Reducing the Overhead of MPC over a Large Population

A. Choudhury[1], A. Patra[2], and N. P. Smart[3]

[1] IIIT Bangalore, India.
[2] Dept. of Computer Science & Automation, IISc Bangalore, India.
[3] Dept. of Computer Science, Uni. Bristol, United Kingdom.
partho31@gmail.com, arpita@csa.iisc.ernet.in, nigel@cs.bris.ac.uk.

Abstract. We present a secure honest majority MPC protocol, against a static adversary, which aims to reduce the communication cost in the situation where there are a large number of parties and the number of adversarially controlled parties is relatively small. Our goal is to reduce the usage of point-to-point channels among the parties, thus enabling them to run multiple different protocol executions. Our protocol has highly efficient theoretical communication cost when compared with other protocols in the literature; specifically the circuit-dependent communication cost, for circuits of suitably large depth, is $\mathcal{O}(|\mathsf{ckt}|\kappa^7)$, for security parameter κ and circuit size $|\mathsf{ckt}|$. Our protocol finds application in cloud computing scenario, where the fraction of corrupted parties is relatively small. By minimizing the usage of point-to-point channels, our protocol can enable a cloud service provider to run multiple MPC protocols.

1 Introduction

Threshold secure multi-party computation (MPC) is a fundamental problem in secure distributed computing. It allows a set of n mutually distrusting parties with private inputs to "securely" compute any publicly known function of their private inputs, even in the presence of a centralized adversary who can control any t out of the n parties and force them to behave in any arbitrary manner. Now consider a situation, where n is very large, say $n \geq 1000$ and the proportion of corrupted parties (namely the ratio t/n) is relatively small, say 5 percent. In such a scenario, involving all the n parties to perform an MPC calculation is wasteful, as typical (secret-sharing based) MPC protocols require all parties to simultaneously transmit data to all other parties. However, restricting to a small subset of parties may lead to security problems. In this paper we consider the above scenario and show how one can obtain a communication efficient, robust MPC protocol which is actively secure against a computationally bounded static adversary. In particular we present a protocol in which the main computation is performed by a "smallish" subset of the parties, with the whole set of parties used occasionally so as to "checkpoint" the computation. By not utilizing the entire set of parties all the time enables them to run many MPC calculations at once. The main result we obtain in the paper is as follows:

Main Result (Informal): Let $\epsilon = \frac{t}{n}$ with $0 \leq \epsilon < 1/2$ and let the t corrupted parties be under the control of a computationally bounded static adversary. Then for a security parameter κ (for example $\kappa = 80$ or $\kappa = 128$), there

exists an MPC protocol with the following circuit-dependent communication complexity[4] to evaluate an arithmetic circuit ckt: **(a).** $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$ for ckt with depth $\omega(t)$. **(b).** $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^4)$ for ckt with $d = \omega(t)$ and $w = \omega(\kappa^3)$ (i.e. $|\mathsf{ckt}| = \omega(\kappa^3 t)$).

Protocol Overview: We make use of two secret-sharing schemes. A secret-sharing scheme $[\cdot]$ which is an actively-secure variant of the Shamir secret-sharing scheme [22] with threshold t. This first secret-sharing scheme is used to share values amongst *all* of the n parties. The second secret-sharing scheme $\langle\cdot\rangle$ is an actively-secure variant of an additive secret-sharing scheme, amongst a well-defined subset $\mathcal{C}$ of the parties.

Assuming the inputs to the protocol are $[\cdot]$ shared amongst the parties at the start of the protocol, we proceed as follows. We first divide ckt into L levels, where each level consists of a sub-circuit. The computation now proceeds in L phases; we describe phase i. At the start of phase i we have that *all* n parties hold $[\cdot]$ sharings of the inputs to level i. The n parties then select (at random) a committee $\mathcal{C}$ of size c. If c is such that $\epsilon^{\mathsf{c}} < 2^{-\kappa}$ then statistically the committee $\mathcal{C}$ will contain at least one honest party, as the inequality implies that the probability that the committee contains no honest party is negligibly small. The n parties then engage in a "conversion" protocol so that the input values to level i are now $\langle\cdot\rangle$ shared amongst the committee. The committee $\mathcal{C}$ then engages in an actively-secure dishonest majority[5] MPC protocol to evaluate the sub-circuit at level i. If no abort occurs during the evaluation of the ith sub-circuit then the parties engage in another "conversion" protocol so that the output values of the sub-circuit are converted from a $\langle\cdot\rangle$ sharing amongst members in $\mathcal{C}$ to a $[\cdot]$ sharing amongst all n parties. This step amounts to check-pointing data. This ensures that the inputs to all the subsequent sub-circuits are saved in the form of $[\cdot]$ sharing which guarantees recoverability as long as $0 \leq \epsilon < \frac{1}{2}$. So the check-pointing prevents from re-evaluating the entire circuit from scratch after every abort of the dishonest-majority MPC protocol.

If however an abort occurs while evaluating the ith sub-circuit then we determine a pair of parties from the committee $\mathcal{C}$, one of whom is guaranteed to be corrupted and eliminate the pair from the set of active parties, and re-evaluate the sub-circuit again. In fact, cheating can also occur in the $\langle\cdot\rangle \leftrightarrow [\cdot]$ conversions and we need to deal with these as well. Thus if errors are detected we need to repeat the evaluation of the sub-circuit at level i. Since there are at most t bad parties, the total amount of backtracking (i.e. evaluating a sub-circuit already computed) that needs to be done is bounded by t. For large n and small t this provides an asymptotically efficient protocol.

The main technical difficulty is in providing actively-secure conversions between the two secret-sharing schemes, and providing a suitable party-elimination strategy for the dishonest majority MPC protocol. The party-elimination strategy we employ follows from standard techniques, as long as we can identify the pair of parties. This

[4] The communication complexity of an MPC protocol has two parts: a circuit-dependent part, dependent on the circuit size and a circuit-independent part. The focus is on the circuit-dependent communication, based on the assumption that the circuit is large enough so that the terms independent of the circuit-size can be ignored; see for example [10, 4, 11, 5].

[5] In the dishonest-majority setting, the adversary may corrupt all but one parties. An MPC protocol in this setting aborts if a corrupted party misbehaves.

requirement, of a dishonest-majority MPC protocol which enables identification of cheaters, without sacrificing privacy, leads us to the utilization of the protocol in [11]. This results in us needing to use double-trapdoor homomorphic commitments as a basic building block. To ensure greater asymptotic efficiency we apply two techniques: **(a).** the check-pointing is done among a set of parties that assures honest majority with overwhelming probability **(b).** the packing technique from [16] to our Shamir based secret sharing.

To obtain an efficient protocol one needs to select L; if L is too small then the sub-circuits are large and so the cost of returning to a prior checkpoint will also be large. If however L is too large then we will need to checkpoint a lot, and hence involve all n parties in the computation at a lot of stages (and thus requiring all n parties to be communicating/computing). The optimal value of L for our protocol turns out to be t.

Related Work: The circuit-dependent communication complexity of the traditional MPC protocols in the honest-majority setting is $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{Poly}(n, \kappa))$; this informally stems from the fact in these protocols we require all the n parties to communicate with each other for evaluating each gate of the circuit. Assuming $0 \leq \epsilon < 1/2$, [10] presents a computationally secure MPC protocol with communication complexity $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{Poly}(\kappa, \log n, \log |\mathsf{ckt}|))$. The efficiency comes from the ability to pack and share several values simultaneously which in turn allow parallel evaluation of "several" gates simultaneously in a single round of communication. However, the protocol still requires communications between all the parties during each round of communication. Our protocol reduces the need for the parties to be communicating with all others at all stages in the protocol; moreover, asymptotically for large n it provides a better communication complexity over [10] (as there is no dependence on n), for circuits of suitably large depth as stated earlier. However, the protocol of [10] is secure against a more powerful adaptive adversary.

In the literature, another line of investigation has been carried out in [6, 9, 12, 13] to beat the $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{Poly}(n, \kappa))$ communication complexity bound of traditional MPC protocols, against a static adversary. The main idea behind all these works is similar to ours, which is to involve "small committees" of parties for evaluating each gate of the circuit, rather than involving all the n parties. The communication complexity of these protocols[6] is of the order $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{Poly}(\log n, \kappa))$. Technically our protocol is different from these protocols in the following ways: **(a).** The committees in [6, 9, 12, 13] are of size $\mathsf{Poly}(\log n)$, which ensures that with high probability the selected committees have honest majority. As a result, these protocols run any existing honest-majority MPC protocol among these small committees of $\mathsf{Poly}(\log n)$ size, which prevents the need to check-point the computation (as there will be no aborts). On the other hand, we only require committees with at least one honest party and our committee size is independent of n, thus providing better communication complexity. Indeed, asymptotically for large n, our protocol provides a better communication complexity over [6, 9, 12, 13] (as there is no dependence on n), for circuits of suitably large depth. **(b).** Our protocol provides

[6] Note, the protocol of [6] involves FHE to further achieve a communication complexity of $\mathcal{O}(\mathsf{Poly}(\log n))$.

a better fault-tolerance. Specifically, [12, 9, 6] requires $\epsilon < 1/3$ and [13] requires $\epsilon < 1/8$; on the other hand we require $\epsilon < 1/2$.

We stress that the committee selection protocol in [6, 9, 12, 13] is unconditionally secure and in the full-information model, where the corrupted parties can see all the messages communicated between the honest parties. On the other hand our implementation of the committee selection protocol is computationally secure. The committee election protocol in [6, 9, 12, 13] is inherited from [14]. The committee selection protocol in these protocols are rather involved and not based on simply randomly selecting a subset of parties, possibly due to the challenges posed in the full information model with unconditional security; this causes their committee size to be logarithmic in n. However, if one is willing to relax at least one of the above two features (i.e. full information model and unconditional security), then it may be possible to select committees with honest majority in a simple way by randomly selecting committees, where the committee size may be independent of n. However investigating the same is out of the scope of this paper.

Finally we note that the idea of using small committees has been used earlier in the literature for various distributed computing tasks, such as the leader election [17, 20], Byzantine agreement [18, 19] and distributed key-generation [8].

On the Choice of ϵ: We select committees of size $\mathfrak{c}$ satisfying $\epsilon^{\mathfrak{c}} < 2^{-\kappa}$. This implies that the selected committee has at least one honest participant with overwhelming probability. We note that it is possible to randomly select committees of "larger" size so that with overwhelming probability the selected committee will have honest majority. We label the protocol which samples a committee with honest majority and then runs an computationally secure honest majority MPC protocol (where we need not have to worry about aborts) as the "naive protocol". The naive protocol will have communication complexity $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{Poly}(\kappa))$.

