Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest*

Jun Furukawa[1] and Yehuda Lindell[2]

[1] NEC Israel Research Center
Jun.Furukawa@necam.com
[2] Bar-Ilan University and Unbound Tech Ltd., Israel**
lindell@biu.ac.il

Abstract. Secure multiparty computation (MPC) enables a set of parties to securely carry out a joint computation of their private inputs without revealing anything but the output. Protocols for semi-honest adversaries guarantee security as long as the corrupted parties run the specified protocol and ensure that nothing is leaked in the transcript. In contrast, protocols for malicious adversaries guarantee security in the presence of arbitrary adversaries who can run any attack strategy. Security for malicious adversaries is typically what is needed in practice (and is always preferred), but comes at a significant cost.

In this paper, we present the first protocol for a two-thirds honest majority that achieves security in the presence of malicious adversaries *at essentially the exact same cost as the best known protocols for semi-honest adversaries*. Our construction is not a general transformation and thus it is possible that better semi-honest protocols will be constructed which do not support our transformation. Nevertheless, for the current state-of-the-art for many parties (based on Shamir sharing), our protocol invokes the best semi-honest multiplication protocol *exactly once* per multiplication gate (plus some additional local computation that is negligible to the overall cost). Concretely, the best version of our protocol requires each party to send on average of just $2\frac{2}{3}$ elements per multiplication gate (when the number of multiplication gates is at least the number of parties). This is four times faster than the previous-best protocol of Barak et al. (ACM CCS 2018) for small fields, and twice as fast as the previous-best protocol of Chida et al. (CRYPTO 2018) for large fields.

1 Introduction

1.1 Background

Protocols for secure computation enable a set of parties with private inputs to compute a joint function of their inputs while revealing nothing but the output.

* This paper appeared at *ACM CCS 2019*.

** Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by the Alter Family Foundation.

The security properties typically required from secure computation protocols include *privacy* (meaning that nothing but the output is revealed), *correctness* (meaning that the output is correctly computed), *independence of inputs* (meaning that a party cannot choose its input as a function of the other parties' inputs), *fairness* (meaning that if one party gets output then so do all), and *guaranteed output delivery* (meaning that all parties always receive output). Formally, the security of a protocol is proven by showing that it behaves like an ideal execution with an incorruptible trusted party who computes the function for the parties [5,16]. In some cases, fairness and guaranteed output delivery are not required. This is standard in the case of no honest majority (since not all functions can be computed fairly without an honest majority), but can also be the case otherwise in order to aid the construction of highly efficient protocols (e.g., as in [1,19]).

Protocols for secure computation must remain secure in the face of adversarial behavior. There are many parameters determining the adversary. Three that are of relevance to this paper are:

- **Adversarial behavior:** If the adversary is *semi-honest*, then it follows the protocol specification but may try to learn more than is allowed by inspecting the protocol transcript. If the adversary is *malicious*, then it may follow an arbitrary attack strategy in its attempt to break the protocol.
- **Adversarial power:** If the protocol is guaranteed to remain secure even if the adversary is computationally unlimited, then the protocol is said to be *information-theoretically secure*. If the adversary is bounded to probabilistic polynomial-time, then the protocol is *computationally secure*.
- **Number of corruptions:** Denote by t the number of corrupted parties and by n the overall number of parties. There are typically three main thresholds that are considered in the literature: $t < n$ (meaning any number of parties may be corrupted), $t < n/2$ (meaning that there is an honest majority), and $t < n/3$ (meaning that less than a third of the parties are corrupted).

Feasibility. In the late 1980s, it was shown that any function can be securely computed. This was demonstrated in the computational setting for any $t < n$ [25,14], in the information-theoretic setting with $t < n/3$ [4,7], and in the information-theoretic setting with $t < n/2$ assuming a broadcast channel [23]. These feasibility results demonstrate that secure computation is possible. However, significant improvements are necessary to make it efficient enough to use in practice.

Efficiency. In the past decade, there has been a large amount of work that has focused on making MPC efficient and practical. This has included work on both the dishonest majority (and two-party) setting, and the setting of an honest majority. The current state-of-the-art is that MPC is now practical for many practical problems of interest, and progress is fast, making it possible to continually expand the domain of practical applications. We discuss more about these works in Section 1.3.

1.2 Our Results

In this paper, we consider the setting of a two-thirds honest majority. We present the first protocol for the malicious setting which is concretely as efficient as the best known protocol for semi-honest. Specifically, our protocol works by running a semi-honest MPC, and then verifying that no cheating took place before revealing any output. Our verification method is novel, and has very low communication cost (in particular, it is independent of the circuit size). Our method is not generic, in the sense that it does not work for any semi-honest protocol. Rather, we use a semi-honest secret-sharing based multiplication protocol that maintains privacy in the presence of malicious adversaries, but not necessarily correctness. Formally, we prove our method in general for multiplication protocols that are secure in the presence of *additive attacks*, as formalized and used in [12,13,19,8]. This essentially means that the only thing that an attacker can do in the multiplication protocol is to make the output of the multiplication of x and y be a (valid) sharing of $x \cdot y + d$, instead of $x \cdot y$. However, unlike the previous best work that required *two* invocations of the multiplication protocol per multiplication gate [8], we require only *one*. It is important to note that the standard multiplication protocols for the honest-majority information-theoretic setting are secure up to additive attacks, including [4,15,10]. We highlight [10] in particular, since this is the fastest semi-honest multiplication known. We therefore prove our primary protocol using the base protocol of [10] to obtain a protocol that is secure for malicious adversaries at essentially the same cost as the best semi-honest protocol of [10].

The idea behind our novel verification method is as follows. In general, there are two attacks that an adversary may carry out. First, it may deal shares that are supposed to be of degree-t but are actually not. Second, it may send incorrect values that make the result of the multiplication gate be incorrect. For both cases, we show that a simple linear check can be used, that requires communication that is independent of the circuit size, and is concretely very efficient. In particular, for the degree-t check, it suffices to verify that a random linear combination of all dealed shares is a degree-t polynomial. Furthermore, for the correctness of multiplication, it suffices to verify that in each multiplication gate with inputs x, y and output z the equation $z - x \cdot y$ equals 0. This can be done efficiently by checking that a random linear combination of all of these equations equals 0. Although such ideas have been used before, their application in this way is novel and yields a protocol that is much more efficient than previous works.

Concretely, we present two variants of the protocol (where these version relate to optimized instantiations of the multiplication protocol of [10]). The first variant is information-theoretic and has an amortized cost of each party sending $4\frac{2}{3}$ field elements per multiplication gate. The second variant is computational and has an amortized cost of each party sending $2\frac{2}{3}$ field elements per multiplication. For typical parameters of number of parties and field size (even for small fields like $GF[2^8]$), the additive cost of the verification procedure is just each party sending $6\frac{2}{3}$ field elements overall.

We remark that our protocol is extremely simple. This is advantageous for efficiency, as well as for security (since verifying that a simple protocol is correctly implemented is easier than for a complex one).

Implementation. We implemented our protocol (and will make it open source upon publication), and ran extensive experiments to compare the different variants and to compare it do prior work. Our experiments show that the cost of the verification is small relative to the rest of the protocol, adding only a few percent to the running time (this percentage is higher for a small number of parties since in this case the overall cost is so low becomes less significant with respect to the rest of the protocol). Our protocol also significantly outperforms the previous best protocols. See Section 7 for more details on these experiments.

Future work. The focus of this paper is practical efficiency. However, the techniques used here seem to be generalizable to arbitrary linear secret-sharing schemes, and semi-honest multiplication protocols with weaker properties than security up to additive attacks. In particular, it seems to suffice that the protocol provides privacy against a malicious adversary before output is revealed, and if a simulator can detect when any cheating took place. We leave these generalizations for future work. (These make no difference today since, as mentioned, the protocol of [10] is the fastest we have today in any case, and it achieves security up to additive attacks.)

1.3 Comparison to Prior Work

The goal of obtaining MPC for an honest majority with *constant communication* – meaning that each party sends an amortized constant number of field elements per multiplication gate – has been well studied, culminating in the breakthrough result of [3]. The protocol of [3] was optimized in [2], reducing the constants to about a quarter of their original cost, but at the cost of not achieving guaranteed output delivery. The above works both achieve perfect security, and the cost is independent of the field size. Concretely, in [2], each party communicates an average of 13 field elements per multiplication gate. Thus, our information-theoretic protocol is about 1/3 of the cost of [2], and our computationally-secure protocol is about 1/5 of the cost of [2]. In practice, this makes a significant difference, as we show in our experiments in Section 7.2.

For the case of $t < n/2$, the result of [8] achieves security for malicious adversaries at the cost of *twice* that of semi-honest (to be more exact, they require a multiplication protocol that is secure up to additive attacks). This protocol is statistical, with an error of $1/|\mathbb{F}|$ and thus for small fields, it must be repeated and the cost increases. In contrast, we only need to repeat the verification for small fields, and this only incurs a small cost. Thus, our protocol is twice as fast as [8] for large fields, and even faster for smaller fields. However, unlike our protocol, [8] achieve a higher corruption threshold of $t < n/2$ rather than $t < n/3$, which is of course an important advantage.

There is a large body of work that has focused on achieving efficient multiparty computation, for the setting of two parties specifically, and a dishonest majority in general. See [11] for an excellent survey of techniques and works in this direction.

2 Preliminaries and Definitions

2.1 Preliminaries

Notation. Let $P_1, ..., P_n$ denote the n parties participating in the computation, and let t denote the number of corrupted parties. In this work, we assume that $t < \frac{n}{3}$. Throughout the paper, we use I to denote the set of corrupted parties and $J = [n] \setminus I$ to denote the set of honest parties. We denote by $\mathbb{F}$ a finite field and by $|\mathbb{F}|$ its size. We denote statistical closeness by $\overset{s}{\equiv}$.

Definitions of security. We use the standard definition of security based on the ideal/real model paradigm [5,16], with security formalized for non-unanimous 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. We distinguish between perfect, statistical and computational security, where relevant. When considering statistical security, we refer to a parameter s and require that the statistical distance be at most 2^{-s}. When considering computational security, we refer to a security parameter κ.

Secret sharing. Although our work can be generalized to essentially any linear secret sharing, we present it specifically for Shamir secret sharing, for clarity. As such, we consider Shamir secret sharing [24] with $\mathbb{F} > n$. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. We have the following procedures:

- share(v): In this procedure, a dealer with a value $v \in \mathbb{F}$, chooses random $a_1, \ldots, a_t \in \mathbb{F}$ and defines $p(x) = v + \sum_{i=1}^{t} a_i \cdot x^i$. The dealer then sends $p(\alpha_j)$ to party P_j, for $j = 1, \ldots, n$. We denote by $[v]$ the case whereby all parties hold a sharing of v. We will sometimes consider sharings via degree-t polynomials, and sometimes via degree-$2t$ polynomials. When this distinction is needed, we will denote such sharings by $[v]_t$ and $[v]_{2t}$, respectively.
  We stress that if the dealer is corrupted, then the shares received by the parties may not be correct in that the dealer may define a polynomial of degree greater than t. In such a case, we call the sharing invalid; else we call it valid. We remark that we will abuse notation and say that the parties hold shares $[v]$ even if these are invalid.
- complete$(v_{i_1}, \ldots, v_{i_{t+1}})$: Given $t+1$ shares $v_{i_1}, \ldots, v_{i_{t+1}}$, this procedure interpolates to find the unique polynomial p passing through all these points, and $v = p(0)$. Then, complete outputs the remaining $n-t-1$ shares defined by p.

– reconstruct$([v]_t, i)$: Given a sharing $[v]_t$ and an index i held by the parties, all parties send their shares of $[v]_t$ to P_i. Party P_i verifies that all points lie on the same degree-t polynomial, and if yes outputs it. As long as more than t parties are honest, this interactive protocol guarantees that if $[v]_t$ is not correct (see formal definition below), then P_i will output $\perp$ and abort. Otherwise, if $[v]_t$ is correct, then P_i will either output v or will abort.
– open$([v]_t)$ or open$([v]_{2t})$: This procedure is the same as reconstruct, except that all parties receive the shares. Thus, naively, once can define open$([v]_t)$ as the execution of reconstruct$([v]_t, i)$ for every $i = 1, \ldots, n$. This is expensive since each party sends n elements overall. As shown in [3], in the ReconsPub procedure, it is possible to run open in parallel on $n - t$ sharings of degree-t at the same cost. Thus, for $t < n/3$ each party sends n elements in order to open $2n/3$ degree-t sharings, at an average cost of 1.5 elements per opening. Likewise, each party sends n elements in order to open $n/3$ degree-$2t$ sharings, at an average cost of 3 elements per opening. It is possible to run open on sharings of degree-$2t$ since there are $2t+1$ honest parties in our case, and thus it is possible to detect (but not correct) any cheating.
– *Local operations:* We denote by $[x]+[y]$ and $c \cdot [x]$ the local operation of each party adding its share in x with its share in y, and multiplying its share in x with a scalar c, respectively. These operations always result in valid sharings of the result, since they are local operations only, and these can be carried out on degree-t and degree-$2t$ sharings.
  We also denote by $[x] \cdot [y]$ the local operation of a party multiplying its share in x with its share in y. Note that in this case, the result is not a valid sharing of the same degree. However, it does hold that $[x]_t \cdot [y]_t = [x \cdot y]_{2t}$.

