himself and the protocol will be restarted with a new dealer. The protocol employs the degree testing protocol to ensure that when the dealer reshares values, he cannot use a polynomial of degree greater than $\ell$.

Since degree testing involves eliminating players, the protocol will need to keep track of who has been eliminated.

At the beginning, the players will randomly elect a dealer. While that dealer is active, each party will keep track of a set of eliminated players denoted by the variable $E$. Players can eliminate themselves if they detect malicious behavior, or they can be eliminated, either by an honest dealer for acting maliciously, or by a malicious dealer, arbitrarily. If the set of eliminated players grows too big, all parties kick out the dealer and rejoin the protocol with a new dealer (chosen without replacement). To simplify exposition, we assume that the set $E$ is a global variable, and that all honest parties agree on its members. In practice, this can be achieved using a broadcast channel, without impacting the claimed communication cost. Our main protocol consists of four phases: the preprocessing phase, the input phase, the evaluation phase, and the output phase. The input, evaluation, and output phase will all rely on values generated by the preprocessing. We will not describe the preprocessing phase in its entirety but rather describe which values each of the other phases need from the preprocessing.

Throughout the computation, parties hold shares of wire values, encoded using polynomials of degree $t = t_p + \ell$. This ensures that $t_p$ parties cannot learn anything about these values. Because we need to multiply these polynomials by degree $\ell$ polynomials that encode masked wire values, the degree of the polynomials becomes $t_p + 2\ell$ during the evaluation. This allows us to error-correct in the presence of less than $\frac{n-t_p-2\ell}{2}$ corruptions, maintaining robustness as claimed in our theorem. When we do not specify the degree of a sharing, we mean that the polynomial has degree $t_p + \ell$.

Below, we let $t = t_p + \ell$.

**Input**

1. Randomly sample $r \in \mathbb{F}^\ell$ and send it to the sender.
2. Send $[\, \boldsymbol{y} \,] \leftarrow \mathsf{share}_t(r)$ to players

**$\rho$-gate**

1. Randomly sample $\boldsymbol{u},\ \boldsymbol{v} \in \mathbb{F}^\ell$ such that $\rho(\boldsymbol{u}) = \boldsymbol{v}$.
2. Send $[\, \boldsymbol{c} \,] \leftarrow \mathsf{share}_t(u),\ [\, \boldsymbol{z} \,] \leftarrow \mathsf{share}_t(v)$ to players.

**Multiplication**

1. Randomly sample $\alpha, \beta, \gamma, \upsilon \in \mathbb{F}^\ell$ such that $\alpha \cdot \beta = \gamma$.
2. $[\, \boldsymbol{a} \,] \leftarrow \mathsf{share}_t(\alpha),\ [\, \boldsymbol{b} \,] \leftarrow \mathsf{share}_t(\beta),\ [\, \boldsymbol{c} \,] \leftarrow \mathsf{share}_t(\gamma),$
   $[\, \boldsymbol{u} \,] \leftarrow \mathsf{share}_t(\upsilon)$
3. Send $[\, \boldsymbol{a} \,], [\, \boldsymbol{b} \,], [\, \boldsymbol{c} \,], [\, \boldsymbol{u} \,]$ to players.

**Output**

1. $r \in_R \mathbb{F}^\ell$. $[\, \boldsymbol{r} \,] \leftarrow \mathsf{share}_t(r)$
2. Send $[\, \boldsymbol{r} \,], [\![ \boldsymbol{r} ]\!]$ to players. ($[\![ \boldsymbol{r} ]\!]$ is a VSS sharing of $r$.)

**Fig. 3.** Preprocessing

The players want to evaluate a SIMD circuit $C$ with $m$ input gates $g_1, \cdots, g_m$ with output gates $g_{|C|-O+1}$ to $g_{|C|}$. We denote $g_{m+1}, \cdots, g_{|C|-O}$ as evaluation gates. For each input gate $g$, we denote $p_g$ as the player who provides input $x_g$ and we will use the variable $[\, \boldsymbol{i_g} \,]$ to store the shares generated by $\mathcal{F}_{\text{pre}}$ for the gate $g$.