For "very small" values of ϵ, the committee size for the naive protocol is comparable to the committee size in our protocol. We demonstrate this with an example, with $n = 1000$ and security level $\kappa = 80$: The committee size we require to ensure both a single honest party in the committee and a committee with honest majority, with overwhelming probability of $(1 - 2^{-80})$ for various choices of ϵ, is given in the following table:

ϵ	$\mathfrak{c}$ to obtain at least one honest party	$\mathfrak{c}$ to obtain honest majority
$1/3$	48	448
$1/4$	39	250
$1/10$	23	84
$1/100$	11	20

From the table it is clear that when ϵ is closer to $1/2$, the difference in the committee size to obtain at least one honest party and to obtain honest majority is large. As a result, selecting committees with honest majority can be prohibitively expensive, thus our selection of small committees with dishonest majority provides significant improvements.

To see intuitively why our protocol selects smaller committees, consider the case when the security parameter κ tends to infinity: Our protocol will require a committee of size roughly $\epsilon \cdot n + 1$, whereas the naive protocol will require a committee of size

roughly $2 \cdot \epsilon \cdot n + 1$. Thus the naive method will use a committee size of roughly twice that of our method. Hence, if small committees are what is required then our method improves on the naive method.

For fixed ϵ and increasing n, we can apply the binomial approximation to the hypergeometric distribution, and see that our protocol will require a committee of size $c \approx \kappa / \log_2(\frac{1}{\epsilon})$. To estimate the committee size for the naive protocol we use the cumulative distribution function for the binomial distribution, $F(b; c, \epsilon)$, which gives the probability that we select at least b corrupt parties in a committee of size c given the probability of a corrupt party being fixed at ϵ. To obtain an honest majority with probability less than $2^{-\kappa}$ we require $F(c/2; c, \epsilon) \approx 2^{-\kappa}$. By estimating $F(c/2; c, \epsilon)$ via Hoeffding's inequality we obtain

$$\exp\left(-2 \cdot \frac{(c \cdot \epsilon - c/2)^2}{c}\right) \approx 2^{-\kappa},$$

which implies

$$\kappa \approx \left(\frac{c \cdot (2 \cdot \epsilon - 1)^2}{2}\right) / \log_e 2.$$

Solving for c gives us

$$c \approx \frac{2 \cdot \kappa \cdot \log_e 2}{(2 \cdot \epsilon - 1)^2}.$$

Thus for fixed ϵ and large n the number of parties in a committee is $O(\kappa)$ for both our protocol, and the naive protocol. Thus the communication complexity of our protocol and the naive protocol is asymptotically the same. But, since the committees in our protocol are always smaller than those in the naive protocol, we will obtain an advantage when the ratio of the different committee size is large, i.e. when ϵ is larger.

The the ratio between the committee size in the naive protocol and that of our protocol (assuming we are in a range when Hoeffding's inequality provides a good approximation) is roughly

$$\frac{-2 \cdot \log_e 2 \cdot \log_2 \epsilon}{(2 \cdot \epsilon - 1)^2}$$

So for large n the ratio between the committee sizes of the two protocols depends on ϵ alone (and is independent of κ). By way of example this ratio is approximately equal to 159 when $\epsilon = 0.45$, 19 when $\epsilon = 1/3$, 7 when $\epsilon = 1/10$ and 9.6 when $\epsilon = 1/100$; although the approximation via Hoeffding's inequality only really applies for ϵ close to $1/2$.

This implies that for values of ϵ close to $1/2$ our protocol will be an improvement on the naive protocol. However, the naive method does not have the extra cost of checkpointing which our method does; thus at some point the naive protocol will be more efficient. Thus our protocol is perhaps more interesting, when ϵ is not too small, say in the range of $[1/100, 1/2]$.

Possible Application of Our Protocol for Cloud-Computing. Consider the situation of an organization performing a multi-party computation on a cloud infrastructure, which involves a large number of machines, with the number of corrupted parties possibly high, but not exceeding one half of the parties, (which is exactly the situation

considered in our MPC protocol). Using our MPC protocol, the whole computation can be then carried out by a small subset of machines, with the whole cloud infrastructure being used only for check-pointing the computation. By not utilizing the whole cloud infrastructure all the time, we enable the cloud provider to serve multiple MPC requests.

Our protocol is not adaptively secure. In fact, vulnerability to adaptive adversary is inherent to most of the committee-based protocols for several distributed computing tasks such as Leader Election [17, 20], Byzantine Agreement [19, 18], Distributed Key-generation [8] and MPC in [12, 9]. Furthermore, We feel that adaptive security is not required in the cloud scenario. Any external attacker to the cloud data centre will have a problem determining which computers are being used in the committee, and an even greater problem in compromising them adaptively. The main threat model in such a situation is via co-tenants (other users processes) to be resident on the same physical machine. Since the precise machine upon which a cloud tenant sits is (essentially) randomly assigned, it is hard for a co-tenant adversary to mount a cross-Virtual Machine attack on a specific machine unless they are randomly assigned this machine by the cloud. Note, that co-tenants have more adversarial power than a completely external attacker. A more correct security model would be to have a form of adaptive security in which attackers pro-actively move from one machine to another, but in a random fashion. We leave analysing this complex situation to a future work.

2 Model, Notation and Preliminaries

We denote by $\mathcal{P} = \{P_1, \ldots, P_n\}$ the set of n parties who are connected by pair-wise private and authentic channels. We assume that there exists a PPT static adversary $\mathcal{A}$, who can maliciously corrupt any t parties from $\mathcal{P}$ at the beginning of the execution of a protocol, where $t = n \cdot \epsilon$ and $0 \leq \epsilon < \frac{1}{2}$. There exists a publicly known randomized function $f : \mathbb{F}_p^n \rightarrow \mathbb{F}_p$, expressed as a publicly known arithmetic circuit ckt over the field $\mathbb{F}_p$ of prime order p (including random gates to enable the evaluation of randomized functions), with party P_i having a private input $x^{(i)} \in \mathbb{F}_p$ for the computation. We let d and w to denote the depth and (average) width of ckt respectively. The finite field $\mathbb{F}_p$ is assumed to be such that p is a prime, with $p > \max\{n, 2^\kappa\}$, where κ is the *computational security parameter*. Apart from κ, we also have an additional *statistical security parameter* s and the security offered by s (which is generally much smaller than κ) does not depend on the computational power of the adversary.

The security of our protocol(s) will be proved in the universal composability (UC) model. The UC framework allows for defining the security properties of cryptographic tasks so that security is maintained under general composition with an unbounded number of instances of arbitrary protocols running concurrently. In the framework, the security requirements of a given task are captured by specifying an ideal functionality run by a "trusted party" that obtains the inputs of the parties and provides them with the desired outputs. Informally, a protocol securely carries out a given task if running the protocol in the presence of a real-world adversary amounts to "emulating" the desired functionality. For more details, see the full version of this paper.

We do not assume a physical broadcast channel. Although our protocol uses an ideal broadcast functionality $\mathcal{F}_{\text{BC}}$ (Fig. 3), that allows a sender $\mathsf{Sen} \in \mathcal{P}$ to reliably broadcast

a message to a group of parties $\mathcal{X} \subseteq \mathcal{P}$, the functionality can be instantiated using point-to-point channels; see the full version of this paper for details.

The communication complexity of our protocols has two parts: the communication done over the point-to-point channels and the broadcast communication. The later is captured by $\mathcal{BC}(\ell, |\mathcal{X}|)$ to denote that in total, $\mathcal{O}(\ell)$ bits is broadcasted in the associated protocol to a set of parties of size $|\mathcal{X}|$.

Two different types of secret-sharing are employed in our protocols. The secret-sharings are inherently defined to include "verification information" of the individual shares in the form of publicly known commitments. We use a variant of the Pedersen homomorphic commitment scheme [21]. In our protocol, we require UC-secure commitments to ensure that a committer must know its committed value and just cannot manipulate a commitment produced by other committers to violate what we call "input independence". It has been shown in [7] that a UC secure commitment scheme is impossible to achieve without setup assumptions. The standard method to implement UC-secure commitments is in the Common Reference String (CRS) model where it is assumed that the parties are provided with a CRS that is set up by a "trusted third party" (TTP). We follow [11], where the authors show how to build a multiparty UC-secure homomorphic commitment scheme (where multiple parties can act as committer) based on any double-trapdoor homomorphic commitment scheme.

Definition 1 (Double-trapdoor Homomorphic Commitment for $\mathbb{F}_p$ [11]). *It is a collection of five PPT algorithms* $(\mathsf{Gen}, \mathsf{Comm}, \mathsf{Open}, \mathsf{Equivocate}, \mathsf{TDExtract}, \odot)$*:*

- $\mathsf{Gen}(1^\kappa) \rightarrow (\mathsf{ck}, \tau_0, \tau_1)$*: the generation algorithm outputs a commitment key* ck*, along with trapdoors* τ_0 *and* τ_1*.*
- $\mathsf{Comm}_{\mathsf{ck}}(x; r_0, r_1) \rightarrow \mathbf{C}_{x,r_0,r_1}$*: the commitment algorithm takes a message* $x \in \mathbb{F}_p$ *and randomness* r_0, r_1 *from the commitment randomness space* $\mathcal{R}$ [7] *and outputs a commitment* $\mathbf{C}_{x;r_0,r_1}$ *of* x *under the randomness* r_0, r_1*.*
- $\mathsf{Open}_{\mathsf{ck}}(\mathbf{C}, (x; r_0, r_1)) \rightarrow \{0, 1\}$*: the opening algorithm takes a commitment* $\mathbf{C}$*, along with a message/randomness triplet* (x, r_0, r_1) *and outputs* 1 *if* $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}($ $x; r_0, r_1)$*, else* 0*.*
- $\mathsf{Equivocate}(\mathbf{C}_{x,r_0,r_1}, x, r_0, r_1, \overline{x}, \tau_i) \rightarrow (\overline{r}_0, \overline{r}_1) \in \mathcal{R}$*: using one of the trapdoors* τ_i *with* $i \in \{0, 1\}$*, the equivocation algorithm can open a commitment* $\mathbf{C}_{x,r_0,r_1}$ *with any message* $\overline{x} \neq x$ *with randomness* $\overline{r}_0$ *and* $\overline{r}_1$ *where* $r_{1-i} = \overline{r}_{1-i}$*.*
- $\mathsf{TDExtract}(\mathbf{C}, x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1, \tau_i) \rightarrow \tau_{1-i}$*: using one of the trapdoors* τ_i *with* $i \in \{0, 1\}$ *and two different sets of message/randomness triplet for the same commitment, namely* x, r_0, r_1 *and* $\overline{x}, \overline{r}_0, \overline{r}_1$*, the trapdoor extraction algorithm can find the other trapdoor* τ_{1-i} *if* $r_{1-i} \neq \overline{r}_{1-i}$*.*
  The commitments are homomorphic meaning that $\mathsf{Comm}(x; r_0, r_1) \odot \mathsf{Comm}(y;$ $s_0, s_1) = \mathsf{Comm}(x+y; r_0+s_0, r_1+s_1)$ *and* $\mathsf{Comm}(x; r_0, r_1)^c = \mathsf{Comm}(c \cdot x; c \cdot r_0,$ $c \cdot r_1)$ *for any publicly known constant* c*.*

We require the following properties to be satisfied:

- **Trapdoor Security***: There exists no PPT algorithm* A *such that* $A(1^\kappa, \mathsf{ck}, \tau_i) \rightarrow \tau_{1-i}$*, for* $i \in \{0, 1\}$*.*

[7] For the ease of presentation, we assume $\mathcal{R}$ to be an additive group.