2.2 Definition of Security

The security parameter is denoted κ; negligible functions and computational indistinguishability are defined in the standard way, with respect to non-uniform polynomial-time distinguishers.

Ideal versus real model definition. We use the ideal/real simulation paradigm in order to define security, where an execution in the real world is compared to an execution in an ideal world where an incorruptible trusted party computes the functionality for the parties [5,16]. We define *security with abort* (and without fairness), meaning that the corrupted parties may receive output while the honest parties do not. Our basic definition does *not* guarantee *unanimous abort*, meaning that some honest party may receive output while the other does not. It is easy to modify our protocols so that the honest parties unanimously abort by running a single (weak) Byzantine agreement at the end of the execution [17]; we therefore omit this step for simplicity.

As we describe at the end of Section 6, our protocol is easily extended to guarantee fairness. The basic definition can be modified to include fairness, as will be described below.

The real model. In the real model, a n-party protocol π is executed by the parties. For simplicity, we consider a synchronous network that proceeds in rounds and a rushing adversary, meaning that the adversary receives its incoming messages in a round before it sends its outgoing message.[3] The adversary $\mathcal{A}$ can be malicious; it sends all messages in place of the corrupted parties, and can follow any arbitrary strategy. The honest parties follow the instructions of the protocol.

Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary controlling $t < \frac{n}{3}$ parties. Let $\text{REAL}_{\pi,\mathcal{A}(z),I}(x_1, \ldots, x_n, \kappa)$ denote the output of the honest parties and $\mathcal{A}$ in an real execution of π, with inputs $x_1, \ldots, x_n$, auxiliary-input z for $\mathcal{A}$, and security parameter κ.

The ideal model. We define the ideal model, for any (possibly reactive) functionality $\mathcal{F}$, receiving inputs from $P_1, \ldots, P_n$ and providing them outputs. Let $I \subset \{1, \ldots, n\}$ be the set of indices of the corrupted parties controlled by the adversary. The ideal execution proceeds as follows:

- **Send inputs to the trusted party:** Each honest party P_j sends its specified input x_j to the trusted party. A corrupted party P_i controlled by the adversary may either send its specified input x_i, some other x'_i or an abort message.
- **Early abort option:** If the trusted party received abort from the adversary $\mathcal{A}$, it sends $\perp$ to all parties and terminates. Otherwise, it proceeds to the next step.
- **Trusted party sends output to the adversary:** The trusted party computes each party's output as specified by the functionality $\mathcal{F}$ based on the inputs received; denote the output of P_j by y_j. The trusted party then sends $\{y_i\}_{i \in I}$ to the corrupted parties.
- **Adversary instructs trusted party to continue or halt:** For each $j \in \{1, \ldots, n\}$ with $j \notin I$, the adversary sends the trusted party either abort_j or continue_j. For each $j \notin I$:
  - If the trusted party received abort_j then it sends P_j the abort value $\perp$ for output.
  - If the trusted party received continue_j then it sends P_j its output value y_j.
- **Outputs:** The honest parties always output the output value they obtained from the trusted party, and the corrupted parties outputs nothing.

Let $\mathcal{S}$ be a non-uniform probabilistic polynomial-time adversary controlling parties P_i for $i \in I$. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S}(z),I}(x_1, \ldots, x_n, \kappa)$ denote the output of the honest parties and $\mathcal{S}$ in an ideal execution with the functionality $\mathcal{F}$, inputs $x_1, \ldots, x_n$ to the parties, auxiliary-input z to $\mathcal{S}$, and security parameter κ.

[3] This modeling is only for simplicity, since in our protocol, all parties receive and send messages in each round. Thus, by instructing each party to only send their round $i+1$ messages after receiving all round-i messages, we have that an execution of the protocol in an asynchronous network is the same as for a rushing adversary in a synchronous network. Note that we do not guarantee output delivery, so "hanging" of the protocol is also allowed.

Security. Informally speaking, the definition says that protocol π securely computes f if adversaries in the ideal world can simulate executions of the real world protocol. In some of our protocols there is a statistical error that is not dependent on the computational security parameter. As in [20], we formalize security in this model by saying that the distinguisher can distinguish with probability at most this error *plus* some factor that is negligible in the security parameter. This is formally different from the standard definition of security since the statistical error does not decrease as the security parameter increases.

Definition 2.1. *Let $\mathcal{F}$ be a n-party functionality, and let π be a n-party protocol. We say that π* securely computes *f* with abort *in the presence of an adversary controlling $t < \frac{n}{3}$ parties, if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ in the real world, there exists a non-uniform probabilistic polynomial-time simulator/adversary $\mathcal{S}$ in the ideal model with $\mathcal{F}$, such that for every $I \subset \{1, \ldots, n\}$ with $|I| < \frac{n}{3}$,*

$$\left\{\text{IDEAL}_{F,\mathcal{S}(z),I}(x_1, \ldots, x_n, \kappa)\right\} \stackrel{\text{c}}{\equiv} \left\{\text{REAL}_{\pi,\mathcal{A}(z),I}(x_1, \ldots, x_n, \kappa)\right\}$$

where $x_1, \ldots, x_n \in \mathbb{F}^*$ under the constraint that $|x_1| = \cdots = |x_n|$, $z \in \mathbb{F}^*$ and $\kappa \in \mathbb{N}$. We say that π securely computes f with abort in the presence of an adversary controlling $t < \frac{n}{3}$ parties with statistical error *$2^{-\sigma}$ if there exists a negligible function $\mu(\cdot)$ such that the distinguishing probability of the adversary is less than $2^{-\sigma} + \mu(\kappa)$.* □

Fairness. The above definition can be modified so that fairness is guaranteed by merely modifying the ideal model so that after the "early abort option" the trusted party simply sends each party its output. That is, if there is no early abort, then *all* parties receive output. This is the only modification required to the definition.

The hybrid model. We prove the security of our protocols in a hybrid model, where parties run a protocol with real messages and also have access to a trusted party computing a subfunctionality for them. The modular sequential composition theorem of [5] states that one can replace the trusted party computing the subfunctionality with a real secure protocol computing the subfunctionality. When the subfunctionality is g, we say that the protocol works in the *g-hybrid model*.

Universal Composability [6]. Protocols that are proven secure in the universal composability framework have the property that they maintain their security when run in parallel and concurrently with other secure and insecure protocols. In [18, Theorem 1.5], it was shown that any protocol that is proven secure with a black-box non-rewinding simulator and also has the property that the inputs of all parties are fixed before the execution begins (called input availability or start synchronization in [18]), is also secure under universal composability. Since the input availability property holds for all of our protocols and subprotocols, it is

sufficient to prove security in the classic stand-alone setting and automatically derive universal composability from [18]. We remark that this also enables us to call the protocol and subprotocols that we use in parallel and concurrently (and not just sequentially), enabling us to achieve more efficient computation (e.g., by running many executions in parallel or running each layer of a circuit in parallel).

3 Building Blocks and Sub-Protocols

In this section, we define a series of building blocks that we need for our protocol. Most of these are used in previous works, like [3,19,2,8]. Our presentation is similar, with some modifications where possible due to us working in the scenario of $t < n/3$ (rather than $t < n/2$ like in [19,8]).

3.1 Generating Random Shares and Coins

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 3.1. 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 3.1 ($\mathcal{F}_{\text{rand}}$ – Generating Random Shares)

Upon receiving r_i for each corrupted party P_i with $i \in I$ from the ideal adversary $\mathcal{S}$, the ideal functionality $\mathcal{F}_{\text{rand}}$ chooses a random $r \in \mathbb{F}$ and generates a sharing $[r]_t$ under the constraint that the share of P_i is r_i for every $P_i \in I$. Then, $\mathcal{F}_{\text{rand}}$ sends each honest party P_j its share in $[r]_t$.

The functionality $\mathcal{F}_{\text{rand}}^{\text{double}}$ is defined similarly to $\mathcal{F}_{\text{rand}}$, but generates *double sharings* that are defined to be two sharings of the same random value r, but where one is of degree-t and the other of degree-$2t$.

FUNCTIONALITY 3.2 ($\mathcal{F}_{\text{rand}}^{\text{double}}$ – Random Double Sharings)

Upon receiving r_i, r_i' for each corrupted party P_i with $i \in I$ from the ideal adversary $\mathcal{S}$, the ideal functionality $\mathcal{F}_{\text{rand}}^{\text{double}}$ chooses a random $r \in \mathbb{F}$, and generates sharings $[r]_t$ and $[r]_{2t}$ under the constraint that the shares of P_i in $[r]_t$ and $[r]_{2t}$ are r_i and r_i', respectively, for every $i \in I$. Then, $\mathcal{F}_{\text{rand}}^{\text{double}}$ sends each honest party P_j its shares in $[r]_t$ and $[r]_{2t}$.

Method and complexity: We use the method called DoubleShareRandom from [2] which is based on [3] in order to generate double random sharings with *perfect security* in the presence of malicious adversaries where $t < n/3$. This protocol generates $n - 2t$ double-random sharings at the cost of $2n + 2(n - 2t)$ elements sent by each party. For $t < n/3$, this generates $n/3$ sharings at the average cost of $\frac{2n+2n/3}{n/3} = 8$ elements per party per double-sharing generated. Although single random shares, as in $\mathcal{F}_{\text{rand}}$, can be generated more efficiently than double random sharings, we only need a few of these. Thus, we will only use $\mathcal{F}_{\text{rand}}^{\text{double}}$, and will discard the $2t$-degree share where not needed. This is the most efficient since $\mathcal{F}_{\text{rand}}^{\text{double}}$ generates many random double sharings at once, and for typical parameters one call to $\mathcal{F}_{\text{rand}}^{\text{double}}$ is enough for the entire protocol.

Generating random coins. $\mathcal{F}_{\text{coin}}$ is an ideal functionality that chooses a random element from $\mathbb{F}$ and hands it to all parties. The simplest way to compute $\mathcal{F}_{\text{coin}}$ securely (in the sense of using existing building blocks) is to use $\mathcal{F}_{\text{rand}}$ to generate a random sharing and then open it. The security of this protocol is immediate, and the cost of the protocol is one call to $\mathcal{F}_{\text{rand}}$ and one execution of open (the latter which costs sending n elements per party).

3.2 Checking Equality to 0

In our protocol, we will need to check whether a given sharing is a sharing of the value 0, without revealing any further information on the shared value. For this purpose, we use a variant of the protocol presented in [8]. The idea behind the protocol is simple. Holding a sharing $[v]$, the parties generate a random sharing $[r]$ and multiply it with $[v]$ (using local multiplication of their shares). Then, the parties open the obtained sharing and check equality to 0. This works since if $v = 0$, then multiplying it with a random r will still yield 0. In contrast, if $v \neq 0$, then the multiplication will result with 0 only when $r = 0$, which happens with probability $\frac{1}{|\mathbb{F}|}$. By repeating sufficiently many times, this probability of error can be made negligible. However, in order to ensure that nothing is revealed by the opening, we need to rerandomize the sharing of $r \cdot v$ by adding a random degree-$2t$ sharing of 0. This is easy to achieve by constructing a double random sharing $[\rho]_t$ and $[\rho]_{2t}$, opening the degree-t sharing to obtain ρ and then computing $[0]_{2t} = [\rho]_{2t} - \rho$. In order to reduce the number of rounds required, we open $[\rho]_t$ together with $[r \cdot v + \rho]_{2t}$ and just verify that the values are equal to each other; this is equivalent to $r \cdot v$ being equal to 0.