For each multiplication gate $g$, we will use the variable $[\, \boldsymbol{a_g} \,], [\, \boldsymbol{b_g} \,], [\, \boldsymbol{c_g} \,]$ and $[\, \boldsymbol{u_g} \,]$ to store the shares generated by $\mathcal{F}_{\text{pre}}$ for the gate $g$. For each $\rho$-gate, we will use the variable $[\, \boldsymbol{v_g} \,], [\, \boldsymbol{w_g} \,]$ to store the shares generated by $\mathcal{F}_{\text{pre}}$ for gate $g$. We use $t_g$ to denote the shares generated by evaluating the gate $g$. For each output gate, We will use the variable $[\, \boldsymbol{\Upsilon_g} \,]$ to store the shares generated by packed secret sharing for the output phase and define $[\, \boldsymbol{\Phi_g} \,]$ to denote the shares generated using the VSS.

We use $E$ to denote the set of eliminated players; it is a global variable which is sometimes updated in the sub-protocols. Each round will have a fresh dealer. We denote $\mathcal{D}$ as the set of eliminated dealers.

**Preprocessing**

1. For each input gate $g$
   The players execute $\mathcal{F}_{\text{pre}}(\mathsf{input})$ where $p_g$ taking the role of the sender. The sender stores the value he receives as $r_g$ and the shares received by players are assigned to $[\, \boldsymbol{i_g} \,]$.
2. For each $\rho$-gate $g$, $[\, \boldsymbol{v_g} \,], [\, \boldsymbol{w_g} \,] \leftarrow \mathcal{F}_{\text{pre}}(\rho)$
3. For each multiplication gate $g$, $[\, \boldsymbol{a_g} \,], [\, \boldsymbol{b_g} \,], [\, \boldsymbol{c_g} \,], [\, \boldsymbol{u_g} \,] \leftarrow \mathcal{F}_{\text{pre}}(\mathsf{multiplication})$
4. For each output gate $g$,
   (a) $[\, \boldsymbol{y_g} \,], [\![\, \boldsymbol{r_g} \,]\!] \leftarrow \mathcal{F}_{\text{pre}}(\mathsf{output})$
   (b) $[\, \boldsymbol{\Upsilon_g} \,] \leftarrow [\, \boldsymbol{y_g} \,] \qquad [\, \boldsymbol{\Phi_g} \,] \leftarrow [\![\, \boldsymbol{r_g} \,]\!]$

**Dealer replacement rule.**

If at any point $|E_i| \geq \frac{n-t_p}{2} - \ell$, the player sets $\mathcal{D} \leftarrow \mathcal{D} \cup \{\, D \,\}$ and return to step 3 of the evaluation.

**Evaluation**

1. For each input gate $g$, $[\, \boldsymbol{t_g} \,] \leftarrow \mathsf{Input}(x_g, r_g, [\, \boldsymbol{i_g} \,])$ where $x_g$ and $r_g$ are the inputs of sender $p_g$.
2. $\mathcal{D} \leftarrow \emptyset$, $E \leftarrow \emptyset$.
3. Sample a random dealer not in $\mathcal{D}$.
4. For each evaluation layer $L$ and for each gate $g \in L$.
   (a) If $L$ is an addition layer then $[\, \boldsymbol{t_g} \,] \leftarrow [\, \boldsymbol{x} \,] + [\, \boldsymbol{y} \,]$
   (b) Otherwise
   - If $L$ is a routing layer then
     $[\, \boldsymbol{x} \,] \leftarrow [\, \boldsymbol{t_{\mathsf{input}(g)}} \,]$
     $[\, \boldsymbol{t_g} \,] \leftarrow \mathsf{routing}([\, \boldsymbol{x} \,], [\, \boldsymbol{v_g} \,], [\, \boldsymbol{w_g} \,])$
   - If $L$ is a multiplication layer then
     $[\, \boldsymbol{x} \,] \leftarrow t_{\mathsf{left}(g)}$, $[\, \boldsymbol{y} \,] \leftarrow t_{\mathsf{right}(g)}$
     $[\, \boldsymbol{t_g} \,] \leftarrow \mathsf{multiplication}([\, \boldsymbol{x} \,], [\, \boldsymbol{y} \,], [\, \boldsymbol{a_g} \,], [\, \boldsymbol{b_g} \,], [\, \boldsymbol{c_g} \,], [\, \boldsymbol{u_g} \,])$
   - If $|E| \geq \frac{n-t_p}{2} - \ell$, $\mathcal{D} \leftarrow \mathcal{D} \cup \{\, D \,\}$ and return to step 3 of the evaluation.
   - If $[\, \boldsymbol{t_g} \,] = \perp$, return to step 4.b (This occurs when a player is eliminated during a degree test.)
5. For each output gate $g$
   (a) $[\, \boldsymbol{x} \,] \leftarrow [\, \boldsymbol{t_{\mathsf{input}(g)}} \,]$
   (b) $\boldsymbol{v} \leftarrow \mathsf{output}([\, \boldsymbol{x} \,])$
   (c) Output $\boldsymbol{v}$