- **Computational Binding**: *There exists no PPT algorithm A with $A(1^\kappa, \mathsf{ck}) \to (x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1)$ and $(x, r_0, r_1) \neq (\overline{x}, \overline{r}_0, \overline{r}_1)$, but $\mathsf{Comm}_{\mathsf{ck}}(x; r_0, r_1) = \mathsf{Comm}_{\mathsf{ck}}(\overline{x}; \overline{r}_0, \overline{r}_1)$.*
- **Statistical Hiding**: *$\forall x, \overline{x} \in \mathbb{F}_p$ and $r_0, r_1 \in \mathcal{R}$, let $(\overline{r}_0, \overline{r}_1) = \mathsf{Equivocate}(\mathbf{C}_{x,r_0,r_1}, x, r_0, r_1, \overline{x}, \tau_i)$, with $i \in \{0, 1\}$. Then $\mathsf{Comm}_{\mathsf{ck}}(x; r_0, r_1) = \mathsf{Comm}_{\mathsf{ck}}(\overline{x}; \overline{r}_0, \overline{r}_1) = \mathbf{C}_{x,r_0,r_1}$; moreover the distribution of (r_0, r_1) and $(\overline{r}_0, \overline{r}_1)$ are statistically close.*

We will use the following instantiation of a double-trapdoor homomorphic commitment scheme which is a variant of the standard Pedersen commitment scheme over a group $\mathbb{G}$ in which discrete logarithms are hard [11]. The message space is $\mathbb{F}_p$ and the randomness space is $\mathcal{R} = \mathbb{F}_p^2$.

- $\mathsf{Gen}(1^\kappa) \to ((\mathbb{G}, p, g, h_0, h_1), \tau_0, \tau_1)$, where $\mathsf{ck} = (\mathbb{G}, p, g, h_0, h_1)$ such that g, h_0, h_1 are generators of the group $\mathbb{G}$ of prime order p and $g^{\tau_i} = h_i$ for $i \in \{0, 1\}$.
- $\mathsf{Comm}_{\mathsf{ck}}(x; r_0, r_1) \to g^x h_0^{r_0} h_1^{r_1} = \mathbf{C}_{x,r_0,r_1}$, with $x, r_0, r_1 \in \mathbb{F}_p$.
- $\mathsf{Open}_{\mathsf{ck}}(\mathbf{C}, (x, r_0, r_1)) \to 1$, if $\mathbf{C} = g^x h_0^{r_0} h_1^{r_1}$, else $\mathsf{Open}_{\mathsf{ck}}(\mathbf{C}, (x, r_0, r_1)) \to 0$.
- $\mathsf{Equivocate}(\mathbf{C}_{x,r_0,r_1}, x, r_0, r_1, \overline{x}, \tau_i) \to (\overline{r}_0, \overline{r}_1)$ where $\overline{r}_{1-i} = r_{1-i}$ and $\overline{r}_i = \tau_i^{-1}(x - \overline{x}) + r_i$.
- $\mathsf{TDExtract}(\mathbf{C}, x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1, \tau_i) \to \tau_{1-i}$, where if $\overline{r}_{1-i} \neq r_{1-i}$, then

$$\tau_{1-i} = \frac{\overline{x} - x + \tau_i(\overline{r}_i - r_i)}{r_{1-i} - \overline{r}_{1-i}}.$$

- The homomorphic operation $\odot$ is just the group operation i.e.

$$\begin{aligned}\mathsf{Comm}(x; r_0, r_1) \odot \mathsf{Comm}(\overline{x}; \overline{r}_0, \overline{r}_1) &= g^x h_0^{r_0} h_1^{r_1} \cdot g^{\overline{x}} h_0^{\overline{r}_0} h_1^{\overline{r}_1} \\ &= g^{x+\overline{x}} \cdot h_0^{r_0+\overline{r}_0} \cdot h_1^{r_1+\overline{r}_1} \\ &= \mathsf{Comm}(x + \overline{x}; r_0 + \overline{r}_0, r_1 + \overline{r}_1).\end{aligned}$$

We can now define the various types of secret-shared data used in our protocols. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_p$ be n publicly known non-zero, distinct values, where α_i is associated with P_i as the *evaluation point*. The $[\cdot]$ sharing is the standard Shamir-sharing [22], where the secret value will be shared among the set of parties $\mathcal{P}$ with threshold t. Additionally, a commitment of each individual share will be available publicly, with the corresponding share-holder possessing the randomness of the commitment.

Definition 2 (The $[\cdot]$ Sharing). *Let $s \in \mathbb{F}_p$; then s is said to be $[\cdot]$-shared among $\mathcal{P}$ if there exist polynomials, say $f(\cdot), g(\cdot)$ and $h(\cdot)$, of degree at most t, with $f(0) = s$ and every (honest) party $P_i \in \mathcal{P}$ holds a share $f_i = f(\alpha_i)$ of s, along with opening information $g_i = g(\alpha_i)$ and $h_i = h(\alpha_i)$ for the commitment $\mathbf{C}_{f_i,g_i,h_i} = \mathsf{Comm}_{\mathsf{ck}}(f_i; g_i, h_i)$. The information available to party $P_i \in \mathcal{P}$ as part of the $[\cdot]$-sharing of s is denoted by $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$. All parties will also have the access to ck. Moreover, the collection of $[s]_i$'s, corresponding to $P_i \in \mathcal{P}$ is denoted by $[s]$.*

The second type of secret-sharing (which is a variation of additive sharing), is used to perform computation via a dishonest majority MPC protocol amongst our committees.

Definition 3 (The $\langle\cdot\rangle$ Sharing). *A value $s \in \mathbb{F}_p$ is said to be $\langle\cdot\rangle$-shared among a set of parties $\mathcal{X} \subseteq \mathcal{P}$, if every (honest) party $P_i \in \mathcal{X}$ holds a share s_i of s along with the opening information u_i, v_i for the commitment $\mathbf{C}_{s_i,u_i,u_i} = \mathsf{Comm}_{\mathsf{ck}}(s_i; u_i, v_i)$, such that $\sum_{P_i \in \mathcal{X}} s_i = s$. The information available to party $P_i \in \mathcal{X}$ as part of the $\langle\cdot\rangle$-sharing of s is denoted by $\langle s\rangle_i = (s_i, u_i, v_i, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$. All parties will also have access to* ck*. The collection of $\langle s\rangle_i$'s corresponding to $P_i \in \mathcal{X}$ is denoted by $\langle s\rangle_{\mathcal{X}}$.*

It is easy to see that both types of secret-sharing are linear. For example, for the $\langle\cdot\rangle$ sharing, given $\langle s^{(1)}\rangle_{\mathcal{X}}, \ldots, \langle s^{(\ell)}\rangle_{\mathcal{X}}$ and publicly known constants $c_1, \ldots, c_\ell$, the parties in $\mathcal{X}$ can locally compute their information corresponding to $\langle c_1 \cdot s^{(1)} + \ldots + c_\ell \cdot s^{(\ell)}\rangle_{\mathcal{X}}$. This follows from the homomorphic property of the underlying commitment scheme and the linearity of the secret-sharing scheme. This means that the parties in $\mathcal{X}$ can locally compute $\langle c_1 \cdot s^{(1)} + \ldots + c_\ell \cdot s^{(\ell)}\rangle_{\mathcal{X}}$ from $\langle s^{(1)}\rangle_{\mathcal{X}}, \ldots, \langle s^{\ell}\rangle_{\mathcal{X}}$, since each party P_i in $\mathcal{X}$ can locally compute $\langle c_1 \cdot s^{(1)} + \ldots + c_\ell \cdot s^{(\ell)}\rangle_i$ from $\langle s^{(1)}\rangle_i, \ldots, \langle s^{\ell}\rangle_i$.

3 Main Protocol

We now present an MPC protocol implementing the standard honest-majority (meaning $\epsilon < 1/2$) MPC functionality $\mathcal{F}_f$ presented in Figure 1 which computes the function f.

Functionality $\mathcal{F}_f$

$\mathcal{F}_f$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$ and is parametrized by an n-input function $f : \mathbb{F}_p^n \to \mathbb{F}_p$.

- Upon receiving $(\mathsf{sid}, i, x^{(i)})$ from every $P_i \in \mathcal{P}$ where $x^{(i)} \in \mathbb{F}_p$, the functionality computes $y = f(x^{(1)}, \ldots, x^{(n)})$, sends (sid, y) to all the parties and the adversary $\mathcal{S}$ and halts.

Fig. 1. The Ideal Functionality for Computing a Given Function f

We now present the underlying idea of our protocol (outlined earlier in the introduction). The protocol is set in a variant of the *player-elimination* framework from [4]. During the computation either pairs of parties, each containing at least one actively corrupted party, or singletons of corrupted parties, are identified due to some adversarial behavior of the corrupted parties. These pairs, or singletons, are then eliminated from the set of eligible parties. To understand how we deal with the active corruptions, we need to define a dynamic set $\mathcal{L} \subseteq \mathcal{P}$ of size $\mathfrak{n}$, which will define the current set of eligible parties in our protocol, and a threshold $\mathfrak{t}$ which defines the maximum number of corrupted parties in $\mathcal{L}$. Initially $\mathcal{L}$ is set to be equal to $\mathcal{P}$ (hence $\mathfrak{n} = n$) and $\mathfrak{t}$ is set to t. We then divide the circuit ckt (representing f) to be evaluated into L levels, where each level consists of a sub-circuit of depth d/L; without loss of generality, we assume d to be a multiple of L. We denote the ith sub-circuit as ckt_i. At the beginning of the protocol, all the parties in $\mathcal{P}$ verifiably $[\cdot]$-share their inputs for the circuit ckt.