The zero-checking protocol that we present here is a bit more efficient than that of [8], since here we have a single opening of a share. In contrast, the protocol for checking equality to 0 in [8] first runs a multiplication protocol involving opening and then has another opening.

The ideal functionality $\mathcal{F}_{\text{checkZero}}$ for checking equality to 0 is formally defined in Functionality 3.3.

FUNCTIONALITY 3.3 ($\mathcal{F}_{\text{checkZero}}$ – Checking Equality to 0)

The ideal functionality $\mathcal{F}_{\text{checkZero}}$ receives (valid) shares of $[v]_t$ from the honest parties P_j for every $j \in J$, and uses them to compute v and the shares of the corrupted parties in $[v]_t$ using the complete procedure.
$\mathcal{F}_{\text{checkZero}}$ sends the ideal adversary $\mathcal{S}$ the corrupted parties' shares in $[v]_t$. In addition:

1. If $v = 0$, then $\mathcal{F}_{\text{checkZero}}$ sends accept to the ideal adversary $\mathcal{S}$. If $\mathcal{S}$ sends reject (resp., accept), then $\mathcal{F}_{\text{checkZero}}$ sends reject (resp., accept) to the honest parties.
2. If $v \neq 0$, then $\mathcal{F}_{\text{checkZero}}$ sends reject to $\mathcal{S}$ and the honest parties.

Observe that $\mathcal{F}_{\text{checkZero}}$ sends the corrupted parties their shares as output. This is needed to enable the simulation (since the simulator needs to know these shares), and has no effect on security since these shares are anyway already known to the adversary. The exact protocol for securely computing $\mathcal{F}_{\text{checkZero}}$ is specified in Protocol 3.4.

PROTOCOL 3.4 (Securely computing $\mathcal{F}_{\text{checkZero}}$ for $t < n/3$)

- **Input:** The parties hold a sharing $[v]_t$.
- **Parameters:** Let s be the security parameter, and let δ be the smallest value such that $|\mathbb{F}|^\delta > 2^s$.
- **The protocol:** Repeat the following δ times (in parallel):
  1. The parties call $\mathcal{F}_{\text{rand}}$ to obtain a sharing $[r]_t$, and call $\mathcal{F}_{\text{rand}}^{\text{double}}$ to obtain a double sharing $[\rho]_t, [\rho]_{2t}$.
  2. The parties locally compute $[r \cdot v + \rho]_{2t} = [r]_t \cdot [v]_t + [\rho]_{2t}$.
  3. The parties run open$([r \cdot v + \rho]_{2t})$ and open$([\rho]_t)$ in parallel. If a party receives $\perp$, then it outputs $\perp$. Else, it continues.
- **Output:** If in *every* repetition, the values opened are equal to each other (i.e., the same ρ in the repetition), then output accept; else output reject.

Complexity: The cost is δ calls to $\mathcal{F}_{\text{rand}}$ and 2δ openings. Using the amortized opening operation on degree-$2t$ and degree-t sharings described in Section 2.1, $n/3$ elements can be opened at the cost of sending n elements. Likewise, $n/3$ (double) random sharings can be generated at the cost of less than $3n$ elements. Thus, as long as $2\delta \approx 2 \cdot \frac{s}{\log|\mathbb{F}|} < \frac{n}{3}$, the overall cost of $\mathcal{F}_{\text{checkZero}}$ is $4n$ elements sent overall (4 by each party). This holds for most reasonable choices of parameters. In fact, for most reasonable parameters, the number of random sharings needed is very small, and so adds very little in practice.

Theorem 3.1. *Protocol 3.4 securely computes $\mathcal{F}_{\text{checkZero}}$ in the $(\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{rand}}^{\text{double}})$-hybrid model with statistical security, for $t < n/3$.*

Proof: Let $\mathcal{A}$ be an adversary; we construct a simulator $\mathcal{S}$ as follows. $\mathcal{S}$ receives the output accept/reject as well as the shares of the corrupted parties in $[v]_t$ from the ideal functionality $\mathcal{F}_{\text{checkZero}}$. Denote the share of the ith party by v_i. $\mathcal{S}$ works as follows:

1. *Simulate δ calls to $\mathcal{F}_{\text{rand}}$:* For each call $\mathcal{S}$ plays the ideal functionalities $\mathcal{F}_{\text{rand}}$ and $\mathcal{F}_{\text{rand}}^{\text{double}}$ in the protocol, and receives from $\mathcal{A}$ values α_i and α'_i, β'_i as P_i's input to $\mathcal{F}_{\text{rand}}$ and $\mathcal{F}_{\text{rand}}^{\text{double}}$, respectively, for each $i \in I$. For $\mathcal{F}_{\text{rand}}^{\text{double}}$, $\mathcal{S}$ chooses a random $\rho \in \mathbb{F}$ and defines sharings $[\rho]_t, [\rho]_{2t}$ passing through α'_i, β'_i, respectively, as specified in $\mathcal{F}_{\text{rand}}^{\text{double}}$.
2. *Simulate δ openings:* For each of the δ times,
   (a) If the output was accept, then $\mathcal{S}$ chooses a random degree-$2t$ polynomial p with $p(0) = \rho$ that passes through the points defined by the shares $\{v_i \cdot \alpha_i + \beta'_i\}_{i \in I}$, where v_i is P_i's share in $[v]_t$ as received from $\mathcal{F}_{\text{checkZero}}$, and α_i, β'_i are as received from $\mathcal{A}$ above. Then, $\mathcal{S}$ simulates the honest parties sending their shares in p and in $[\rho]_t$ to the corrupted parties in the executions of open.
   If $\mathcal{A}$ sends any incorrect shares in the openings, then $\mathcal{S}$ sends reject to $\mathcal{F}_{\text{checkZero}}$; else it sends accept to $\mathcal{F}_{\text{checkZero}}$.
   (b) If the output was reject, then $\mathcal{S}$ chooses a random $r' \in \mathbb{F}$ and a random degree-$2t$ polynomial p with $p(0) = r'$ that passes through the points defined by the shares $\{v_i \cdot \alpha_i + \beta'_i\}_{i \in I}$, where v_i and α_i, β'_i are as above. Then, $\mathcal{S}$ simulates the honest parties sending their shares in p and in $[\rho]_t$ to the corrupted parties in the executions of open.

We argue that the distribution over the output is statistically close to a real execution. Let bad be the event that $v \neq 0$ and all r' values chosen by $\mathcal{S}$ equal 0. We consider a hybrid experiment with an ideal functionality that is the same as $\mathcal{F}_{\text{checkZero}}$ except that even if $v \neq 0$, the ideal simulator can send accept to the functionality, and all honest parties will accept. We denote this hybrid experiment by $\mathcal{H}$ and the simulator for this experiment by $\mathcal{S}_H$. Simulator $\mathcal{S}_H$ works identically to $\mathcal{S}$, except that if the event bad occurs, then it sends accept to the ideal functionality. Since bad happens with probability $|\mathbb{F}|^{-\delta}$, and otherwise everything is the same, we have that for every distinguisher D,

$$\left|\Pr\left[D\left(\text{IDEAL}_{\mathcal{F}_{\text{checkZero}},\mathcal{S}}([v]_t)\right) = 1\right] - \Pr\left[D\left(\mathcal{H}_{\mathcal{S}_H}([v]_t)\right) = 1\right]\right| \leq \frac{1}{|\mathbb{F}|^{\delta}}. \tag{1}$$

It remains to show that

$$\Pr\left[D\left(\mathcal{H}_{\mathcal{S}_H}([v]_t)\right) = 1\right] = \Pr\left[D\left(\text{HYBRID}_{\Pi,\mathcal{A}}^{\mathcal{F}_{\text{rand}},\mathcal{F}_{\text{rand}}^{\text{double}}}([v]_t)\right) = 1\right], \tag{2}$$

where Π denotes Protocol 3.4. Observe that in a real execution, all outputs from $\mathcal{F}_{\text{rand}}$ are uniformly distributed. Thus, when $v \neq 0$, the probability that all r values chosen by $\mathcal{F}_{\text{rand}}$ equal 0 and thus the honest parties accept is exactly the probability that bad occurs in $\mathcal{H}$. Furthermore, setting $r' = v \cdot r$ for every iteration, we have that the distribution over the value r' chosen by S_H and the

value $v \cdot r$ generated in the protocol are identical. Thus, the output of the honest parties is identical in both executions. Finally, observe that the view of $\mathcal{A}$ of the opened values in a real execution and in the execution with $\mathcal{S}_H$ is identical (for all outputs). This follows since in both cases, the shares of the honest parties (that are revealed to the adversary during open) are random shares of ρ if $v = 0$, and are random shares of a uniformly distributed element (r' or $r \cdot v + \rho$) if $v \neq 0$. This is due to the fact that $\mathcal{S}_H$ chooses these at random in $\mathcal{H}$, and due to the fact that the shares from $\mathcal{F}_{\text{rand}}^{\text{double}}$ in the real protocol are random. The theorem is derived by combining Equations (1) and (2). ■

4 Secure Multiplication Verification

In this section, we present two subprotocols for verifying that all values used in the protocol were correct. There are two properties that need to be checked:

1. *Property 1 – that a set of sharings are valid degree-t sharings:* Denoting the shares to be checked by $[x_1]_t, \ldots, [x_M]_t$, we need to verify that each $[x_i]_t$ is such that all of the honest parties' shares lie on the same degree-t polynomial.
2. *Property 2 – that a set of multiplication values are all correct:* Denote a series of multiplication gates by $g_1, \ldots, g_N$, where each g_k is a triple (i_k, j_k, ℓ_k) representing the multiplication gate $w_{\ell_k} = w_{i_k} \cdot w_{j_k}$. Then, we need to verify that there does not exist any k such that $w_{\ell_k} \neq w_{i_k} \cdot w_{j_k}$. (After checking the first property, we know that $[w_{i_k}]_t, [w_{j_k}]_t, [w_{\ell_k}]_t$ are all valid.)

Next, we explain how each property is checked. The basic idea is to generate check-sharings for each of the two cases. The protocol for property 1 is exactly that of [19, Protocol 3.1] (batch correctness check of shares), whereas the protocol for property 2 is novel.

4.1 Verifying that Shares are Degree-*t*

For the sake of completeness, we describe the verification check of [19] for this property. In order to verify the first property described above, that a set of sharings are of degree-t, the parties generate (pseudo)random values $a_1, \ldots, a_M$ and define the sharing $[u]_t = \sum_{k=1}^{M} a_k \cdot [x_k]_t$. Observe that if all of the $[x_k]_t$ are degree-t sharings, then $[u]_t$ is a degree-t sharing. However, if there exists a k such that $[x_k]_t$ is *not* a degree-t sharing, then $[x]_t$ will be a degree-t sharing except with probability $1/|\mathbb{F}|$. This is due to the fact that each sharing is a polynomial and so $[u]_t$ can only be of degree-t if the higher-level terms of other sharings cancel out the higher-level terms of $a_k \cdot [x_k]_t$. Since $a_k \in \mathbb{F}$ is chosen randomly after all shares are fixed, the probability that the sum of all other higher-level terms equals a_k times the higher level terms of $[x_k]_t$ is at most $1/|\mathbb{F}|$ (fix all other values; then there is at most a single value $a_k \in \mathbb{F}$ that can cause the higher level terms to become 0). See Protocol 4.1 for a specification of the protocol.

PROTOCOL 4.1 (Verification that a set of shares are degree-t)

Inputs: The parties hold a series of shares $\{[x_k]_t\}_{k=1}^M$.

Parameter: Let δ be such that $|\mathbb{F}|^\delta \geq 2^s$, where s is the statistical security parameter, and let κ be a computational security parameter for a pseudorandom function F.

The protocol:

1. *Phase 1 – prepare random values:*
   (a) The parties call $\mathcal{F}_{\text{coin}}$ to obtain a single key $K \in \{0,1\}^\kappa$ for a pseudorandom function F. Then, the parties compute $\delta \cdot M$ pseudorandom values $(a_1^1, \ldots, a_M^1), \ldots, (a_1^\delta, \ldots, a_M^\delta)$ by setting $a_k^j = F_K(k \| j)$.
   (b) The parties call $\mathcal{F}_{\text{rand}}$ to obtain sharings $([\rho_1]_t, \ldots, [\rho_\delta]_t)$ of unknown random values $\rho_1, \ldots, \rho_\delta$.
2. *Phase 2 – actual verification:*
   (a) For $j = 1, \ldots, \delta$, the parties locally compute the check polynomial

$$[u_j]_t = \sum_{k=1}^{M} a_k^j \cdot [x_k]_t + [\rho_j]_t.$$

   (b) For $j = 1, \ldots, \delta$, the parties run $\mathsf{open}([u_j]_t)$.
3. *Output:* If a party receives $\perp$ in any opening or if the opened sharing defines a polynomial of degree greater than t (i.e., not all values lie on a single degree-t polynomial), then it outputs reject. Else, it outputs accept.