**Fig. 4.** Main Protocol

**Input phase.** In the input phase, a sender provides his input $\boldsymbol{x}$, and the other parties receive shares of that input, which can then be used in the evaluation phase. The preprocessing functionality randomly samples $\boldsymbol{r} \in \mathbb{F}^\ell$, gives the value to the sender, and provides $[\ \boldsymbol{r}\ ]$ to the other players. The sender broadcasts $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{r}$. The players then compute $[\ \boldsymbol{x}\ ] = [\ \boldsymbol{r}\ ] + \boldsymbol{y}$. Due to a lack of space, we provide the full description in the full version of the paper.

**Output phase.** In the output phase, parties take shares $[\ \boldsymbol{x}\ ]$ of the output and reconstruct $\boldsymbol{x}$. We have to limit the adversary to an additive attack on the revealed output, ensuring that the adversary cannot arbitrarily choose the output. The preprocessing functionality creates two shares of a value $\boldsymbol{r} \in \mathbb{F}^\ell$, once using packed Shamir secret sharing resulting, in $[\ \boldsymbol{r}\ ]$, and once using a VSS, resulting in $[[\ \boldsymbol{r}\ ]]$. The players will use $[\ \boldsymbol{r}\ ]$ and $[[\ \boldsymbol{r}\ ]]$ to mask and then unmask $\boldsymbol{x}$. That is, they locally, homomorphically add $r$ to the output by adding their shares of $[\ \boldsymbol{x}\ ]$ and $[\ \boldsymbol{r}\ ]$, and then reveal $r$ by opening the VSS sharing. Because VSS is binding, the adversary can only modify the value before it is unmasked. As such, the only attack that can be done is an additive attack. Due to a lack of space, we provide the full description in the full version of our paper.

**Multiplication.** The multiplication is the most complex operation, the goal is to take shares $[\ \boldsymbol{x}\ ], [\ \boldsymbol{y}\ ]$ and produce shares of $[\ \boldsymbol{x} \cdot \boldsymbol{y}\ ]$. To do so, we willl use beaver triple $[\ \boldsymbol{a}\ ], [\ \boldsymbol{b}\ ], [\ \boldsymbol{a} \cdot \boldsymbol{b}\ ]$, and a sharing of a random $\boldsymbol{r} \in \mathbb{F}^\ell$, $[\ \boldsymbol{r}\ ]$. First the players will send $[\ \boldsymbol{x} - \boldsymbol{a}\ ], [\ \boldsymbol{y} - \boldsymbol{b}\ ]$. The dealer will reconstructs values $x - a, y - b, (x - a)(y - b)$, and re-shares them using degree $\ell$ polynomials. The players verify that the shares given to them by the dealer are of degree $\ell$, using the degree test protocol. The players will then compute $[\ \boldsymbol{u}\ ] = [\ \boldsymbol{x} - \boldsymbol{a}\ ]_\ell \cdot [\ \boldsymbol{y}\ ] + [\ \boldsymbol{y} - \boldsymbol{b}\ ]_\ell \cdot [\ \boldsymbol{x}\ ] + [\ \boldsymbol{r}\ ]$ and send the shares to the dealer. The dealer re-shares $\boldsymbol{u}$ using a degree $\ell$ polynomial, and the players again test that the degree is no more than $\ell$. Finally, the players will compute $[\ \boldsymbol{x} \cdot \boldsymbol{y}\ ] = [\ \boldsymbol{u}\ ] - [\ \boldsymbol{a} \cdot \boldsymbol{b}\ ] - [\ \boldsymbol{r}\ ]$.