For evaluating a sub-circuit ckt_l, instead of involving all the parties in $\mathcal{L}$, we rather involve a small and random committee $\mathcal{C} \subset \mathcal{L}$ of parties of size $\mathfrak{c}$, where $\mathfrak{c}$ is the minimum value satisfying the constraint that $\epsilon^{\mathfrak{c}} \leq 2^{-\kappa}$; recall $\epsilon = t/n$. During the course

of evaluating the sub-circuit, if any inconsistency is reported, then the (honest) parties in $\mathcal{P}$ will identify either a single corrupted party or a pair of parties from $\mathcal{L}$ where the pair contains at least one corrupted party. The identified party(ies) is(are) eliminated from $\mathcal{L}$ and the value of $\mathfrak{t}$ is decremented by one, followed by re-evaluation of ckt_l by choosing a new committee from the *updated* set $\mathcal{L}$. This is reminiscent of the player-elimination framework from [4], however the way we apply the player-elimination framework is different from the standard one. Specifically, in the player-elimination framework, the *entire* set of eligible parties $\mathcal{L}$ is involved in the computation and the player elimination is then performed over the entire $\mathcal{L}$, thus requiring huge communication. On the contrary, in our context, only a small set of parties $\mathcal{C}$ is involved in the computation, thus significantly reducing the communication complexity. It is easy to see that after a sequence of t failed sub-circuit evaluations, $\mathcal{L}$ will be left with only honest parties and so each sub-circuit will be evaluated successfully from then onwards.

Note that the way we eliminate the parties, the fraction of corrupted parties in $\mathcal{L}$ after any un-successful attempt for sub-circuit evaluation, is upper bounded by the fraction of corrupted parties in $\mathcal{L}$ prior to the evaluation of the sub-circuit. Specifically, let $\epsilon_{\mathsf{old}} = \mathfrak{t}/\mathfrak{n}$ be the fraction of corrupted parties in $\mathcal{L}$ prior to the evaluation of a sub-circuit ckt_l and let the evaluation fail, with either a single party or a pair of parties being eliminated from $\mathcal{L}$. Moreover, let ϵ_{new} be the fraction of corrupted parties in $\mathcal{L}$ after the elimination. Then for single elimination, we have $\epsilon_{\mathsf{new}} = \frac{\mathfrak{t}-1}{\mathfrak{n}-1}$ and so $\epsilon_{\mathsf{new}} \leq \epsilon_{\mathsf{old}}$ if and only if $\mathfrak{n} \geq \mathfrak{t}$, which will always hold. On the other hand, for double elimination, we have $\epsilon_{\mathsf{new}} = \frac{\mathfrak{t}-1}{\mathfrak{n}-2}$ and so $\epsilon_{\mathsf{new}} \leq \epsilon_{\mathsf{old}}$ if and only if $\mathfrak{n} \geq 2\mathfrak{t}$, which will always hold.

Since a committee $\mathcal{C}$ (for evaluating a sub-circuit) is selected randomly, except with probability at most $\epsilon^{\mathfrak{c}} < 2^{-\kappa}$, the selected committee contains at least one honest party and so the sub-circuit evaluation among $\mathcal{C}$ needs to be performed via a dishonest majority MPC protocol. We choose the MPC protocol of [11], since it can be modified to identify pairs of parties consisting of at least one corrupted party in the case of the failed evaluation, without violating the privacy of the honest parties. To use the protocol of [11] for sub-circuit evaluation, we need the corresponding sub-circuit inputs (available to the parties in $\mathcal{P}$ in $[\cdot]$-shared form) to be converted and available in $\langle\cdot\rangle$-shared form to the parties in $\mathcal{C}$ and so the parties in $\mathcal{P}$ do the same. After every successful evaluation of a sub-circuit, via the dishonest majority MPC protocol, the outputs of that sub-circuit (available in $\langle\cdot\rangle$-shared form to the parties in a committee) are converted and saved in the form of $[\cdot]$-sharing among all the parties in $\mathcal{P}$. As the set $\mathcal{P}$ has a honest majority, $[\cdot]$-sharing ensures robust reconstruction implying that the shared values are recoverable. Since the inputs to a sub-circuit come either from the outputs of previous sub-circuit evaluations or the original inputs, both of which are $[\cdot]$-shared, a failed attempt for a sub-circuit evaluation does not require a re-evaluation of the entire circuit from scratch but requires a re-evaluation of that sub-circuit only.

3.1 Supporting Functionalities

We now present a number of ideal functionalities defining sub-components of our main protocol; see the full version for the UC-secure instantiations of these functionalities.

Basic Functionalities: The functionality $\mathcal{F}_{\text{CRS}}$ for generating the common reference string (CRS) for our main MPC protocol is given in Figure 2. The functionality outputs the commitment key of a double-trapdoor homomorphic commitment scheme, along with the encryption key of an IND-CCA secure encryption scheme (to be used later for UC-secure generation of completely random $\langle\cdot\rangle$-shared values as in [11]). The functionality $\mathcal{F}_{\text{BC}}$ for group broadcast is given in Figure 3. This functionality broadcasts the message sent by a sender $\mathsf{Sen} \in \mathcal{P}$ to all the parties in a sender specified set of parties $\mathcal{X} \subseteq \mathcal{P}$; in our context, the set $\mathcal{X}$ will always contain at least one honest party. The functionality $\mathcal{F}_{\text{COMMITTEE}}$ for a random committee selection is given in Figure 4. This functionality is parameterized by a value $\mathfrak{c}$, it selects a set $\mathcal{X}$ of $\mathfrak{c}$ parties at random from a specified set $\mathcal{Y}$ and outputs the selected set $\mathcal{X}$ to the parties in $\mathcal{P}$.

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

$\mathcal{F}_{\text{CRS}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$ and is parameterized by κ.

- Upon receiving (sid, i) from every party $P_i \in \mathcal{P}$, the functionality computes $\mathsf{Gen}(1^\kappa) \rightarrow (\mathsf{ck}, \tau_0, \tau_1)$ and $G(1^\kappa) \rightarrow (\mathsf{pk}, \mathsf{sk})$, where G is the key-generation algorithm of an IND-CCA secure encryption scheme[a] and Gen is the key-generation algorithm of a double-trapdoor homomorphic commitment scheme. The functionality then sets $\mathsf{CRS} = (\mathsf{ck}, \mathsf{pk})$ and sends $(\mathsf{sid}, i, \mathsf{CRS})$ to every party $P_i \in \mathcal{P}$ and the adversary $\mathcal{S}$ and halts.

[a] For use in the protocol of [11]

Fig. 2. The Ideal Functionality for Generating CRS

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

$\mathcal{F}_{\text{BC}}$ interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.

- Upon receiving $(\mathsf{sid}, \mathsf{Sen}, x, \mathcal{X})$ from the sender $\mathsf{Sen} \in \mathcal{P}$ such that $\mathcal{X} \subseteq \mathcal{P}$, the functionality sends $(\mathsf{sid}, j, \mathsf{Sen}, x)$ to every $P_j \in \mathcal{X}$ and to the adversary $\mathcal{S}$ and halts.

Fig. 3. The Ideal Functionality for Broadcast

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

$\mathcal{F}_{\text{COMMITTEE}}$, parametrized by a constant $\mathfrak{c}$, interacts with the parties in $\mathcal{P}$ and the adversary $\mathcal{S}$.

- Upon receiving $(\mathsf{sid}, i, \mathcal{Y})$ from every $P_i \in \mathcal{P}$, the functionality selects $\mathfrak{c}$ parties at random from the set $\mathcal{Y}$ that is received from the majority of the parties and denotes the selected set as $\mathcal{X}$. The functionality then sends $(\mathsf{sid}, i, \mathcal{X})$ to every $P_i \in \mathcal{P}$ and $\mathcal{S}$ and halts.

Fig. 4. The Ideal Functionality for Selecting a Random Committee of Given Size $\mathfrak{c}$

Functionality Related to $[\cdot]$-sharings: In Figure 5 we present the functionality $\mathcal{F}_{\text{GEN}[\cdot]}$ which allows a dealer $\mathsf{D} \in \mathcal{P}$ to verifiably $[\cdot]$-share an already committed secret among the parties in $\mathcal{P}$. The functionality is invoked when it receives three polynomials, say $f(\cdot), g(\cdot)$ and $h(\cdot)$ from the dealer D and a commitment, say $\mathbf{C}$, supposedly the commitment of $f(0)$ with randomness $g(0), h(0)$ (namely $\mathbf{C}_{f(0),g(0),h(0)}$), from the (majority of the) parties in $\mathcal{P}$. The functionality then hands $f_i = f(\alpha_i), g_i = g(\alpha_i), h_i = h(\alpha_i)$ and commitments $\{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}}$ to $P_i \in \mathcal{P}$ after 'verifying' that **(a):** All the three polynomials are of degree at most t and **(b):** $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(f(0); g(0), h(0))$ i.e. the value (and the corresponding randomness) committed in $\mathbf{C}$ are embedded in the constant term of $f(\cdot), g(\cdot)$ and $h(\cdot)$ respectively. If either of the above two checks fail, then the functionality returns $\mathsf{Failure}$ to the parties indicating that D is corrupted.

In our MPC protocol where $\mathcal{F}_{\text{GEN}[\cdot]}$ is called, the dealer will compute the commitment $\mathbf{C}$ as $\mathsf{Comm}_{\mathsf{ck}}(f(0); g(0), h(0))$ and will broadcast it prior to making a call to $\mathcal{F}_{\text{GEN}[\cdot]}$. It is easy to note that $\mathcal{F}_{\text{GEN}[\cdot]}$ generates $[f(0)]$ if D is honest or well-behaved. If $\mathcal{F}_{\text{GEN}[\cdot]}$ returns $\mathsf{Failure}$, then D is indeed corrupted.

Functionality $\mathcal{F}_{\text{GEN}[\cdot]}$

$\mathcal{F}_{\text{GEN}[\cdot]}$ interacts with the parties in $\mathcal{P}$, a dealer $\mathsf{D} \in \mathcal{P}$, and the adversary $\mathcal{S}$ and is parametrized by a commitment key ck of a double-trapdoor homomorphic commitment scheme, along with t.

- On receiving $(\mathsf{sid}, \mathsf{D}, f(\cdot), g(\cdot), h(\cdot))$ from D and $(\mathsf{sid}, i, \mathsf{D}, \mathbf{C})$ from every $P_i \in \mathcal{P}$, the functionality verifies whether $f(\cdot), g(\cdot)$ and $h(\cdot)$ are of degree at most t and $\mathbf{C} \stackrel{?}{=} \mathsf{Comm}_{\mathsf{ck}}(f(0); g(0), h(0))$, where $\mathbf{C}$ is received from the majority of the parties.
- If any of the above verifications fail then the functionality sends $(\mathsf{sid}, i, \mathsf{D}, \mathsf{Failure})$ to every $P_i \in \mathcal{P}$ and $\mathcal{S}$ and halts.
- Else for every $P_i \in \mathcal{P}$, the functionality computes the share $f_i = f(\alpha_i)$, the opening information $g_i = g(\alpha_i), h_i = h(\alpha_i)$, and the commitment $\mathbf{C}_{f_i,g_i,h_i} = \mathsf{Comm}_{\mathsf{ck}}(f_i; g_i, h_i)$. It sends $(\mathsf{sid}, i, \mathsf{D}, [s]_i)$ to every $P_i \in \mathcal{P}$ where $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ and halts.