The following lemma is proven in [19].[4]

Lemma 4.1 (Lemma 3.2 of [19]). *Let n be the number of parties, let $t < n/3$ be the maximum number of corrupted parties controlled by an (unbounded) adversary, and assume that F is a pseudorandom function for non-uniform distinguishers. If there exists a $j \in [M]$ such that the sharing of x_j is not of degree-t, then the honest parties accept in Protocol 4.1 with probability at most $|\mathbb{F}|^{-\delta} \leq 2^{-s}$.*

We remark that Protocol 4.1 is not proven to securely realize an appropriate "check ideal functionality". Nevertheless, it suffices for what we need (we use it in the same way as [19] who take this approach). The fact that this suffices is shown in the proofs of security of the protocols where it is used (Protocols 5.1 and 6.1). In particular, Protocol 4.1 reveals nothing when the input sharing *is* of degree-t. However, it can reveal something if the sharing is *not* of degree-t. In Protocol 5.1, the verification is applied to random shares before they are used;

[4] The actual lemma stated in [19] bounds the cheating by $(|\mathbb{F}|-1)^{-\delta}$ rather than $|\mathbb{F}|^{-\delta}$. However, this is due to the fact that they choose all a_k^j values in $\mathbb{F} \setminus \{0\}$ instead of in $\mathbb{F}$, which is not actually needed. In addition, they prove it for $t < n/2$; we wrote $t < n/3$ simply since this is the setting we are considering here.

thus, there is no need for privacy of incorrect shares. Likewise, in Protocol 4.1, the verification is only applied to the sharing of inputs by the parties. Thus, honest parties' inputs are guaranteed to be private, which is all that is required in our case.

4.2 Verifying Correctness of Multiplication

In this section, we present our protocol for multiplication correctness verification. This protocol is novel and what enables us to achieve high efficiency. The protocol uses similar ideas to previous works, but applies them in a different way to achieve a check with communication complexity that is independent of the size of the circuit. In this section, we assume that all input shares are of degree-t, since this is verified previously using the protocol of Section 4.1.

In order to verify the second property described above, that all multiplication values are correct, the parties generate
pseudo-random values $b_1, \ldots, b_N$ and define the sharing

$$[v_j]_{2t} = \left(\sum_{k=1}^{N} b_k \cdot ([w_{i_k}]_t \cdot [w_{j_k}]_t - [w_{\ell_k}]_t) \right) + [\rho']_{2t},$$

where the k'th gate has input wires w_{i_k}, w_{j_k} and output wire w_{ℓ_k}.[5] Observe that if the multiplication values are correct, then $[v_j]_{2t}$ should be a degree-$2t$ sharing of ρ', since all $(w_{i_k} \cdot w_{j_k} - w_{\ell_k})$ equal 0. However, if there exists some k for which $w_{i_k} \cdot w_{j_k} \neq w_{\ell_k}$, then $w_{i_k} \cdot w_{j_k} - w_{\ell_k} \neq 0$ and $[v_j]_{2t}$ is a sharing of the value ρ' with probability at most $1/|\mathbb{F}|$. This is for the same reason as described above: since $b_k \in \mathbb{F}$ is random it follows that $b_k \cdot (w_{i_k} \cdot w_{j_k} - w_{\ell_k})$ is random, and thus it cancels out with the rest of the check sharing with probability at most $1/|\mathbb{F}|$.

As such, the verification is carried out by the parties generating double-random sharings securely using $\mathcal{F}_{\text{rand}}^{\text{double}}$ to obtain $[\rho']_t$ and $[\rho']_{2t}$. The check-sharing uses $[\rho']_{2t}$ and the parties then open the check-sharing and $[\rho']_t$ and verify that they are to the same value.

A crucial point in this verification step is that since all sharings have already been validated to be of degree-t, the local operation to generate the check-sharing (involving parties locally multiplying their sharings of w_{i_k} and w_{j_k}) defines a degree-$2t$ polynomial. As such, this polynomial is fully masked by $[\rho']_{2t}$ and so opening it reveals nothing. In addition, since $t < n/3$, the honest parties' shares alone *fully determine* the polynomial. Thus, the corrupted parties cannot change the opened value by sending incorrect shares. *This is the crucial property that enables us to carry out this verification step with such efficiency.*

We remark that it is not possible to simply open $[v_j]_{2t}$ and $[\rho_j]_t$ and compare that they are the same, without calling $\mathcal{F}_{\text{checkZero}}$. This is because in the case of

[5] Note that even though $[w_{i_k}]_t \cdot [w_{j_k}]_t$ is a sharing of degree-$2t$ (since each party locally multiplies its shares) and $[w_{\ell_k}]_t$ is a sharing of degree-t, it is possible for the parties to compute $[w_{i_k}]_t \cdot [w_{j_k}]_t - [w_{\ell_k}]_t$ locally. This is due to the fact that we are only interested in the constant term of the resulting (degree-$2t$) polynomial, and it will be zero if $w_{\ell_k} = w_{i_k} \cdot w_{j_k}$.

PROTOCOL 4.2 (Verification of Multiplication Correctness)

Inputs: The parties hold shares $\{[w_k]_t\}_{k=1}^M$ of wire values and gate definitions $(g_1, \ldots, g_N)$, where each g_k is a triple (i, j, ℓ) representing the multiplication gate $w_\ell = w_i \cdot w_j$.

Parameter: Let δ be such that $|\mathbb{F}|^\delta \geq 2^s$, where s is the statistical security parameter, and let κ be a computational security parameter for a pseudorandom function F.

The protocol:

1. *Phase 1 – prepare random values:*
   (a) The parties call $\mathcal{F}_{\text{coin}}$ to obtain a single key $K \in \{0,1\}^\kappa$ for a pseudorandom function F. Then, the parties compute $\delta \cdot N$ pseudorandom values $(b_1^1, \ldots, b_N^1), \ldots, (b_1^\delta, \ldots, b_N^\delta)$ by setting $b_k^j = F_K(k \| j)$.
   (b) The parties call $\mathcal{F}_{\text{rand}}^{\text{double}}$ to obtain double sharings $([\rho_1]_t, [\rho_1]_{2t}), \ldots, ([\rho_\delta]_t, [\rho_\delta]_{2t})$ of unknown random values $\rho_1, \ldots, \rho_\delta$.
2. *Phase 2 – actual verification:* Denote $g_k = (i_k, j_k, \ell_k)$ for every $k = 1, \ldots, N$.
   (a) For $j = 1, \ldots, \delta$, each party P_i locally computes its $2t$-share of
   $$[v_j]_{2t} = \left(\sum_{k=1}^{N} b_k^j \cdot (w_{i_k} \cdot w_{j_k} - w_{\ell_k}) \right) + [\rho'_j]_{2t}.$$
   (b) For $j = 1, \ldots, \delta$, the parties run $\mathsf{open}([v_j]_{2t})$, and obtain v_j. If a party receives $\perp$ in any opening, then it sends $\perp$ to all other parties, outputs $\perp$ and halts. Else, it continues.
   (c) For $j = 1, \ldots, \delta$, each P_i locally computes $[v'_j]_t = [\rho'_j]_t - v_j$.
   (d) For $j = 1, \ldots, \delta$, the parties call $\mathcal{F}_{\text{checkZero}}$ with $[v'_j]_t$.
3. *Output:* If a party receives accept from all calls to $\mathcal{F}_{\text{checkZero}}$ then it outputs accept; else, it outputs reject.

cheating, this will reveal the value $v_j - \rho_j$ which in the case of cheating in the kth gate only, is the difference between w_{ℓ_k} and $w_{i_k} \cdot w_{j_k}$. If π_{mult} is secure under an additive attack, then this value is already known and so it is fine to reveal it. However, we wish to prove the protocol secure, even for π_{mult} that is weaker (e.g., only being private, and enable detection of cheating). Thus, we need to call $\mathcal{F}_{\text{checkZero}}$ here. This makes very little difference in practice anyway, since it is run once for the entire circuit.

As in Lemma 4.1, we prove here only that the verification in Protocol 4.2 works. The fact that it reveals nothing is explained above, and proven formally when we simulate the full protocol itself.

Lemma 4.2. *Let n be the number of parties, let $t < n/3$ be the maximum number of corrupted parties controlled by an (unbounded) adversary, assume that F is a pseudorandom function for non-uniform distinguishers, and assume that all input shares $\{[w_k]_t\}_{k=1}^M$ are of degree-t. If there exists a k such that $w_{\ell_k} \neq w_{i_k} \cdot w_{j_k}$ where $g_k = (i_k, j_k, \ell_k)$, then there exists a negligible function μ such that the honest parties output* accept *in Protocol 4.2 with probability at most $2^{-s} + \mu(\kappa)$.*

Proof: We begin by proving the lemma when a truly random function is used instead of F, chosen after all $[w_1]_t, \ldots, [w_M]_t$ are fixed. By the assumption in the lemma statement, we have that all $[w_1]_t, \ldots, [w_M]_t$ are valid degree-t sharings but there exists a k such that $w_{\ell_k} \neq w_{i_k} \cdot w_{j_k}$ where $g_k = (i_k, j_k, \ell_k)$. Let d be such that $w_{\ell_d} \neq w_{i_d} \cdot w_{j_d}$; this implies that $b_d^j \cdot (w_{i_d} \cdot w_{j_d} - w_{\ell_d}) \neq 0$. Thus, $[v_j]_{2t}$ is a sharing of ρ'_j if and only if

$$b_d^j \cdot (w_{i_d} \cdot w_{j_d} - w_{\ell_d}) = -\left(\sum_{k=1 (k \neq d)}^{N} b_k^j \cdot (w_{i_k} \cdot w_{j_k} - w_{\ell_k}) \right).$$

As above, since b_d^j is uniformly distributed in $\mathbb{F}$, this equality holds with probability at most $1/|\mathbb{F}|$, and so it holds for *all* $j = 1, \ldots, \delta$ with probability at most $|\mathbb{F}|^{-\delta} < 2^{-s}$. Now, since in this case, all w-shares are guaranteed to be of degree-t, it follows that all polynomials $[v_j]_{2t}$ are guaranteed to be of degree-$2t$ (since $[v_j]_{2t}$ is computed via local operations only). Since $t < n/3$, there are at least $2t + 1$ honest parties, and so the honest parties' shares *fully determine* the polynomials that define the shares of $v_1, \ldots, v_\delta$. Thus, the corrupted parties cannot do anything to cause the honest parties to accept if $v_j \neq \rho'_j$ (any values sent by them that are not correct will be identified as being on a different polynomial, resulting in $\perp$ from the open procedure). We conclude that the honest parties output accept with probability at most 2^{-s} in this case.

We now complete the proof of the lemma, for the case that the pseudorandom function F is used, and not a truly random function. By the assumption, F is pseudorandom for non-uniform distinguishers. Assume, by contradiction, that there exists a series of shares $\{[w_k]_t\}_{k=1}^M$ and a $k \in \{1, \ldots, M\}$ such that $[w_k]_t$ is not a valid degree-t sharing and yet the honest parties output accept with probability that is non-negligibly greater than 2^{-s}. Then, these shares constitute non-uniform advice to the adversary D for distinguishing the pseudorandom function. This adversary D calls its oracle to obtain α, β values as in the protocol and checks that the honest parties' shares in $[u_j]_t$ define a degree-t polynomial or that $[v_j]_{2t}$ is a sharing of ρ'_j, for some $j \in \{1, \ldots, \delta\}$. If one of these events occur, then D outputs 1; otherwise it outputs 0. By the analysis above, D outputs 0 when given oracle access to a truly random function with probability at most 2^{-s}. In contrast, D outputs 1 with probability non-negligibly greater than 2^{-s} when given oracle access to the pseudorandom function. ■

Remarks on proof of security: We stress that we have actually proven a stronger claim, that the probability that the honest parties output reject in each of the δ iterations is negligibly close to $1 - \frac{1}{|\mathbb{F}|}$, in each of the cases. We will use this fact in the simulation of the full protocol.