**Routing.** The input is $[\ \boldsymbol{x}\ ]$ and the output should be $[\rho(\boldsymbol{x})]$. The preprocessing functionality generates shares $[\ \boldsymbol{r}\ ], [\ \boldsymbol{r'}\ ]$ such that $\rho(\boldsymbol{r}) = \boldsymbol{r'}$. Then when provided $[\ \boldsymbol{x}\ ]$, the players will send $[\ \boldsymbol{x} + \boldsymbol{r}\ ]$ to the dealer who will reshare $[\rho(\boldsymbol{x} + \boldsymbol{r})]$. The players will then verify that the dealer reshared $\boldsymbol{x} + \boldsymbol{r}$ using a low degree polynomial via the degree test. The players will then compute $[\ \boldsymbol{x}\ ] = [\rho(\boldsymbol{x} + \boldsymbol{r})] - [\ \boldsymbol{r'}\ ]$. Due to a lack of space, we provide the full description in the full version of our paper.

Formally, we prove that our protocol realizes two things. First, we show that the protocol securely realizes $f_C$ *with low expected communication overhead* if less than $t_r$ players are corrupt. Second, we prove that our protocol securely realizes $f_C^{\mathbf{A}}$ (the functionality that allows the adversary to tamper with each wire individually) if less than $t_p$ players are corrupt. Then, by running our protocol on a circuit secure against tampering on individual wires, our protocol securely realizes $f_C$. The compilers of [11, 10] allow us to compile any circuit into an equivalent circuit that is secure against individual tampering with only a constant blowup in circuit size. As a result, it is easy to see that by employing our protocol with the results of [11, 10], we achieve the desired security properties as well as the desired level of efficiency.

The players input sharings $[\ \boldsymbol{x}\ ], [\ \boldsymbol{y}\ ]$, a triple of shares $[\ \boldsymbol{a}\ ]$, $[\ \boldsymbol{b}\ ]$ and $[\ \boldsymbol{c}\ ]$ such that $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$, and sharing $[\ \boldsymbol{r}\ ]$ of a random value $\boldsymbol{r} \in \mathbb{F}^{\ell}$. All of these values are shared using polynomials of degree $t_p + \ell$. The shares $[\ \boldsymbol{r}\ ]$ are used to hide values from the dealer. The global variable $E$, which stores the set of eliminated players, may be updated. If it is updated, all players terminate, discard their shares, and output $\perp$. Otherwise, the players produce shares $[\ \boldsymbol{z}\ ]$ such that $\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{y}$.