Fig. 5. The Ideal Functionality for Verifiably Generating $[\cdot]$-sharing

We note that $\mathcal{F}_{\text{GEN}[\cdot]}$ is slightly different from the standard ideal functionality (see e.g. [2]) of verifiable secret sharing (VSS) where the parties output *only* their shares (and not the commitment of all the shares). In most of the standard instantiations of a VSS functionality (in the computational setting), for example the Pedersen VSS [21], a public commitment of all the shares and the secret are available to the parties without violating any privacy. In order to make these commitments available to the external protocol that invokes $\mathcal{F}_{\text{GEN}[\cdot]}$, we allow the functionality to compute and deliver the shares along with the commitments to the parties. We note, [1] introduced a similar functionality for "committed VSS" that outputs to the parties the commitment of the secret provided by the dealer due to the same motivation mentioned above.

3.2 Supporting Sub-protocols

Our MPC protocol also makes use of the following sub-protocols. Due to space constraints, here we only present a high level description of these protocols and state their communication complexity. The formal details of the protocols are available in the full version. Since we later show that our main MPC protocol that invokes these sub-protocols is UC-secure, it is not required to prove any form of security for these sub-protocols separately.

(A) Protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$: it takes input $\langle s\rangle_{\mathcal{X}}$ for a set $\mathcal{X}$ containing at least *one* honest party and either produces a sharing $[s]$ (if all the parties in $\mathcal{X}$ behave honestly) or outputs one of the following: the identity of a single corrupted party or a pair of parties (with at least one of them being corrupted) from $\mathcal{X}$. The protocol makes use of the functionalities $\mathcal{F}_{\text{GEN}[\cdot]}$ and $\mathcal{F}_{\text{BC}}$.

More specifically, let $\langle s\rangle_i$ denote the information (namely the share, opening information and the set of commitments) of party $P_i \in \mathcal{X}$ corresponding to the sharing $\langle s\rangle_{\mathcal{X}}$. To achieve the goal of our protocol, there are two clear steps to perform: *first*, the correct commitment for each share of s corresponding to its $\langle\cdot\rangle_{\mathcal{X}}$-sharing, now available to the parties in $\mathcal{X}$, is to be made available to *all* the parties in $\mathcal{P}$; *second*, each $P_i \in \mathcal{X}$ is required to act as a dealer and verifiably $[\cdot]$-share its already committed share s_i among $\mathcal{P}$. Note that the commitment to s_i is included in the set of commitments that will be already available among $\mathcal{P}$ due to the first step. Clearly, once $[s_i]$ are generated for each $P_i \in \mathcal{X}$, then $[s]$ is computed as $[s] = \sum_{P_i\in\mathcal{X}}[s_i]$; this is because $s = \sum_{P_i\in\mathcal{X}} s_i$.

Now there are two steps that may lead to the failure of the protocol. First, $P_i \in \mathcal{X}$ may be identified as a corrupted dealer while calling $\mathcal{F}_{\text{GEN}[\cdot]}$. In this case a single corrupted party is outputted by every party in $\mathcal{P}$. Second, the protocol may fail when the parties in $\mathcal{P}$ try to reach an agreement over the correct set of commitments of the shares of s. Recall that each $P_i \in \mathcal{X}$ holds a set of commitments as a part of $\langle s\rangle_{\mathcal{X}}$. We ask each $P_i \in \mathcal{X}$ to call $\mathcal{F}_{\text{BC}}$ to broadcast among $\mathcal{P}$ the set of commitments held by him. It is necessary to ask each $P_i \in \mathcal{X}$ to do this as we can not trust any single party from $\mathcal{X}$, since all we know (with overwhelming probability) is that $\mathcal{X}$ contains at least one honest party. Now if the parties in $\mathcal{P}$ receive the same set of commitments from all the parties in $\mathcal{X}$, then clearly the received set is the correct set of commitments and agreement on the set is reached among $\mathcal{P}$. If this does not happen the parties in $\mathcal{P}$ can detect a pair of parties with conflicting sets and output the said pair. It is not hard to see that indeed one party in the pair must be corrupted. To ensure an agreement on the selected pair when there are multiple such conflicting pairs, we assume the existence of a predefined publicly known algorithm to select a pair from the lot (for instance consider the pair (P_a, P_b) with minimum value of $a + n \cdot b$). Intuitively the protocol is secure as the shares of honest parties in $\mathcal{X}$ remain secure.

The communication complexity of protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$ is stated in Lemma 1, which easily follows from the fact that each party in $\mathcal{X}$ needs to broadcast $\mathcal{O}(|\mathcal{X}|\kappa)$ bits to $\mathcal{P}$.

Lemma 1. *The communication complexity of protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$ is $\mathcal{BC}\big(|\mathcal{X}|^2\kappa, n\big)$ plus the complexity of $\mathcal{O}(|\mathcal{X}|)$ invocations to the realization of the functionality $\mathcal{F}_{\text{GEN}[\cdot]}$.*

(B) Protocol $\Pi_{\langle\cdot\rangle}$: the protocol enables a designated party (dealer) $\mathsf{D} \in \mathcal{P}$ to verifiably $\langle\cdot\rangle$-share an already committed secret f among a set of parties $\mathcal{X}$ containing at least one honest party. More specifically, every $P_i \in \mathcal{P}$ holds a (publicly known) commitment $\mathbf{C}_{f,g,h}$. The dealer D holds the secret $f \in \mathbb{F}_p$ and randomness pair (g, h), such that $\mathbf{C}_{f,g,h} = \mathsf{Comm}_{\mathsf{ck}}(f; g, h)$; and the goal is to generate $\langle f\rangle_{\mathcal{X}}$. In the protocol, D first additively shares f as well as the opening information (g, h) among $\mathcal{X}$. In addition, D is also asked to publicly commit each additive-share of f, using the corresponding additive-share of (g, f). The parties can then publicly verify whether indeed D has $\langle\cdot\rangle$-shared the same f as committed in $\mathbf{C}_{f,g,h}$, via the homomorphic property of the commitments. Intuitively f remains private in the protocol for an honest D as there exists at least one honest party in $\mathcal{X}$. Moreover the binding property of the commitment ensures that a potentially corrupted D fails to $\langle\cdot\rangle$-share an incorrect value $f' \neq f$.

If we notice carefully the protocol achieves a little more than $\langle\cdot\rangle$-sharing of a secret among a set of parties $\mathcal{X}$. All the parties in $\mathcal{P}$ hold the commitments to the shares of f, while as per the definition of $\langle\cdot\rangle$-sharing the commitments to shares should be available to the parties in $\mathcal{X}$ alone. A closer look reveals that the public commitments to the shares of f among the parties in $\mathcal{P}$ enable them to publicly verify whether D has indeed $\langle\cdot\rangle$-shared the same f among $\mathcal{X}$ as committed in $\mathbf{C}_{f,g,h}$ via the homomorphic property of the commitments. The communication complexity of $\Pi_{\langle\cdot\rangle}$ is stated in Lemma 2.

Lemma 2. *The communication complexity of protocol $\Pi_{\langle\cdot\rangle}$ is $\mathcal{O}(|\mathcal{X}|\kappa)$ and $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$.*

(C) Protocol $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$: the protocol takes as input $[s]$ for any secret s and outputs $\langle s\rangle_{\mathcal{X}}$ for a designated set of parties $\mathcal{X} \subset \mathcal{P}$ containing at least one honest party.

Let $f_1, \ldots, f_n$ be the Shamir-shares of s. Then the protocol is designed using the following two-stage approach: **(1)**: First each party $P_k \in \mathcal{P}$ acts as a dealer and verifiably $\langle\cdot\rangle$-share's its share f_k via protocol $\Pi_{\langle\cdot\rangle}$; **(2)** Let $\mathcal{H}$ be the set of $|\mathcal{H}| > t + 1$ parties P_k who have correctly $\langle\cdot\rangle$-shared its Shamir-share f_k; without loss of generality, let $\mathcal{H}$ be the set of first $|\mathcal{H}|$ parties in $\mathcal{P}$. Since the original sharing polynomial (for $[\cdot]$-sharing s) has degree at most t with s as its constant term, then there exists publicly known constants (namely the Lagrange's interpolation coefficients) $c_1, \ldots, c_{|\mathcal{H}|}$, such that $s = c_1 f_1 + \ldots + c_{|\mathcal{H}|} f_{|\mathcal{H}|}$. Since corresponding to each $P_k \in \mathcal{H}$ the share f_k is $\langle\cdot\rangle$-shared, it follows easily that each party $P_i \in \mathcal{X}$ can compute $\langle s\rangle_i = c_1\langle f_1\rangle_i + \ldots + c_{|\mathcal{H}|}\langle f_{|\mathcal{H}|}\rangle_i$. The correctness of the protocol follows from the fact that the corrupted parties in $\mathcal{P}$ will fail to $\langle\cdot\rangle$-share an incorrect Shamir-share of s, thanks to the protocol $\Pi_{\langle\cdot\rangle}$. The privacy of s follows from the fact that the Shamir shares of the honest parties in $\mathcal{P}$ remain private, which follows from the privacy of the protocol $\Pi_{\langle\cdot\rangle}$.

The communication complexity of the protocol $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$ is stated in Lemma 3 which follows from the fact that n invocations to $\Pi_{\langle\cdot\rangle}$ are done in the protocol.

Lemma 3. *The communication complexity of $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$ is $\mathcal{O}(n|\mathcal{X}|\kappa)$ and $\mathcal{BC}\big(n|\mathcal{X}|\kappa, n\big)$.*

(D) Protocol $\Pi_{\textbf{RANDZERO}[\cdot]}$: the protocol is used for generating a random $[\cdot]$-sharing of 0. To design the protocol, we also require a standard Zero-knowledge (ZK) functionality $\mathcal{F}_{\text{ZK.BC}}$ to publicly prove a commitment to zero. The functionality is a "prove-and-broadcast" functionality that upon receiving a commitment and witness pair $(\mathbf{C}, (u, v))$ from a designated prover P_j, verifies if $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(0; u, v)$ or not. If so it sends $\mathbf{C}$ to

all the parties. A protocol $\Pi_{\text{ZK.BC}}$ realizing $\mathcal{F}_{\text{ZK.BC}}$ can be designed in the CRS model using standard techniques, with communication complexity $\mathcal{O}(\mathsf{Poly}(n)\kappa)$.