4.3 Properties and Extensions

Security level achieved. We reiterate that although we assume the existence of pseudorandom functions (for non-uniform distinguishers), we obtain *statistical security* that holds even for all-powerful adversaries.

Unconditional security. It is possible to replace the use of the pseudorandom function with an ϵ-biased pseudorandom generation [21], and to repeat the test δ times where $\epsilon^\delta \leq 2^{-s}$. This suffices since the values generated are at most ϵ from uniform, meaning that the equations tested can only have different results from when truly random values are used with probability at most ϵ. Since ϵ-bias pseudorandom generators can be constructed unconditionally, we have the following two lemmas:

Lemma 4.3. *Let n be the number of parties, let $t < n/3$ be the maximum number of corrupted parties controlled by an (unbounded) adversary, and modify Protocol 4.1 to use an ϵ-biased pseudorandom generation with $\epsilon^\delta \leq 2^{-s}$. If there exists a $j \in [M]$ such that the sharing of x_j is not of degree-t, then the honest parties accept in this variant of Protocol 4.1 with probability at most $|\mathbb{F}|^{-\delta} \leq 2^{-s}$.*

Lemma 4.4. *Let n be the number of parties, let $t < n/3$ be the maximum number of corrupted parties, and modify Protocol 4.2 to use an ϵ-biased pseudorandom generation with $\epsilon^\delta \leq 2^{-s}$. If there exists a k such that $[w_k]_t$ is not a valid degree-t sharing, or $w_{\ell_k} \neq w_{i_k} \cdot w_{j_k}$ where $g_k = (i_k, j_k, \ell_k)$, then the honest parties output* accept *in this variant of Protocol 4.2 with probability at most 2^{-s+1}.*

Complexity of both checks: In most cases δ is very small (this is due to the fact that the size of the field must anyway be larger than the number of parties, and so the only exception is when the number of parties may be very small). For the calculation below, we assume that $n-2t > 2\delta^2+2$. The DoubleShareRandom(t, t') procedure of [2,3] generates $n-2t$ double sharings with each party sending n elements. This means that we can generate δ double sharings and another $\delta+2$ regular sharing at the cost of each party sending n field elements overall (for the regular sharings, the parties just throw out the degree-$2t$ shares). We also note that $\mathcal{F}_{\text{coin}}$ requires a single random sharing, and $\mathcal{F}_{\text{checkZero}}$ requires δ calls to generate double-random sharings. Thus, by assuming $n-2t > 2\delta^2+2$, a single call to DoubleShareRandom suffices. In addition, recall that $n-t$ values can be opened with open at the cost of each party sending n elements. Thus, three calls to open suffice throughout (one call to open suffices for the δ^2 values opened in all of the δ calls to $\mathcal{F}_{\text{checkZero}}$ because $n-t > \delta^2$).

Under the above assumption (that $n-2t > 2\delta^2+2$), the cost of both verifications is a single call to DoubleShareRandom to generate all the sharings needed at the cost of $2n+2n/3$ elements per party (to generate $n-2t = n/3$ double sharings), a single opening for $\mathcal{F}_{\text{coin}}$ at the cost of n elements per party, δ elements opened for Step 2b (at the cost of n elements per party) and another δ elements opened in Step 2b (at the cost of n elements per party) using the ReconsPub procedure of [2,3], and δ calls to $\mathcal{F}_{\text{checkZero}}$ at the cost of another execution of ReconsPub and n elements per party with δ calls to open for each call). The overall communication cost *per party* is therefore $(2n+2n/3)+4n = 6\frac{2}{3}\cdot n$ elements.

In the extreme case that the number of parties is small and the field is small, we need to call DoubleShareRandom at most $2\delta^2+2$ times and open at most $2\delta^2$ times. Thus, the cost in this case is $O(\delta^2 n)$ elements per party (with a small constant).

5 Damgård-Nielsen (DN) Multiplication Protocol – π_{mult}

5.1 Information-Theoretic DN Multiplication

Our underlying multiplication protocol is the one by Damgård and Nielsen [10], denoted DN from here on. A full description of the DN multiplication protocol appears in Protocol 5.1. As in [19], we include an additional check that the degree-t random sharings generated are of degree-t. This is needed in order to ensure that the masking by a degree-$2t$ sharing later on suffices to hide all secrets. As shown in [19], this also suffices to make the protocol secure up to an additive attack, as described above.

PROTOCOL 5.1 (The DN Multiplication Protocol – π_{mult})

Setup phase for multiplications: The parties generate a series of $n-t$ double random shares $\{[r_k]_t, [r_k]_{2t}\}_{k=1}^{n-t}$ where $[r_k]_t$ is a sharing of r_k using a t-degree polynomial and $[r_k]_{2t}$ is a sharing of r_k using a $2t$-degree polynomial. This generation works as follows:

1. Each party P_i chooses a random element $u_i \in \mathbb{F}$ and runs $\mathsf{share}(u_i)$ twice as the dealer, once using a degree-t polynomial and then using a degree-$2t$ polynomial.
2. Holding shares $([u_1]_t, \ldots, [u_n]_t)$ and $([u_1]_{2t}, \ldots, [u_n]_{2t})$, each party P_i locally computes
$$([r_1]_t, \ldots, [r_{n-t}]_t) = ([u_1]_t, \ldots, [u_n]_t) \cdot V_{n,n-t}$$
$$([r_1]_{2t}, \ldots, [r_{n-t}]_{2t}) = ([u_1]_{2t}, \ldots, [u_n]_{2t}) \cdot V_{n,n-t}$$
where $V_{n,n-t}$ is the Vandermonde matrix.

The above is run in parallel $\left\lceil \frac{N}{n-t} \right\rceil$ times, where N is the number of multiplications in the circuit to be computed. At the end, the parties have N random double sharings; denote them $([r_1]_t, [r_1]_{2t}), \ldots, ([r_N]_t, [r_N]_{2t})$.

Single verification: The parties run Protocol 4.1 on input shares $[r_1]_t, \ldots, [r_N]_t$. Each party proceeds if and only if it outputs accept from the protocol.

Multiplications: Let $[x]_t$ and $[y]_t$ be the next shares to be multiplied; denote by x_i and y_i the shares of x and y held by P_i.

1. Let $[r]_t, [r]_{2t}$ be the next unused double-random shares generated in the setup. Parties $P_2, \ldots, P_{2t+1}$ compute $[x]_t \cdot [y]_t - [r]_{2t}$ and send the result to party P_1 ($[x]_t \cdot [y]_t$ is locally computed by each party P_i computing $x_i \cdot y_i$ and is a valid degree-$2t$ sharing of $x \cdot y$).
2. Party P_1 uses the $2t+1$ shares it holds (its own plus $2t$ received) to reconstruct $\Delta = x \cdot y - r$, and then sends it to all the other parties.
3. Each party sends Δ to all other parties. (When many multiplications are run in parallel—as in the full layer of the circuit being computed—the parties send a single value $H(\Delta_1, \ldots, \Delta_m)$ where H is a collision-resistant hash function.)
4. If a party received the same Δ (or hash) from all, then it locally computes its output share $[z]_t = [x \cdot y]_t = [r]_t + \Delta = [r]_t + (xy - r)$. Else, it aborts.

Observe that Protocol 5.1 begins with generating double-random sharings; as such, we could use $\mathcal{F}_{\text{rand}}^{\text{double}}$ instead. However, $\mathcal{F}_{\text{rand}}^{\text{double}}$ generates double-random sharings that are provably correct; in contrast to the setup phase of Protocol 5.1 which has no such guarantees. This makes the generation of these sharings much more efficient in Protocol 5.1 (an average of 3 elements per party for each double sharing in Protocol 5.1, versus an average of 8 elements per party for each double sharing using $\mathcal{F}_{\text{rand}}^{\text{double}}$).

Security of π_{mult}: It has already been proven in [19] (building on [12,13]) that the version in Protocol 5.1 of the Damgård-Nielsen multiplication is secure up to additive attacks (see Section 6.1.1 and π_{mult}). Intuitively, this is due to the fact that all $[r_i]_t$ values are verified to be of degree-t and therefore valid, and the output is defined by *adding* some value to $[r_i]_t$. As such, the output of each multiplication is a valid value. Regarding the fact that the adversary can add some d to $x \cdot y$, if P_1 is corrupted then this is clear. In addition, since there is no check that the same r_i is shared in the degree-t and degree-$2t$ sharing, a gap between them would also result in some $d \neq 0$ being added to $x \cdot y$. However, since the simulator knows all actual values (since it holds all the honest parties' shares in the simulation), it knows exactly what that gap would be. We stress that in order to have this d be well defined, we must ensure that the same value Δ is sent by P_1 to all other parties. This is the reason for the echo of the Δ values in Step 3 of Protocol 5.1.[6] Formally, π_{mult} securely computes $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$, as formalized in Functionality 5.2.

FUNCTIONALITY 5.2 ($\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ - Secure Mult. Up To Additive Attack)
Let I denote the subset of corrupted parties.

1. Upon receiving the shares in $[x]_t$ and $[y]_t$ from the honest parties, the ideal functionality $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ computes x and y, and computes the corrupted parties' shares in $[x]_t$ and $[y]_t$.
2. $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ hands the ideal-model adversary $\mathcal{S}$ the corrupted parties' shares in $[x]_t$ and $[y]_t$.
3. Upon receiving d and $\{\alpha_i\}_{i \in I}$ from $\mathcal{S}$, functionality $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ defines $z = x \cdot y + d$ and generates a random (valid degree-t) sharing $[z]_t$ of z, under the constraint that for every $i \in I$, P_i's share in $[z]_t$ is α_i.
4. $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ hands the honest parties their shares in $[z]_t$.

We have the following (stated for $t < n/3$ since that is what we need):

Lemma 5.1 ([12,13,19]). *Assuming that the input shares $[x]_t, [y]_t$ held by the honest parties are of degree-t, Protocol 5.1 securely computes $\mathcal{F}_{\mathsf{mult}}^{\text{add}}$ in the presence of an (unbounded) adversary corrupting any $t < n/3$ parties, with statistical security.*

[6] We stress that without this step, there is an actual attack on the privacy of the protocol when run over multiple layers of multiplications. This was pointed out to us by Yifan Song.

Complexity: The share procedure involves each party sending one field element to each other party. Thus, in the setup phase each party sends $2n$ elements, meaning an overall $2n^2$. Then, in the multiplication phase itself, $2t$ parties send 1 element each to P_1, and P_1 sends back 1 element to each party resulting in a total of n elements. It is possible for a different party to "play" P_1 in each multiplication, so we can average the number of elements sent by all parties. The multiplication phase is run $n-t$ times, and so the total number of elements sent in both the setup and multiplication phases is $2n^2 + (2t+n)\cdot(n-t)$. Since this is the cost of computing $n-t$ multiplications, the average overall cost per multiplication is $\frac{2n^2}{n-t} + 2t + n$. Averaging this over the n parties, the cost is $\frac{2n}{n-t} + \frac{2t}{n} + 1$ elements per party per multiplication. Thus, when taking $t = n/3$ we have $n - t = 2n/3$, and the average cost per multiplication per party is $3 + 2/3 + 1 = 4\frac{2}{3}$ elements.

We remark that the cost of sending the hash of the Δ values is not counted, since it is insignificant except for "very narrow" circuits. In particular, this adds a single element per part for every single layer of the circuit being computed.

5.2 Computational DN Multiplication with PRFs

It is possible to construct a computationally-secure version of the DN multiplication protocol with significantly less communication. This was described in [22] for the case of $t < n/2$ yielding a multiplication protocol where 3 field elements are sent by each party per multiplication. We use the same ideas here for $t < n/3$ to achieve DN-type multiplication at the cost of $\frac{8}{3}$ field elements sent per party per multiplication.

Note, we are not referring to the non-interactive method of generating secret sharings as described in [9], since that method has exponential computational complexity and is thus only efficient for a very small number of parties. In contrast, here our aim is to *reduce* the number of elements sent while maintaining efficient computational complexity. even for a large number of parties.

The optimization. The idea behind the optimization is that in order to generate a sharing via a degree-d polynomial, it is possible to first choose d shares in any way at all. Then, the $(d+1)$th share is chosen at random (defining a random secret), the polynomial is then reconstructed using interpolation and the remaining shares are computed (or equivalently, Lagrange interpolation is used to directly generate the shares of the parties). Of course, in order for this to be a *secure* secret sharing of the secret, the first d shares must be random to all parties other than each party receiving the share. However, this can be achieved computationally by choosing these d shares *pseudorandomly*. As such, the dealer can initially send a pseudorandom function key to d parties. Then, in order to generate a sharing via a degree-d polynomial, each of the d parties can compute their local point by applying the pseudorandom function to some unique identifier for this derivation. The dealer who chose these keys can also generate the points locally, and thus no interaction is needed for these d parties.