**Evaluation**

1. **Players**
   i. $[\ \boldsymbol{d}\ ] \leftarrow [\ \boldsymbol{y}\ ] - [\ \boldsymbol{b}\ ]$
   ii. $[\ \boldsymbol{e}\ ] \leftarrow [\ \boldsymbol{x}\ ] - [\ \boldsymbol{a}\ ]$
   iii. Send $[\ \boldsymbol{d}\ ], [\ \boldsymbol{e}\ ]$ to dealer.
2. **Dealer**
   i. $\boldsymbol{v} \leftarrow \mathsf{recover}([\ \boldsymbol{d}\ ]) \cdot \mathsf{recover}([\ \boldsymbol{e}\ ])$
   ii. $[\ \boldsymbol{\alpha}\ ]_\ell \leftarrow \mathsf{share}_\ell(\boldsymbol{v})$
   iii. $[\ \boldsymbol{\epsilon}\ ]_\ell \leftarrow \mathsf{reshare}_\ell([\ \boldsymbol{e}\ ])$
   iv. $[\ \boldsymbol{\delta}\ ]_\ell \leftarrow \mathsf{reshare}_\ell([\ \boldsymbol{d}\ ])$
3. **Degree Test**
   i. Execute degree-testing with dealer input $[\ \boldsymbol{\alpha}\ ]_\ell$.
   ii. Execute degree-testing with dealer input $[\ \boldsymbol{\delta}\ ]_\ell$.
   iii. Execute degree-testing with dealer input $[\ \boldsymbol{\epsilon}\ ]_\ell$.
   iv. If the degree test outputs $\perp$ in any of these executions, then all players terminate and output $\perp$.
4. **Players**
   i. $[\ \boldsymbol{g_1}\ ] \leftarrow [\ \boldsymbol{\delta}\ ]_\ell \cdot [\ \boldsymbol{x}\ ] + [\ \boldsymbol{\epsilon}\ ]_\ell \cdot [\ \boldsymbol{y}\ ]$
   ii. $[\ \boldsymbol{g}\ ] \leftarrow [\ \boldsymbol{g_1}\ ] - [\ \boldsymbol{\alpha}\ ]_\ell + [\ \boldsymbol{r}\ ]$
   iii. Send $[\ \boldsymbol{g}\ ]$ to dealer.
5. **Dealer**
   i. $[\ \boldsymbol{h}\ ]_\ell \leftarrow \mathsf{reshare}_\ell([\ \boldsymbol{g}\ ])$
6. **Degree Test**
   i. Execute degree-testing with dealer input $[\ \boldsymbol{h}\ ]_\ell$.
   ii. If the degree test outputs $\perp$, then all players terminate and output $\perp$.
7. **Players**
   i. Set $\boldsymbol{O} \leftarrow [\ \boldsymbol{h}\ ]_\ell + [\ \boldsymbol{c}\ ] - [\ \boldsymbol{r}\ ]$
   ii. Output $\boldsymbol{O}$

**Recovery failure subroutine**

1. If $\mathsf{recover}(\cdot)$ ever fails to output a polynomial of degree $t_p + \ell$ that agrees with $\frac{n+t_p}{2} + \ell$ shares, then the dealer broadcasts "failure", sets $E \leftarrow P$, and all players terminate with output $\perp$. The parameters in this subroutine are chosen so that it is guaranteed that either the Berklaml-welch algorithm reconstruct a correct value for the secret or that the $\mathsf{recover}$ operation will output $\perp$.

**Fig. 5.** Multiplication

**Theorem 2.** *For any number of players $n$, privacy threshold $n \geq t_p \geq \frac{n}{2}$, packing parameter $\ell < \frac{n-t_p}{2}$, $\pi_{mpc}$ $(t_p, O(1/|\mathbb{F}|))$-securely realizes $f_C^{\mathbf{A}}$ with abort in the $\mathcal{F}_{pre}$-hybrid model.*

**Theorem 3.** *For any number of players $n$, privacy threshold $n \geq t_p \geq \frac{n}{2}$, robustness threshold $t_r \leq \frac{n-t_p-2\ell}{2}$, $\ell < \frac{n-t_p}{2}$. $\pi_{mpc}$ $(t_p, t_r, O(1/|\mathbb{F}|))$-securely realizes $(f_C, f_C^{\mathbf{A}})$ for arithmetic circuit $C$ with depth $d$ in the $\mathcal{F}_{pre}$-hybrid model, with full security, and expected communication overhead*

$$O\big(|C| \log(|C|) \cdot \frac{n}{\ell} + d^2 \cdot n + \mathsf{poly}(n, s)\big).$$

Due to space constraint, we only provide a short summary of how we prove our main protocol secure. A more complete argument appears in the full version of the paper.