Protocol $\Pi_{\text{RANDZERO}[\cdot]}$ invokes the ideal functionalities $\mathcal{F}_{\text{ZK.BC}}$ and $\mathcal{F}_{\text{GEN}[\cdot]}$. The idea is as follows: Each party $P_i \in \mathcal{P}$ first broadcasts a random commitment of 0 and proves in a zero-knowledge (ZK) fashion that it indeed committed 0. Next P_i calls $\mathcal{F}_{\text{GEN}[\cdot]}$ as a dealer D to generate $[\cdot]$-sharing of 0 that is consistent with the commitment of 0. The parties then locally add the sharings of the dealers who are successful as dealers in their corresponding calls to $\mathcal{F}_{\text{GEN}[\cdot]}$. Since there exists at least one honest party in this set of dealers, the resultant sharing will be indeed a random sharing of 0, see the full version for details. Looking ahead, we invoke $\Pi_{\text{RANDZERO}[\cdot]}$ only once in our main MPC protocol Π_f (more on this later); so we avoid giving details of the communication complexity of the protocol. However assuming standard realization of $\mathcal{F}_{\text{ZK.BC}}$, the protocol has complexity $\mathcal{O}(\mathsf{Poly}(n)\kappa)$.

(E) Dis-honest Majority MPC Protocol : Apart from the above sub-protocols, we use a non-robust, dishonest-majority MPC protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ with the capability of fault-detection. The protocol, allows a designated set of parties $\mathcal{X} \subset \mathcal{P}$, containing at least one honest party, to perform $\langle\cdot\rangle$-shared evaluation of a given circuit C. In case some corrupted party in $\mathcal{X}$ behaves maliciously, the parties in $\mathcal{P}$ identify a pair of parties from $\mathcal{X}$, with at least one of them being corrupted. The starting point of $\Pi_{\mathsf{C}}^{\text{NR}}$ is the dishonest majority MPC protocol of [11], which takes $\langle\cdot\rangle$-shared inputs of a given circuit, from a set of parties, say $\mathcal{X}$, having a dishonest majority. The protocol then achieves the following:

- If all the parties in $\mathcal{X}$ behave honestly, then the protocol outputs $\langle\cdot\rangle$-shared circuit outputs among $\mathcal{X}$.
- Else the honest parties in $\mathcal{X}$ detect misbehaviour by the corrupted parties and abort the protocol.

We observe that for an aborted execution of the protocol of [11], there exists an honest party in $\mathcal{X}$ that can *locally* identify a corrupted party from $\mathcal{X}$, who deviated from the protocol. We exploit this property in $\Pi_{\mathsf{C}}^{\text{NR}}$ to enable the parties in $\mathcal{P}$ identify a pair of parties from $\mathcal{X}$ with at least one of them being corrupted.

Protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ proceeds in two stages, the *preparation stage* and the *evaluation stage*, each involving various other sub-protocols (details available in the full version). In the preparation stage, if all the parties in $\mathcal{X}$ behave honestly, then they jointly generate $\mathsf{C_M} + \mathsf{C_R}$ shared multiplication triples $\{(\langle \mathbf{a}^{(i)}\rangle_{\mathcal{X}}, \langle \mathbf{b}^{(i)}\rangle_{\mathcal{X}}, \langle \mathbf{c}^{(i)}\rangle_{\mathcal{X}})\}_{i=1,\ldots,\mathsf{C_M}+\mathsf{C_R}}$, such that $\mathbf{c}^{(i)} = \mathbf{a}^{(i)} \cdot \mathbf{b}^{(i)}$ and each $(\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \mathbf{c}^{(i)})$ is random and unknown to the adversary; here $\mathsf{C_M}$ and $\mathsf{C_R}$ are the number of multiplication and random gates in C respectively. Otherwise, the parties in $\mathcal{P}$ identify a pair of parties in $\mathcal{X}$, with at least one of them being corrupted.

Assuming that the desired $\langle\cdot\rangle$-shared multiplication triples are generated in the preparation stage, the parties in $\mathcal{X}$ start evaluating C in a shared fashion by maintaining the following standard invariant for each gate of C: *Given $\langle\cdot\rangle$-shared inputs of the gate, the parties securely compute the $\langle\cdot\rangle$-shared output of the gate*. Maintaining the invariant for the linear gates in C does not require any interaction, thanks to the linearity of $\langle\cdot\rangle$-sharing. For a multiplication gate, the parties deploy a preprocessed $\langle\cdot\rangle$-shared

multiplication triple from the preparation stage (for each multiplication gate a different triple is deployed) and use the standard Beaver's trick [3]. While applying Beaver's trick, the parties in $\mathcal{X}$ need to publicly open two $\langle\cdot\rangle$-shared values using a reconstruction protocol $\Pi_{\text{REC}\langle\cdot\rangle}$ (presented in the full version). It may be possible that the opening is non-robust[8], in which case the circuit evaluation fails and the parties in $\mathcal{P}$ identify a pair of parties from $\mathcal{X}$ with at least one of them being corrupted. For a random gate, the parties consider an $\langle\cdot\rangle$-shared multiplication triple from the preparation stage (for each random gate a different triple is deployed) and the first component of the triple is considered as the output of the random gate. The protocol ends once the parties in $\mathcal{X}$ obtain $\langle\cdot\rangle$-shared circuit outputs $\langle y_1\rangle_{\mathcal{X}}, \ldots, \langle y_{\mathsf{out}}\rangle_{\mathcal{X}}$; so no reconstruction is required at the end.

The complete details of $\Pi_{\mathsf{C}}^{\text{NR}}$ is provided in the full version. The protocol invokes two ideal functionalities $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ and $\mathcal{F}_{\text{BC}}$ where the functionality $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ is used to generate $\langle\cdot\rangle$-sharing of random values (again see the full version). For our purpose we note that the protocol provides a statistical security of 2^{-s} and has communication complexity as stated in Lemma 4 and proved in the full version. Note that there are two types of broadcast involved: among the parties in $\mathcal{X}$ and among the parties in $\mathcal{P}$.

Lemma 4. *For a statistical security parameter s, protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ has communication complexity of $\mathcal{O}(|\mathcal{X}|^2(|\mathsf{C}|+s)\kappa), \mathcal{BC}\big(|\mathcal{X}|^2(|\mathsf{C}|+s)\kappa, |\mathcal{X}|\big)$ and $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$.*

3.3 The MPC Protocol

Finally, we describe our MPC protocol. Recall that we divide the circuit ckt into sub-circuits $\mathsf{ckt}_1, \ldots, \mathsf{ckt}_L$ and we let in_l and out_l denote the number of input and output wires respectively for the sub-circuit ckt_l. At the beginning of the protocol, each party $[\cdot]$-share their private inputs by calling $\mathcal{F}_{\text{GEN}[\cdot]}$. The parties then select a random committee of parties by calling $\mathcal{F}_{\text{COMMITTEE}}$ for evaluating the lth sub-circuit via the dishonest majority MPC protocol of [11]. We use a Boolean flag NewCom in the protocol to indicate if a new committee has to be decided, prior to the evaluation of lth sub-circuit or the committee used for the evaluation of the $(l-1)$th sub-circuit is to be continued. Specifically a successful evaluation of a sub-circuit is followed by setting NewCom equals to 0, implying that the current committee is to be continued for the evaluation of the subsequent sub-circuit. On the other hand, a failed evaluation of a sub-circuit is followed by setting NewCom equals to 1, implying that a fresh committee has to be decided for the re-evaluation of the same sub-circuit from the updated set of eligible parties $\mathcal{L}$, which is modified after the failed evaluation. After each successful sub-circuit evaluation, the corresponding $\langle\cdot\rangle$-shared outputs are converted into $[\cdot]$-shared outputs via protocol $\Pi_{\langle\cdot\rangle\rightarrow[\cdot]}$, while prior to each sub-circuit evaluation, the corresponding $[\cdot]$-shared inputs are converted to the required $\langle\cdot\rangle$-shared inputs via protocol $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$. The process is repeated till the function output is $[\cdot]$-shared, after which it is robustly reconstructed (as we have honest majority in $\mathcal{P}$).

[8] As we may not have honest majority in $\mathcal{X}$, we could not always ensure robust reconstruction during $\Pi_{\text{REC}\langle\cdot\rangle}$.

Without affecting the correctness of the above steps, but to ensure simulation security (in the UC model), we add an additional output re-randomization step before the output reconstruction: the parties call $\Pi_{\text{RANDZERO}[\cdot]}$ to generate a random $[0]$, which they add to the $[\cdot]$-shared output (thus keeping the same function output). Looking ahead, during the simulation in the security proof, this step allows the simulator to cheat and set the final output to be the one obtained from the functionality, even though it simulates the honest parties with 0 as the input (see the full version for the details).

Let E be the event that at least one party in each of the selected committees during sub-circuit evaluations is honest; the event E occurs except with probability at most $(t+1)\cdot\epsilon^{\mathsf{c}} \approx 2^{-\kappa}$. This is because at most $(t+1)$ (random) committees need to be selected (a new committee is selected after each of the t failed sub-circuit evaluation plus an initial selection is made). It is easy to see that conditioned on E, the protocol is private: the inputs of the honest parties remain private during the input stage (due to $\mathcal{F}_{\text{GEN}[\cdot]}$), while each of the involved sub-protocols for sub-circuit evaluations does not leak any information about honest party's inputs. It also follows that conditioned on E, the protocol is correct, thanks to the binding property of the commitment and the properties of the involved sub-protocols.

The properties of the protocol Π_f are stated in Theorem 1 and the security proof is available in the full version; we only provide the proof of communication complexity here. The (circuit-dependent) communication complexity in the theorem is derived after substituting the calls to the various ideal functionalities by the corresponding protocols implementing them. The broadcast complexity has two parts: the broadcasts among the parties in $\mathcal{P}$ and the broadcasts among small committees.