In the first step of the setup in Protocol 5.1, each party needs to generate two sharings of some random u_i, using a degree-t and degree-$2t$ polynomial.

Furthermore, recall that only $2t+1$ parties send the opening of the degree-$2t$ polynomial $[x \cdot y - r]_{2t}$. Thus, only $2t+1$ parties need to ever receive the degree-$2t$ sharing of u_i in the first place, and these can be the the $2t$ parties who received pseudorandom function keys from the dealer plus one additional party. In summary, the first step of Protocol 5.1 can be replaced with the following:

1. *Initial setup:* Each party P_i chooses pseudorandom function keys $k_i^1, \ldots, k_i^t$ and sends them to t different parties (different dealers should be sent these to different parties in order to load balance). In addition, P_i chooses $\tilde{k}_i^1, \ldots, \tilde{k}_i^{2t}$. If $i > 2t$ (and so P_i is not one of the "first" $2t$ parties), then it sends key $\tilde{k}_i^j$ to party P_j for $j = 1, \ldots, 2t$. If $1 \leq i \leq 2t$ (and so P_i is one of the first $2t$ parties), then it sends the keys to $P_1, \ldots, P_{2t+1}$ except for to itself.
2. *Step 1 of the setup – sharing a random u_i associated with unique id:* Each party P_i chooses a random u_i and computes shares of t parties using $k_i^1, \ldots, k_i^t$ and $\tilde{k}_i^1, \ldots, \tilde{k}_i^{2t}$, by applying the pseudorandom function with the appropriate key to id. Then, it interpolates with point $(0, u_i)$ to obtain polynomials defining $[u_i]_t$ and $[u_i]_{2t}$. Finally, for the degree-t sharing, it sends each of the $n-t-1 = 2t$ other parties their share, and for the degree-$2t$ sharing it computes the $(2t+1)$th share. If $i > 2t$ then it sends the share to P_{2t+1}, whereas if $1 \leq i \leq 2t$ then it defines this as its own share. Likewise, each party with a pseudorandom function key defines its own share via local computation.
3. *Step 2 of the setup:* All n parties compute $[r_1]_t, \ldots, [r_{n-t}]_t$ via the Vandermonde multiplication, whereas only parties $P_1, \ldots, P_{2t+1}$ compute $[r_1]_{2t}, \ldots, [r_{n-t}]_{2t}$ (because only they have degree-$2t$ shares).

The rest of the protocol remains the same. As in basic DN multiplication, we stress that the $2t+1$ parties involved in reconstruction of $[x \cdot y - r]_{2t}$ is different for each batch of $n-t$ shares generated. This provides load balancing of the work. By a straightforward reduction to the pseudorandom function, we have:

Lemma 5.2. *If the function used in the optimized version of Protocol 5.1 is a pseudorandom function, then this variant of Protocol 5.1 securely computes $\mathcal{F}_{\mathsf{mult}}^{\mathrm{add}}$ in the presence of a (polynomial-time) adversary corrupting any $t < n/3$ parties, with computational security.*

Complexity. The setup procedure now involves each party sending only $n - t - 1 = 2t$ field elements to other parties for the degree-t sharing, and either 0 or 1 field element for the degree-$2t$ sharing. Thus, in the setup phase the overall communication is $2tn$. Then, the cost in the multiplication phase is the same as previously, meaning that for $n-t$ multiplication gates the parties send $(2t+n) \cdot (n-t)$ field elements overall. The total number of elements therefore sent in both the setup and multiplication phases is $2tn + (2t+n) \cdot (n-t)$ for $n-t$ multiplications, yielding an average overall cost per multiplication of $\frac{2tn}{n-t} + 2t + n$. Averaging this over the n parties, we have a cost of $\frac{2t}{n-t} + \frac{2t}{n} + 1$ elements per party per multiplication. Thus, when taking $t = n/3$ we have $n - t = 2n/3$, and the average cost per multiplication per party is $1 + 2/3 + 1 = 2\frac{2}{3}$ elements.

6 The Protocol

We are now ready to present the protocol for securely computing any functionality, via an arithmetic circuit representation of the function (over a field that is larger than the number of parties). The idea behind the protocol is simple: the parties first share their inputs, and then they compute shares of the output by locally computing addition gates and using π_{mult} to compute multiplication gates. After all of this computation has concluded, the parties run the verification method of Protocol 4.2 to ensure that all multiplications are valid. Recall that this check assumes that all shares are of degree-t. For the shares generated on the wires coming out of multiplication gates, this is guaranteed by the degree verification of $[r_1]_t, \ldots, [r_N]_t$ incorporated into π_{mult} (this ensures that all shares output from multiplications are also of degree-t).

PROTOCOL 6.1 (Computing Any Arithmetic Circuit)

Inputs: Each party P_j ($j \in \{1, \ldots, n\}$) holds an input $x_j \in \mathbb{F}^\ell$.

Auxiliary Input: a description of a finite field $\mathbb{F}$ and an arithmetic circuit C over $\mathbb{F}$ that computes f; let N be the overall number of gates in C.

The protocol (throughout, if any party receives $\perp$ as output from a call to a sub-functionality, then it sends $\perp$ to all other parties, outputs $\perp$ and halts):

1. *Secret sharing the inputs:*
   (a) For each input v_i held by P_j, party P_j runs share as the dealer with v_i.
   (b) The parties run Protocol 4.1 on all shares received in the previous step. The parties proceed if and only if they output accept from the protocol.
   (c) Each party P_j records its vector of shares $(v_1^j, \ldots, v_M^j)$ of all inputs.
2. *Circuit emulation:* Let $G_1, ..., G_N$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, ..., N$ the parties work as follows:
   - G_k *is an addition gate:* Given shares $[x]$ and $[y]$ on the input wires, the parties locally compute $[x + y] = [x] + [y]$.
   - G_k *is a multiplication-by-constant gate:* Given share $[x]$ on the input wire and the constant $a \in \mathbb{F}$, the parties locally compute $[a \cdot x] = a \cdot [x]$.
   - G_k *is a multiplication gate:* Given shares $[x]$ and $[y]$ on the input wires:
     (a) The parties run π_{mult} on $[x]$ and $[y]$ to receive the share $[z]$ on the output wire.
3. *Multiplication verification:* The parties run Protocol 4.2 with the set of inputs being the shares $\{[w_\ell]_t\}$ on the input wires, and the degree-t shares $\{[r_k]_t\}$ generated for all multiplication gates in π_{mult}. If a party outputs accept from Protocol 4.2, then it proceeds to the next step. Else, it outputs $\perp$ and halts.
4. *Output reconstruction:* For each output wire of the circuit, the parties run (reconstruct$[v]_t, j$), where $[v]_t$ is the sharing of the value on the output wire, and P_j is the party who receives this output.
   If a party received $\perp$ as output from any call, then it sends $\perp$ to the other parties, outputs $\perp$ and halts.

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

For the input wires, we run an additional invocation of Protocol 4.1 to ensure that everything is indeed of degree-t. This enables us to use straightforward (semi-honest secure) input-sharing and not far more expensive VSS or the like. In practice, this verification can be carried out in the same invocation of Protocol 4.1 as in π_{mult}, and so there is no additional cost. At the end of the protocol, if the verification passes, then the parties open the shares on the output wires and conclude. See Protocol 6.1 for a full description.

The intuition as to why the protocol is secure follows from the privacy of the underlying multiplication protocol (as stated in Section 5.1) and the validity of the verification method (proven in Sections 4.1 and 4.2). Specifically, if all shares on all wires are of degree-t, and all multiplications are correct, then the output is certainly correct. In addition, the security of the secret sharing scheme and the DN-multiplication protocol means that nothing beyond the output is revealed. Since π_{mult} as presented is secure under additive attacks, we prove the protocol secure for *any* multiplication protocol that is secure up to additive attacks.

Theorem 6.1. *Let n be the number of parties, $t < n/3$ be the maximum number of corrupted parties, let f be an n-party functionality, and let π_{mult} be a multiplication protocol that is secure up to additive attacks. Assume that F used in Protocol 4.1 and 4.2 is a pseudorandom function for non-uniform distinguishers. Then, Protocol 6.1 t-securely computes f with statistical security in the presence of malicious adversaries. If the version of π_{mult} of Section 5.2 is used, then Protocol 6.1 t-securely computes f with computational security in the presence of malicious adversaries.*

Proof: The intuition is provided above and so we proceed directly with the proof. We prove the protocol secure in the $\mathcal{F}_{\mathsf{mult}}^{\mathsf{add}}$-hybrid model (recall that this is the multiplication functionality with security up to additive attacks).

Let $\mathcal{A}$ be an adversary controlling the subset of parties indexed by $I \subset [n]$; we construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ invokes $\mathcal{A}$ on the corrupted parties' inputs (that it has in the ideal model).
2. *Simulation of secret-sharing stage:* $\mathcal{S}$ plays the role of the honest parties in this phase, using 0-values for all honest-party inputs. If in the verification step, the honest parties would abort, then $\mathcal{S}$ simulates the honest parties messages in Protocol 4.1 and sends abort to the trusted party computing the functionality (we stress that $\mathcal{S}$ can simulate the honest parties' messages since it receives all of the incorrect shares sent by the adversary). Else, it sends the values defined by the shares dealt by $\mathcal{A}$ for the corrupted parties. It can obtain these values since it obtains all of the shares sent by the corrupted parties. (If any of these sharings are invalid and don't define a single degree-t polynomial and yet the honest parties did not abort in the simulated execution, then $\mathcal{S}$ outputs bad and halts.)
3. *Simulation of circuit-emulation stage:* $\mathcal{S}$ emulates the trusted party running $\mathcal{F}_{\mathsf{mult}}^{\mathsf{add}}$ in each multiplication execution. Observe that $\mathcal{F}_{\mathsf{mult}}^{\mathsf{add}}$ does not provide the adversary any output, beyond the corrupted parties' shares on the input wires to the multiplication. $\mathcal{S}$ derives these values from the honest parties'

shares using the complete procedure described in Section 2.1. If $\mathcal{S}$ receives any $d \neq 0$ from $\mathcal{A}$ in any of the calls to $\mathcal{F}_{\mathsf{mult}}^{\mathrm{add}}$, then $\mathcal{S}$ stores cheat.

4. *Simulation of verification stage:*
   (a) If $\mathcal{S}$ did not store cheat in the previous step, then it plays the role of the honest parties in Protocol 4.2, sending honest values. The only place that the values sent by the honest parties depend on the input shares is when opening $[v_j]_{2t}$, but since this is masked, $\mathcal{S}$ chooses a random $v_j \in \mathbb{F}$ and sets the honest parties' shares in the opening by running complete on the corrupted parties' shares and v_j (it can do this since it knows the shares the corrupted parties' hold). Finally, as long as no invalid values are sent by $\mathcal{A}$ for the openings, it simulates the output of $\mathcal{F}_{\mathrm{checkZero}}$ being accept.
   (b) If $\mathcal{S}$ did store cheat in the circuit-emulation phase, then it works in exactly as in the previous case, except that it simulates the output of $\mathcal{F}_{\mathrm{checkZero}}$ being reject.
5. If there was no abort until this point, $\mathcal{S}$ uses complete on the output values it received for the corrupted parties along with their shares, in order to generate the honest parties' shares that would be sent at this point. In addition, for any honest party P_j for whom all shares sent by the corrupted parties in reconstruct are correct, it sends $(\mathsf{continue}, j)$ to the trusted party computing f to notify it to provide output to P_j in the ideal model; otherwise it sends (abort, j) to notify that P_j not receive output in the ideal model.

We argue that the simulation is statistically close to the real execution when using the information-theoretic multiplication protocol described in Protocol 5.1, and is computationally indistinguishable when using the computational multiplication protocol described in Section 5.2. Regarding the secret-sharing phase, this follows immediately from the property of the secret sharing scheme (that guarantees that sharings of 0 and other values are identically distributed), and by the fact that Protocol 4.1 reveals nothing. In order to see this latter fact, observe that $\mathcal{F}_{\mathrm{coin}}$ and $\mathcal{F}_{\mathrm{rand}}$ are guaranteed to be secure (and so we actually prove security in the $(\mathcal{F}_{\mathrm{coin}}, \mathcal{F}_{\mathrm{rand}})$-hybrid model), and the only values opened are $[u_1]_t, \ldots, [u_\delta]_t$. Now, this masking is only of degree-t. However, all honest parties provide input sharings of degree-t only (since that is what the protocol specification says to do). Thus, all honest values are perfectly masked, and using 0 or the honest party's correct input value yields the same distribution.