**Security under Honest Dealer.** Since our protocol is in a hybrid-model, the simulator can simulate a run of the preprocessing functionality and store the generated values. This allows the simulator to extract the adversary's inputs. The simulator runs the honest parties with dummy inputs and determines whether the adversary causes the honest dealer to abort, or causes an additive attack. We show that if the adversary sends bad shares to the dealer, then the simulator can determine, based solely on these shares, which of these three things happen: (1) the dealer aborts because he failed to reconstruct a secret, (2) the bad shares can be ignored (which is the case if enough players are honest), or (3) the attack by the adversary can be mapped to an additive attack. We prove the previous statement by using the fact that Shamir secret sharing is a linear error correcting code, and from the following fact about such codes.

Let $(\mathsf{encode}, \mathsf{decode})$ be a linear error-correcting, let $c = \mathsf{encode}(m)$ be an encoding of $m$, and let be $\mu$ be an error vector. By linearity of the error-correcting code, we have that $\mathsf{decode}(\mathsf{encode}(m) + \mu) = \mathsf{decode}(\mathsf{encode}(m)) + \mathsf{decode}(\mu)$. In particular, this implies that $\mathsf{decode}(\mathsf{encode}(m) + \mu) = \perp$ if and only if $\mathsf{decode}(\mu) = \perp$. The error vector $\mu$ in this case represents the difference between the shares the adversary should have sent, versus the shares it actually sent.

**Security under Malicious Dealer.** At a high level, the simulation of a malicious dealer is similar to that of an honest dealer. The main difference is that this simulator must ensure that the dealer does not share a polynomial of too high a degree. This is easily detected by inspecting the shares sent to the degree-test functionality, and the dealer can then be replaced. If the dealer's polynomial is of the appropriate degree, the simulator can compute the value of an additive attack by reconstructing the shared secret and comparing it with the secret that the dealer should have sent.

## 5 Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. #1564088, #1111599, #1514261, #1652259 and #1563722. Daniel Genkin was also supported by financial assistance award 70NANB15H328 from the U.S. Department of Commerce, NIST, the 2017-2018 Rothschild Postdoctoral Fellowship, and DARPA Contract #FA8650-16-C-7622.

## References


1. D. Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO 91, Proceedings*, pages 420–432, 1991.
2. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology - EUROCRYPT 2010, Proceedings*, pages 445–465, 2010.
3. I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology - CRYPTO 2008, Proceedings*, pages 241–261, 2008.
4. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In *Computer Security - ESORICS 2013, Proceedings*, pages 1–18, 2013.
5. I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, Proceedings*, pages 572–590, 2007.
6. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 Proceedings*, pages 643–662, 2012.
7. I. Damgård and S. Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *Theory of Cryptography*, pages 621–641. Springer, 2013.
8. R. Dowsley, J. Müller-Quade, A. Otsuka, G. Hanaoka, H. Imai, and A. C. A. Nascimento. Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. *IEICE Transactions*, 94-A(2):725–734, 2011.
9. M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992*, pages 699–710, 1992.
10. D. Genkin, Y. Ishai, and A. Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In *Advances in Cryptology - CRYPTO 2015, Proceedings, Part II*, pages 721–741, 2015.
11. D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai, and E. Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Symposium on Theory of Computing, STOC 2014*, pages 495–504, 2014.
12. C. Hazay, E. Orsini, P. Scholl, and E. Soria-Vazquez. Efficient MPC from syndrome decoding (or: Honey, I shrunk the keys). *IACR Cryptology ePrint Archive*, 2018:208, 2018.
13. Y. Ishai, J. Katz, E. Kushilevitz, Y. Lindell, and E. Petrank. On achieving the "best of both worlds" in secure multiparty computation. *SIAM J. Comput.*, 40(1):122–141, 2011.
14. Y. Ishai, E. Kushilevitz, M. Prabhakaran, A. Sahai, and C. Yu. Secure protocol transformations. In *Advances in Cryptology - CRYPTO 2016 Proceedings, Part II*, pages 430–458, 2016.
15. J. Katz. On achieving the "best of both worlds" in secure multiparty computation. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, 2007*, pages 11–20, 2007.
16. J. B. Nielsen and S. Ranellucci. On the computational overhead of MPC with dishonest majority. In *Public-Key Cryptography - PKC 2017, Proceedings, Part II*, pages 369–395, 2017.