Theorem 1. *Let $f : \mathbb{F}_p^n \to \mathbb{F}_p$ be a publicly known n-input function with circuit representation ckt over $\mathbb{F}_p$, with average width w and depth d (thus $w = \frac{|\mathsf{ckt}|}{d}$). Moreover, let ckt be divided into sub-circuits $\mathsf{ckt}_1, \ldots, \mathsf{ckt}_L$, with $L = t$ and each sub-circuit ckt_l having fan-in in_l and fan-out out_l. Furthermore, let $\mathsf{in}_l = \mathsf{out}_l = \mathcal{O}(w)$. Then conditioned on the event E, protocol Π_f (κ, s)-securely realizes the functionality $\mathcal{F}_f$ against $\mathcal{A}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{COMMITTEE}}, \mathcal{F}_{\text{GEN}[\cdot]}, \mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}, \mathcal{F}_{\text{ZK.BC}})$-hybrid modelin the UC security framework. The circuit-dependent communication complexity of the protocol is $\mathcal{O}(|\mathsf{ckt}|\cdot(\frac{n\cdot t}{d}+\kappa)\cdot\kappa^2)$, $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\frac{n\cdot t\cdot\kappa^2}{d}, n\big)$ and $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\kappa^3, \kappa\big)$.*

PROOF (COMMUNICATION COMPLEXITY): We analyze each phase of the protocol:

1. **Input Commitment Stage:** Here each party broadcasts $\mathcal{O}(\kappa)$ bits to the parties in $\mathcal{P}$ and so the broadcast complexity of this step is $\mathcal{BC}\big(n\kappa, n\big)$.
2. $[\cdot]$**-sharing of Committed Inputs:** Here n calls to $\mathcal{F}_{\text{GEN}[\cdot]}$ are made. Realizing $\mathcal{F}_{\text{GEN}[\cdot]}$ with the protocol $\Pi_{[\cdot]}$, see the full version, this incurs a communication complexity of $\mathcal{O}(n^2\kappa)$ and $\mathcal{BC}\big(n^2\kappa, n\big)$.
3. **Sub-circuit Evaluations:** We first count the total communication cost of evaluating the sub-circuit ckt_l with in_l input gates and out_l output gates.
   - Converting the in_l $[\cdot]$-shared inputs to in_l $\langle\cdot\rangle$-shared inputs will require in_l invocations to the protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$. The communication complexity of this step is $\mathcal{O}(n\cdot\mathsf{c}\cdot\mathsf{in}_l\cdot\kappa)$ and $\mathcal{BC}\big(n\cdot\mathsf{c}\cdot\mathsf{in}_l\cdot\kappa, n\big)$; this follows from Lemma 3 by substituting $|\mathcal{X}| = \mathsf{c}$.

Protocol $\Pi_f(\mathcal{P}, \mathsf{ckt})$

For session ID sid, every party $P_i \in \mathcal{P}$ does the following:

Initialization. Set $\mathcal{L} = \mathcal{P}$, $\mathfrak{n} = |\mathcal{L}|$, $\mathfrak{t} = t$ and $\mathsf{NewCom} = 1$. Divide ckt into L sub-circuits $\mathsf{ckt}_1, \ldots, \mathsf{ckt}_L$, each of depth d/L.

CRS Generation. Invoke $\mathcal{F}_{\text{CRS}}$ with (sid, i) and get back $(\mathsf{sid}, i, \mathsf{CRS})$, where $\mathsf{CRS} = (\mathsf{pk}, \mathsf{ck})$.

Input Commitment. On input $x^{(i)}$, choose random polynomials $f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot)$ of degree $\leq t$, such that $f^{(i)}(0) = x^{(i)}$ and compute the commitment $\mathbf{C}_{x^{(i)}, g_0^{(i)}, h_0^{(i)}} = \mathsf{Comm}_{\mathsf{ck}}(x^{(i)}; g_0^{(i)}, h_0^{(i)})$ where $g_0^{(i)} = g^{(i)}(0),\ h_0^{(i)} = h^{(i)}(0)$.

- Call $\mathcal{F}_{\text{BC}}$ with message $(\mathsf{sid}, i, \mathbf{C}_{x^{(i)}, g_0^{(i)}, h_0^{(i)}}, \mathcal{P})$.
- Corresponding to each $P_j \in \mathcal{P}$, receive $(\mathsf{sid}, i, j, \mathbf{C}_{x^{(j)}, g_0^{(j)}, h_0^{(j)}})$ from $\mathcal{F}_{\text{BC}}$.

$[\cdot]$-sharing of Committed Inputs.

- Act as a dealer D and call $\mathcal{F}_{\text{GEN}[\cdot]}$ with $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$.
- For every $P_j \in \mathcal{P}$, call $\mathcal{F}_{\text{GEN}[\cdot]}$ with $(\mathsf{sid}, i, j, \mathbf{C}_{x^{(j)}, g_0^{(j)}, h_0^{(j)}})$.
- For every $P_j \in \mathcal{P}$, if $(\mathsf{sid}, i, j, \mathsf{Failure})$ is received from $\mathcal{F}_{\text{GEN}[\cdot]}$, substitute a default predefined public sharing $[0]$ of 0 as $[x^{(j)}]$, set $[x^{(j)}]_i = [0]_i$ and update $\mathcal{L} = \mathcal{L} \setminus \{P_j\}$, decrement $\mathfrak{t}$ and $\mathfrak{n}$ by one. Else receive $(\mathsf{sid}, i, j, [x^{(j)}]_i)$ from $\mathcal{F}_{\text{GEN}[\cdot]}$.

Start of While Loop Over the Sub-circuits. Initialize $l = 1$. While $l \leq L$ do:

- **Committee Selection.** If $\mathsf{NewCom} = 1$, then call $\mathcal{F}_{\text{COMMITTEE}}$ with $(\mathsf{sid}, i, \mathcal{L})$ and receive $(\mathsf{sid}, i, \mathcal{C})$ from $\mathcal{F}_{\text{COMMITTEE}}$.
- $[\cdot]$ **to** $\langle\cdot\rangle_{\mathcal{C}}$ **Conversion of Inputs of Sub-circuit** ckt_l**.** Let $[x_1], \ldots, [x_{\mathsf{in}_l}]$ denote the $[\cdot]$-sharing of the inputs to ckt_l:
  - For $k = 1, \ldots, \mathsf{in}_l$, participate in $\Pi_{[\cdot]\to\langle\cdot\rangle}$ with $(\mathsf{sid}, i, [x_k]_i, \mathcal{C})$. Output $(\mathsf{sid}, i, \langle x_k\rangle_i)$ in $\Pi_{[\cdot]\to\langle\cdot\rangle}$, if P_i belongs to $\mathcal{C}$. Else output (sid, i).
- **Evaluation of the Sub-circuit** ckt_l**.** If $P_i \in \mathcal{C}$ then participate in $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$ with $(\mathsf{sid}, i, \langle x_1\rangle_i, \ldots, \langle x_{\mathsf{in}_l}\rangle_i, \mathcal{C})$, else participate in $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$ with $(\mathsf{sid}, i, \mathcal{C})$.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ is the output during $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$, then set $\mathcal{L} = \mathcal{L} \setminus \{P_a, P_b\}$, $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n} = \mathfrak{n} - 2$, $\mathsf{NewCom} = 1$ and go to **Committee Selection** step.
- $\langle\cdot\rangle_{\mathcal{C}}$ **to** $[\cdot]$ **conversion of Outputs of** ckt_l**.** If $(\mathsf{sid}, i, \mathsf{Success}, \langle y_1\rangle_i, \ldots, \langle y_{\mathsf{out}_l}\rangle_i)$ or $(\mathsf{sid}, i, \mathsf{Success})$ is obtained during $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$, then participate in $\Pi_{\langle\cdot\rangle\to[\cdot]}$ with $(\mathsf{sid}, i, \langle y_k\rangle_i)$ or (sid, i) (respectively) for $k = 1, \ldots, \mathsf{out}_l$.
  - If $(\mathsf{sid}, i, \mathsf{Success}, [y_k]_i)$ is the output in $\Pi_{\langle\cdot\rangle\to[\cdot]}$ for every $k = 1, \ldots, \mathsf{out}_l$, then increment l and set $\mathsf{NewCom} = 0$.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ is the output in $\Pi_{\langle\cdot\rangle\to[\cdot]}$ for *some* $k \in \{1, \ldots, \mathsf{out}_l\}$, then set $\mathcal{L} = \mathcal{L} \setminus \{P_a, P_b\}$, $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n} = \mathfrak{n} - 2$, $\mathsf{NewCom} = 1$ and go to the **Committee Selection** step.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a)$ is the output in $\Pi_{\langle\cdot\rangle\to[\cdot]}$ for *some* $k \in \{1, \ldots, \mathsf{out}_l\}$, then set $\mathcal{L} = \mathcal{L} \setminus \{P_a\}$, $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n} = \mathfrak{n} - 1$, $\mathsf{NewCom} = 1$ and go to the **Committee Selection** step.

Output Rerandomization. Let $[y]$ denote the $[\cdot]$-sharing of the output of ckt. Participate in $\Pi_{\text{RANDZERO}[\cdot]}$ with (sid, i), obtain $(\mathsf{sid}, i, [0]_i)$ and locally compute $[z]_i = [y]_i + [0]_i$.

Output Reconstruction. Interpret $[z]_i$ as $(f_i, g_i, h_i, \{\mathbf{C}_{f_j, g_j, h_j}\}_{P_j \in \mathcal{P}})$. Initialize a set $\mathcal{T}_i$ to $\emptyset$.

- Send $(\mathsf{sid}, i, j, f_i, g_i, h_i)$ to every $P_j \in \mathcal{P}$. On receiving $(\mathsf{sid}, j, i, f_j, g_j, h_j)$ from every party P_j include party P_j in $\mathcal{T}_i$ if $\mathbf{C}_{f_j, g_j, h_j} \neq \mathsf{Comm}_{\mathsf{ck}}(f_j; (g_j, h_j))$.
- Interpolate $f(\cdot)$ such that $f(\alpha_j) = f_j$ holds for every $P_j \in \mathcal{P} \setminus \mathcal{T}_i$. If $f(\cdot)$ has degree at most t, output $(\mathsf{sid}, i, z = f(0))$ and halt; else output $(\mathsf{sid}, i, \mathsf{Failure})$ and halt.

Fig. 6. Protocol for UC-secure realizing $\mathcal{F}_f$

- Since the size of ckt_l is at most $\frac{|\mathsf{ckt}|}{L}$, evaluating the same via protocol $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$ will have communication complexity $\mathcal{O}(\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s)\kappa)$, $\mathcal{BC}\big(\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s)\kappa,\mathfrak{c}\big)$ and $\mathcal{BC}\big(\mathfrak{c}\cdot\kappa,n\big)$; this follows from Lemma 4 by substituting $|\mathcal{X}|=\mathfrak{c}$.
- Finally converting the out_l $\langle\cdot\rangle$-shared outputs to $[\cdot]$-shared outputs require out_l invocations to the protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$. This has communication complexity $\mathcal{O}(n\cdot\mathfrak{c}\cdot\mathsf{out}_l\cdot\kappa)$, $\mathcal{BC}\big(\mathsf{out}_l\cdot\mathfrak{c}^2\cdot\kappa,n\big)$ and $\mathcal{BC}\big(n\cdot\mathfrak{c}\cdot\kappa,n\big)$; this follows from Lemma 1 by substituting $|\mathcal{X}|=\mathfrak{c}$.