Next, the circuit emulation phase is perfectly simulated, by the use of the $\mathcal{F}_{\mathsf{mult}}^{\mathrm{add}}$-hybrid model. Note that this assumes that all inputs are of degree-t. However, this is guaranteed by the execution of Protocol 4.1 on the input values, and the fact that all outputs from $\mathcal{F}_{\mathsf{mult}}^{\mathrm{add}}$ are of degree-t (by the functionality definition). Observe that by Lemma 4.1, the probability that a sharing is not of degree-t but the parties did not abort, is at most 2^{-s}.

Finally, for the multiplication verification phase, recall that all values on all wires are guaranteed to be of degree-t, and that $[\rho'_j]_{2t}$ is generated using $\mathcal{F}_{\mathrm{rand}}^{\mathrm{double}}$ which is guaranteed to therefore be of degree-$2t$. Thus, the polynomial sharing defined by $\sum_{k=1}^{N} b_k^j \cdot (w_{i_k} \cdot w_{j_k} - w_{\ell_k})$ via local multiplications and additions

is guaranteed to be of degree-$2t$. This implies that $[v_j]_{2t}$, which is obtained by adding $[\rho'_j]_{2t}$ to this polynomial sharing, is a truly random sharing. Thus, $\mathcal{S}$ perfectly simulates the messages received by the corrupted parties (except with probability 2^{-s} when inputs were not of degree-t). The output of $\mathcal{F}_{\text{checkZero}}$ is also simulated perfectly, except with probability at most 2^{-s}, which occurs when cheat was stored and yet the polynomial defined is a zero-polynomial, and so the honest parties would not abort. This is bound by probability 2^{-s}, as shown in Lemma 4.2.

Regarding the computational case, this follows from the exact same reasoning, except that the transition from the $\mathcal{F}_{\mathsf{mult}}^{\mathsf{add}}$-hybrid model to the real model is computational instead of information-theoretic. ■

Relaxing the requirement. Informally speaking, in the proof of Theorem 6.1 we only really utilize the fact that the adversary's view alone in π_{mult} can be simulated (i.e., a privacy requirement), and that a *simulator* can detect if a party has cheated. This is a very mild requirement on π_{mult} and we leave the task of formalizing it (and generalizing the protocol to arbitrary linear secret sharing schemes) to future work. Nevertheless, we stress that the DN multiplication protocol is the most efficient known, even for semi-honest, and thus this makes no difference in practice right now. However, if a more efficient protocol is found later and it does not fulfil security up to additive attacks, then this can be beneficial.

Complexity: We count the *number of elements per party sent* for the *computationally secure* variant of the protocol. Let $N_I, N_\times, N_O$ denote the number of input, multiplication, and output gates in C, respectively. Then, each input gate is a single call to share which costs 1 element on average per party, and each output gate is a single call to reconstruct which is also exactly 1 element per party. As we have described in Section 5.2, the cost of π_{mult} for the case of $t < n/3$ is $2\frac{2}{3}$ elements per party per multiplication. Finally, as shown in Section 4.3, assuming $n - 2t > 2\delta + 2$ (which holds for large enough n as well as for most reasonable parameters), the cost of verification is $6\frac{2}{3}n$ elements per party. Without making this assumption, the cost is $O(\delta^2 n)$ elements per party.

We conclude that the total number of field elements sent per party in the protocol is $N_I + 2\frac{2}{3} \cdot N_\times + N_O + 6\frac{2}{3}n$ (when assuming $n - 2t > 2\delta + 2$) and at most $N_I + 2\frac{2}{3} \cdot N_\times + N_O + O(\delta^2 n)$ (even for a very small field and small number of parties). Observing that the cost of *semi-honest* alone is exactly $N_I + 2\frac{2}{3} \cdot N_\times + N_O$, we have that the only overhead occurred in order to obtain malicious security is the *additive factor* of between $6\frac{2}{3}n$ and $O(\delta^2 n)$ elements sent per party.

Achieving fairness. Our protocol can be easily extended to guarantee fairness. Since $t < n/3$, it suffices for all honest parties to first agree (via a Byzantine Agreeement protocol) that they did not receive any abort in the verification. If this is the case, then in the opening, all honest parties are guaranteed to have at least $2t + 1$ honest shares and at most t corrupt shares. Thus, using standard

error correcting techniques, the honest parties can determine the correct values and output them. Observe that this method adds very little cost to the protocol.

7 Experiments and Evaluation

We implemented our protocol and carried out extensive experiments. Our implementation is *single threaded*, to facilitate accurate comparisons with other protocols. Our implementation will be made open source upon publication.

7.1 Experiment 1 – Our Protocol Comparison

The aim of this experiment was to understand the efficiency gain achieved of the PRF version of DN-multiplication of Section 5.2 versus the information-theoretic version of Section 5.1. Theoretically, the saving is over 40%. However, this is in communication, and the necessity to compute many PRF invocations (using AES) may impact the running time. In addition, we analyzed the additional cost incurred for achieving malicious security over semi-honest security. We counted the amount of time spent on verification in the malicious protocol (which is the only difference between the semi-honest and malicious variants) as well as running independent semi-honest executions.

We ran the above experiment using a circuit of 1,000,000 multiplication gates of depth-20, with a 61-bit field (defined by a Mersenne prime). The experiment was run on `c5.xlarge` instances on AWS with all parties in the EAST-US region. The results appear in Table 1 and Table 2, and in Figure 1.

All running times are given in *milliseconds*, and are the average of *20 executions*. The columns titled "verify time" and "% on verify" describe the amount of time spent on the verification procedures of Section 4 in the malicious protocol, whereas the column titled "semi-honest" is an independent execution of the completely semi-honest protocol (without verification inside Protocol 5.1 or Protocol 6.1) with the "% difference" being between the full malicious and independent semi-honest executions.

Observe that the percentage of time spent on verification is small, and decreases as the number of parties increases. Since this verification step has communication that is independent of the circuit size (and only cheat local computation), this is also true as circuits get bigger. In particular, for 100 parties, the percentage of time spent on verification is a few percent only. Observe that the running time of the purely semi-honest protocol is typically farther away; this is surprising since the malicious protocol without the verification is exactly the same. The only explanation that we have is that the variance on the network at different running times has a big impact when the running times are so low.

Regarding the comparison between the information-theoretic and PRF versions, the difference in running times is more significant for a smaller number of parties, less than 50% for up to 60 parties and about 40% for over 60 parties (with an anomalous point at 100 parties). This matches the theoretical expectation (and in fact, even more for up to 60 parties). The fact that the PRF computations are insignificant is due to the fact that AES-NI makes such computations very low cost.

Parties	Malicious	Verify Time	% on Verify	Semi-Honest	% Difference
10	401	36	8.9%	401	0%
20	936	53	5.6%	828	11.5%
30	1241	68	5.5%	1168	5.9%
40	1598	69	4.3%	1343	15.9%
50	1891	62	3.3%	1985	-5.0%
60	2512	126	5.0%	2219	11.7%
70	2585	75	2.9%	2870	-11.0%
80	2974	97	3.2%	2884	3.0%
90	3689	120	3.3%	3529	4.3%
100	3999	142	3.6%	4089	-2.2%

Table 1. Information-theoretic multiplication protocol version of Section 5.1

Parties	Malicious	Verify Time	% on Verify	Semi-Honest	% Difference
10	187	34	18.0%	159	14.8%
20	374	45	12.1%	343	8.4%
30	760	62	8.2%	644	15.3%
40	640	57	8.9%	609	4.9%
50	840	49	5.8%	850	-1.2%
60	1112	87	7.8%	1056	5.0%
70	1366	49	3.6%	1160	15.1%
80	1606	72	4.5%	1528	4.9%
90	2417	133	5.5%	1962	18.8%
100	3036	76	2.5%	2262	25.5%

Table 2. The computational multiplication protocol version of Section 5.2

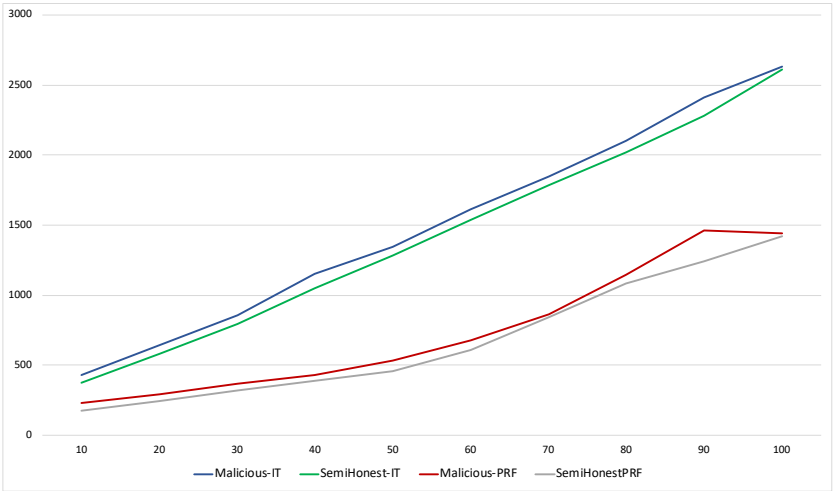

Fig. 1. Graphic comparison of all protocol versions.

7.2 Experiment 2 – Comparison to Prior Work

In this experiment, we compared our new protocol to the protocols of [2,8]. In this experiment, we used the more efficient PRF version of the protocol of Section 5.2. The protocol of [2], called HyperMPC, is for $t < n/3$ like our protocol, but achieves perfect security (in contrast to statistical security as here). The communication cost of HyperMPC is 13 field elements per multiplication gate, for any field size. Our protocol is under 3 elements per multiplication gate (plus the additive cost) and so is expected to be about 4 times faster than HyperMPC. The protocol of [8], that we call CRYPTO18 below, is for $t < n/2$ and has twice the cost of the semi-honest protocol for large fields (and even more for smaller fields).[7] We stress that although our protocol is more efficient than [8], we achieve a weaker threshold of $t < n/3$ instead of $t < n/2$. The experiments here were run on `c5.xlarge` instances on AWS in the EAST-US region, with a circuit with 1,000,000 multiplication gates and depths 20 and 100. The results of these experiments, for different depth circuits and different fields, appear in Figures 2 and 3, at the end of the paper.

It is interesting to note that in $GF[2^8]$, the cost of HyperMPC and our protocol is almost the same, despite the fact that our protocol has about a quarter of the communication. We conjecture that this is due to the fact that in small fields, the amount of communication is so low already in HyperMPC that the rounds of communication overrides the other costs (at least on circuits this size). However, in larger fields, the difference in running time is great. Note also that HyperMPC and CRYPTO18 have the same cost in large fields, as is expected by the theoretical costs.

7.3 Experiment 3 – Results for Mobile Phone Executions

One of the benefits of low bandwidth protocols is to enable many parties on weak devices to run secure computation amongst themselves. This was articulated in [2] who constructed an end-to-end system.

In order to demonstrate the suitability of our protocol for such a setting, we ran the protocol on ARM machines and on a mix of ARM machines and servers in AWS. We used the new AWS service for ARM machines with `a1.large` instances, and with `c5.xlarge` server instances. The specification of the ARM `a1.large` machines are two Cortex A72 CPUs, with clock speed 2.5GHz and 4GB RAM. These CPUs are those used in phones like Huawei P9, Xiaomi Redmi Pro and Samsung Galaxy C9 Pro (all released in 2016); note that these phones have four Cortex A72 CPUs and not two. Thus, this experiment demonstrates the viability of running MPC from simple mobile phones; in particular, high-end phones are not needed.

[7] For a statistical error of 2^{-40}, the CRYPTO18 protocol would be 3 times the semi-honest for a 31-bit field, and 6 times the semi-honest for $GF[2^8]$. Since [8] does not include an implementation for the protocol version for smaller fields, we only can compare our protocol to it for a 61-bit field.

We ran the experiment in two different *network latency* configurations: 90ms and 300ms. There are realistic latencies between mobile phones and clouds, and the two configurations reflect distances to the cloud. For example, 90ms is the latency of a mobile phone to a relatively close cloud (e.g., the latency from a mobile phone in the Middle East to all European AWS clouds is below 90ms), and 300ms is approximately the global latency. (In order to see this from your phone and your location, run `https://www.cloudping.info` from your mobile phone.)

The experiment is for the same 1,000,000 multiplication gate and depth-20 circuit above, over the Mersenne-31 field. Each experiment was run 20 times, and the result reported is the mean running time. The results appear in Table 3. The running times reported are extremely realistic (albeit, not "real time", but this is not the expected application), and demonstrate for the first time the viability of running end-to-end MPC with a large number of mobiles running the execution.

Parties Configuration	**Network Latency**	**Running Time**
10 ARM `a1.large`	90ms	9.9
50 ARM `a1.large`	90ms	46.4
50 ARM `a1.large` and 50 servers `c5.xlarge`	90ms	95.9
10 ARM `a1.large`	300ms	22.1
50 ARM `a1.large`	300ms	101.7
50 ARM `a1.large` and 50 servers `c5.xlarge`	300ms	303.2

Table 3. Running times in *seconds* for a circuit of 1,000,000 multiplication gates and depth-20 with a 31-bit Mersenne prime.

Acknowledgments

We thank Meital Levy for implementing the protocol, Lior Koskas for running the experiments, and Shai Halevi for helpful comments. We thank Yifan Song for pointing out an error in an earlier version of the paper.

References

1. T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman and O. Weinstein. Optimized Honest-Majority MPC for Malicious Adversaries - Breaking the 1 Billion-Gate Per Second Barrier. In the *IEEE S&P*, 2017.
2. A. Barak, M. Hirt, L. Koskas and Y. Lindell. An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for Weak Participants. In the 25*th ACM CCS*, pages 695-712, 2018.
3. Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In *TCC 2008*, Springer (LNCS 4948), pages 213–230, 2008.

4. M. Ben-Or, S. Goldwasser and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In 20*th STOC*, 1988.
5. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
6. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic In 42*nd FOCS*, pages 136–145, 2001.
7. D. Chaum, C. Crépeau and I. Damgård. Multi-party Unconditionally Secure Protocols. In 20*th STOC*, pages 11–19, 1988.
8. K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell and A. Nof. Fast Large-Scale Honest-Majority MPC for Malicious Adversaries. In *CRYPTO 2018*, Springer (LNCS 10993), pages 34–64, 2018.
9. R. Cramer, I. Damgård and Y. Ishai, Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC*, Springer (LNCS 3378) pages 342–362, 2005.
10. I. Damgård and J. Nielsen. Scalable and unconditionally secure multiparty computation. In *CRYPTO 2007*, Springer (LNCS 4622), pages 572–590, 2007.
11. D. Evans, V. Kolesnikov and M. Rosulek. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.
12. D. Genkin, Y. Ishai, M. Prabhakaran, A. Sahai and E. Tromer. Circuits Resilient to Additive Attacks with Applications to Secure Computation. In *STOC 2014*, 2014.
13. D. Genkin, Y. Ishai and A. Polychroniadou. Efficient Multi-party Computation: From Passive to Active Security via Secure SIMD Circuits. In *CRYPTO 2015*.
14. O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. In 19*th STOC*, pages 218–229, 1987.
15. R. Gennaro, M. Rabin and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In 17*th PODC*, 1998.
16. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications*, 2004.
17. S. Goldwasser and Y. Lindell. Secure Computation Without Agreement. In the *Journal of Cryptology*, 18(3):247–287, 2005.
18. E. Kushilevitz, Y. Lindell and T. Rabin. Information-Theoretically Secure Protocols and Security Under Composition. In the *SIAM Journal on Computing*, 39(5):2090–2112, 2010.
19. Y. Lindell and A. Nof. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In the 24*th ACM CCS*, pages 259–276, 2017. (References to exact protocol and theorem numbers are from the exact version https://eprint.iacr.org/2017/816/20181212:105515.)
20. Y. Lindell and B. Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In the *8th TCC*, Springer (LNCS 6597), 329–346, 2011.
21. J. Naor and M. Naor. Small-Bias Probability Spaces: Efficient Constructions and Applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.
22. P.S. Nordholt and M. Veeningen. Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification. In *ACNS 2018*, Springer (LNCS 10892), pages 321–339, 2018.
23. T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multi-party Protocols with Honest Majority. In 21*st STOC*, pages 73–85, 1989.
24. A. Shamir. How to share a secret. CACM, 22(11), pages 612–613, 1979.
25. A. Yao. How to Generate and Exchange Secrets. 27*th FOCS*, pages 162–167, 1986.

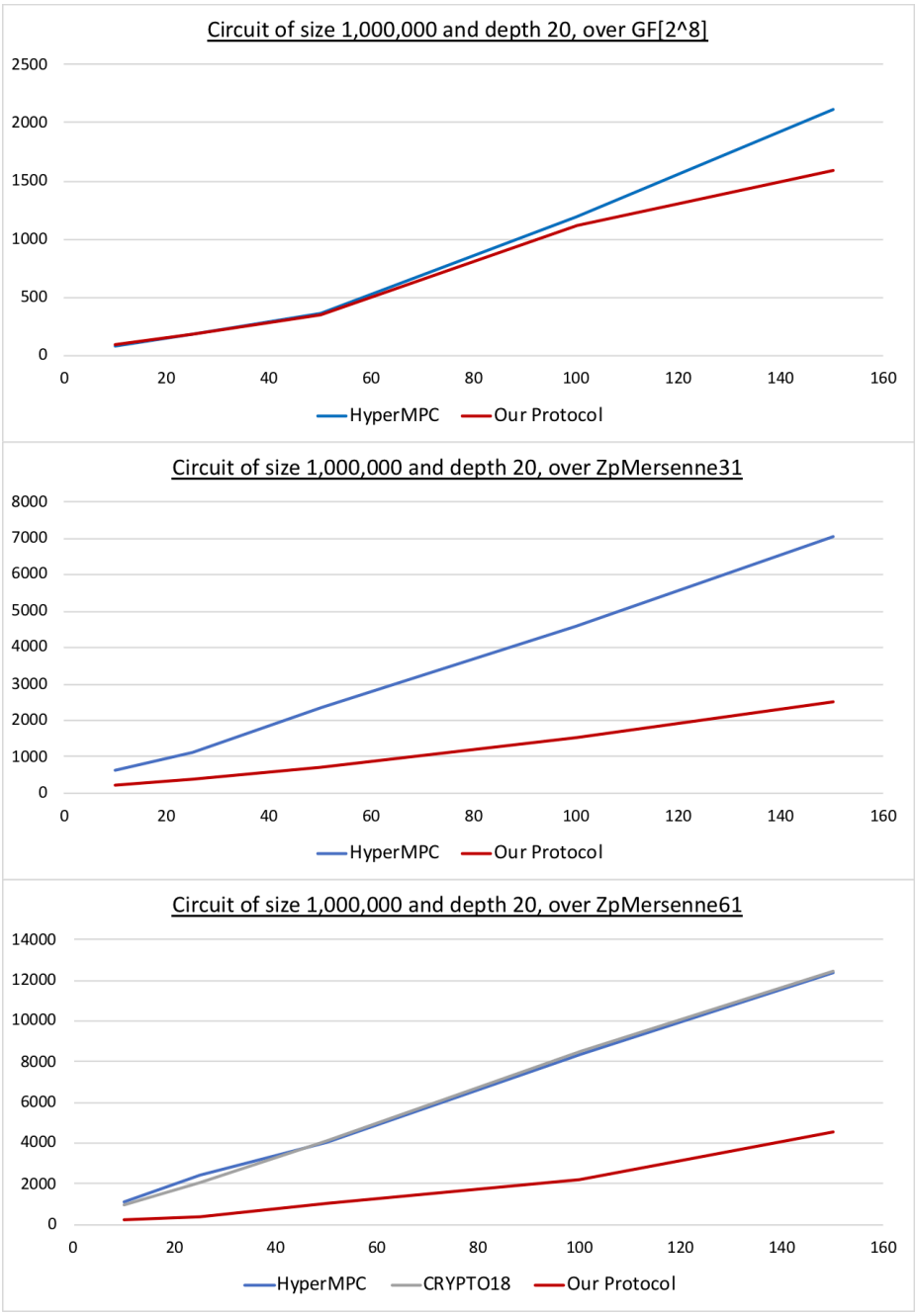

Fig. 2. Comparison of protocols for a circuit of depth 20: HyperMPC refers to [2] and CRYPTO18 refers to [8]. Our protocol is the PRF version.

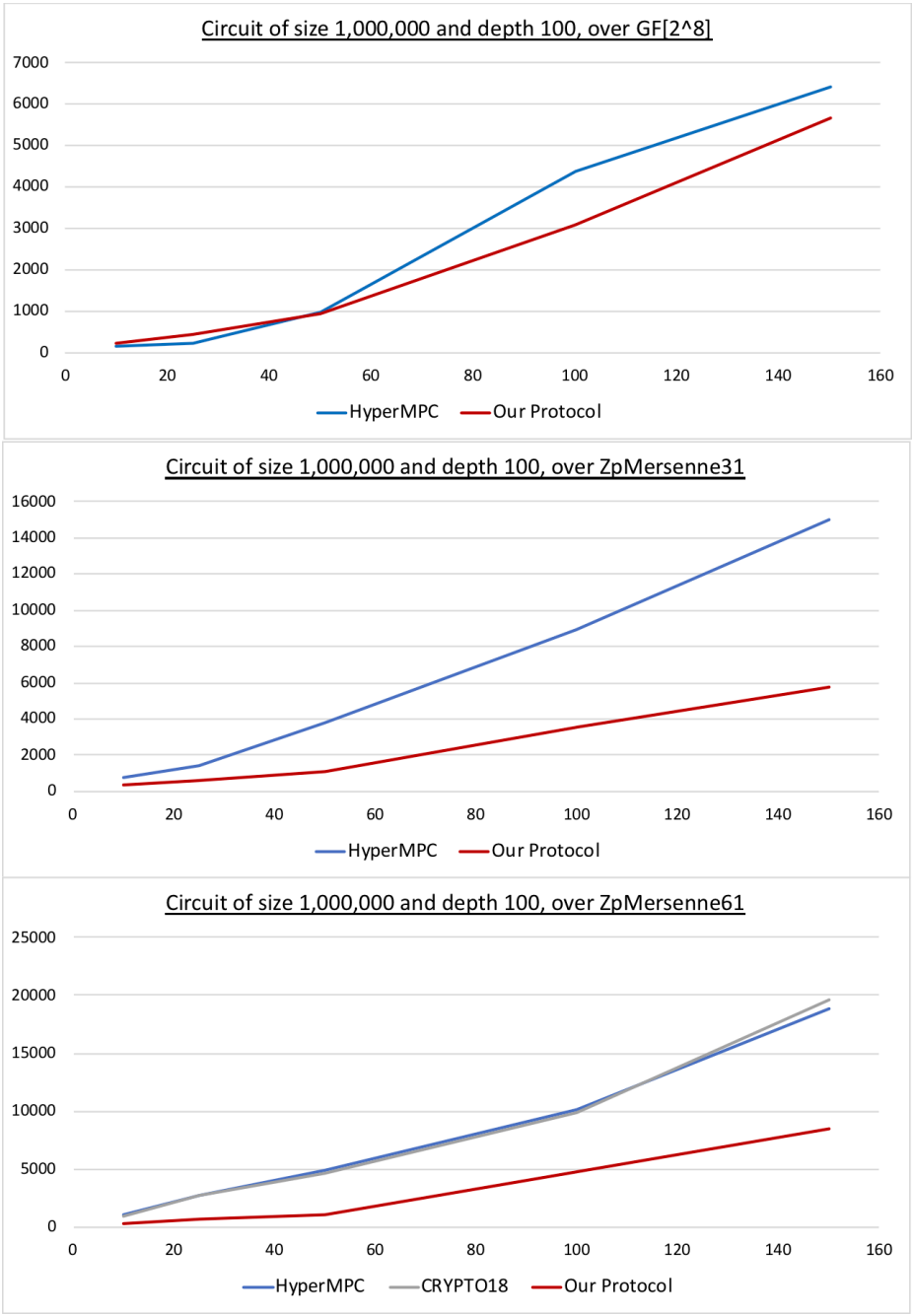

Fig. 3. Comparison of protocols for a circuit of depth 100: HyperMPC refers to [2] and CRYPTO18 refers to [8]. Our protocol is the PRF version.