Thus evaluating ckt_l has communication complexity $\mathcal{O}((n^2+n\cdot\mathfrak{c}\cdot\mathsf{in}_l+n\cdot\mathfrak{c}\cdot\mathsf{out}_l+\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s))\kappa)$, $\mathcal{BC}\big((n^2+n\cdot\mathfrak{c}\cdot\mathsf{in}_l+\mathfrak{c}^2\cdot\mathsf{out}_l)\kappa,n\big)$ and $\mathcal{BC}\big(\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s)\kappa,\mathfrak{c}\big)$. Assuming $\mathsf{in}_l=\mathcal{O}(w)$ and $\mathsf{out}_l=\mathcal{O}(w)$, with $w=\frac{|\mathsf{ckt}|}{d}$, this results in $\mathcal{O}((n^2+n\cdot\mathfrak{c}\cdot\frac{|\mathsf{ckt}|}{d}+\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s))\kappa)$, $\mathcal{BC}\big((n^2+n\cdot\mathfrak{c}\cdot\frac{|\mathsf{ckt}|}{d})\kappa,n\big)$ and $\mathcal{BC}\big((\mathfrak{c}^2\cdot(\frac{|\mathsf{ckt}|}{L}+s))\kappa,\mathfrak{c}\big)$. The total number of sub-circuit evaluations is at most $L+t$, with L successful evaluations and at most t failed evaluations. Substituting $L=t$, we get the overall communication complexity $\mathcal{O}((|\mathsf{ckt}|\cdot(\frac{n\cdot t\cdot\mathfrak{c}}{d}+\mathfrak{c}^2)+n^2t+\mathfrak{c}^2s\cdot t)\kappa)$, $\mathcal{BC}\big((|\mathsf{ckt}|\cdot\frac{n\cdot t\cdot\mathfrak{c}}{d}+n^2t)\kappa,n\big)$ and $\mathcal{BC}\big((|\mathsf{ckt}|\cdot\mathfrak{c}^2+\mathfrak{c}^2\cdot s\cdot t)\kappa,\mathfrak{c}\big)$.

4. **Output Rerandomization and Reconstruction:** The costs $\mathcal{O}(\mathsf{Poly}(n,\kappa))$ bits.

The circuit-dependent complexity of the whole protocol comes out to be $\mathcal{O}(|\mathsf{ckt}|\cdot(\frac{nt\cdot\mathfrak{c}}{d}+\mathfrak{c}^2)\kappa)$ bits of communication over the point-to-point channels and broadcast-complexity of $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\frac{nt\cdot\mathfrak{c}}{d}\cdot\kappa,n\big)$ and $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\mathfrak{c}^2\cdot\kappa,\mathfrak{c}\big)$. Since $\mathfrak{c}$ has to be selected so that $\epsilon^{\mathfrak{c}}<2^{-\kappa}$ holds, asymptotically we can set $\mathfrak{c}$ to be $\mathcal{O}(\kappa)$. (For any practical purpose, $\kappa=80$ is good enough.) It implies that the (circuit-dependent) communication complexity is $\mathcal{O}(|\mathsf{ckt}|(\frac{nt}{d}+\kappa)\kappa^2)$, $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\frac{nt\kappa^2}{d},n\big)$ and $\mathcal{BC}\big(|\mathsf{ckt}|\kappa^3,\kappa\big)$. □

We propose two optimizations for our MPC protocol that improves its communication complexity.

$[\cdot]$-sharing among a smaller subset of $\mathcal{P}$. While for simplicity, we involve the entire set of parties in $\mathcal{P}$ to hold $[\cdot]$-shared values in the protocol, it is enough to fix and involve a set of just z parties that guarantees a honest majority with overwhelming probability. From our analysis in Section 1, we find that $z=\mathcal{O}(\kappa)$. Indeed it is easy to note that all we require from the set involved in holding a $[\cdot]$-sharing is honest majority that can be attained by any set containing $\mathcal{O}(\kappa)$ parties. This optimization replaces n by κ in the complexity expressions mentioned in Theorem 1. It implies that the (circuit-dependent) communication complexity is $\mathcal{O}(|\mathsf{ckt}|(\frac{\kappa t}{d}+\kappa)\kappa^2)$, $\mathcal{BC}\big(|\mathsf{ckt}|\cdot\frac{t\kappa^3}{d},\kappa\big)$ and $\mathcal{BC}\big(|\mathsf{ckt}|\kappa^3,\kappa\big)$. Now instantiating the broadcast functionality in the above modified protocol with the Dolev-Strong (DS) broadcast protocol (see the full version), we obtain the following:

Corollary 1. *If $d=\omega(t)$ and if the calls to $\mathcal{F}_{\mathrm{BC}}$ are realized via the DS broadcast protocol, then the circuit-dependent communication complexity of Π_f is $\mathcal{O}(|\mathsf{ckt}|\cdot\kappa^7)$.*

When we restrict to widths w of the form $w=\omega(\kappa^3)$, we can instantiate all the invocations to $\mathcal{F}_{\mathrm{BC}}$ in the protocols $\Pi_{\langle\cdot\rangle\to[\cdot]}$ and $\Pi_{[\cdot]\to\langle\cdot\rangle}$ (invoked before and after the sub-circuit evaluations) by the Fitzi-Hirt (FH) multi-valued broadcast protocol [15], see the full version. This is because, setting $w=\omega(\kappa^3)$ ensures that the combined message over all the instances of $\Pi_{\langle\cdot\rangle\to[\cdot]}$ (respectively $\Pi_{[\cdot]\to\langle\cdot\rangle}$) to be broadcast by any party

satisfies the bound on the message size of the FH protocol. Incorporating the above, we obtain the following corollary with better result.

Corollary 2. *If $d = \omega(t)$ and $w = \omega(\kappa^3)$ (i.e. $|\mathsf{ckt}| = \omega(\kappa^3 t)$), then the circuit-dependent communication complexity of Π_f is $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^4)$.*

Packed Secret-Sharing. We can employ packed secret-sharing technique of [16] to checkpoint multiple outputs of the sub-circuits together in a single $[\cdot]$-sharing. Specifically, if we involve all the parties in $\mathcal{P}$ to hold a $[\cdot]$-sharing, we can pack $n - 2t$ values together in a single $[\cdot]$-sharing by setting the degree of the underlying polynomials to $n - t - 1$. It is easy to note that robust reconstruction of such a $[\cdot]$-sharing is still possible, as there are $n - t$ honest parties in the set $\mathcal{P}$ and exactly $n - t$ shares are required to reconstruct an $(n - t - 1)$ degree polynomial. For every sub-circuit ckt_l, the w_{out_l} output values are grouped so that each group contains $n - 2t$ secrets and each group is then converted to a single $[\cdot]$-sharing.

If we restrict to circuits for which any circuit wire has length at most $d/L = d/t$ (i.e. reaches upto at most d/L levels), then we ensure that the outputs of circuit ckt_l can only be the input to circuit ckt_{l+1}. With this restriction, the use of packed secret-sharing becomes applicable at all stages, and the communication complexity becomes $\mathcal{O}(|\mathsf{ckt}| \cdot (\frac{t}{d} + \kappa) \cdot \kappa^2)$, $\mathcal{BC}\big(|\mathsf{ckt}| \cdot \frac{t \cdot \kappa^2}{d}, n\big)$ and $\mathcal{BC}\big(|\mathsf{ckt}| \cdot \kappa^3, \kappa\big)$; i.e. a factor of n less in the first two terms compared to what is stated in Theorem 1. Realizing the broadcasts using DS and FH protocol respectively, we obtain the following corollaries:

Corollary 3. *If $d = \omega(\frac{n^3 \cdot t}{\kappa^4})$ and if the calls to $\mathcal{F}_{\mathrm{BC}}$ are realized via the DS broadcast protocol, then the circuit-dependent communication complexity of Π_f is $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$.*

Corollary 4. *If $d = \omega(\frac{n \cdot t}{\kappa^5})$ and $w = \omega(n^2 \cdot (n + \kappa))$ (i.e. $|\mathsf{ckt}| = \omega(\frac{n^3 \cdot t}{\kappa^5}(n + \kappa))$), then the circuit-dependent communication complexity of the protocol Π_f is $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$.*

Acknowledgements

This work has been supported in part by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO, by EPSRC via grant EP/I03126X, and by Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number FA8750-11-2-0079[9] and the third author was supported in part by a Royal Society Wolfson Merit Award.

References

1. M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In *Advances in Cryptology - CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334, 2004.

[9] The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Defense Advanced Research Projects Agency (DARPA) or the U.S. Government.

2. G. Asharov and Y. Lindell. A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation. *IACR Cryptology ePrint Archive*, 2011:136, 2011.
3. D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91*, volume 576 of *LNCS*, pages 420–432, 1991.
4. Z. BeerliováTrubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In *Theory of Cryptography – TCC 2008*, volume 4948 of *LNCS*, pages 213–230, 2008.
5. E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680, 2012.
6. E. Boyle, S. Goldwasser, and S. Tessaro. Communication locality in secure multi-party computation how to run sublinear algorithms in a distributed setting. In *Theory of Cryptography – TCC 2014*, volume 8349 of *LNCS*, pages 356–376, 2014.
7. R. Canetti and M. Fischlin. Universally Composable Commitments. In *Advances in Cryptology – CRYPTO 2001*, pages 19–40, 2001.
8. J. F. Canny and S. Sorkin. Practical large-scale distributed key generation. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 138–152, 2004.
9. A. Choudhury. Breaking the $\mathcal{O}(n|c|)$ barrier for unconditionally secure asynchronous multiparty computation - (extended abstract). In *Progress in Cryptology - INDOCRYPT 2013*, volume 8250 of *LNCS*, pages 19–37, 2013.
10. I. Damgård, Y. Ishai, M. Krøigaard, J.B. Nielsen, and A. Smith. Scalable Multiparty Computation with Nearly Optimal Work and Resilience. In *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *LNCS*, pages 241–261, 2008.
11. I. Damgård and C. Orlandi. Multiparty Computation for Dishonest Majority: From Passive to Active Security at Low Cost. In *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *LNCS*, pages 558–576, 2010.
12. V. Dani, V. King, M. Movahedi, and J. Saia. Brief Announcement: Breaking the $O(nm)$ Bit Barrier, Secure Multiparty Computation with a Static Adversary. In *Principles of Distributed Computing – PODC 2012*, pages 227–228, 2012.
13. V. Dani, V. King, M. Movahedi, and J. Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *ICDN 2014*, volume 8314 of *LNCS*, pages 242–256, 2014.
14. Uriel Feige. Noncryptographic selection protocols. In *FOCS*, pages 142–153, 1999.
15. Matthias Fitzi and Martin Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In *Principles of Distributed Computing – PODC 2006*, pages 163–168. ACM, 2006.
16. M. K. Franklin and M. Yung. Communication Complexity of Secure Computation (Extended Abstract). In *Symposium on Theory of Computing – STOC 1992*, pages 699–710. ACM, 1992.
17. B. M. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast Asynchronous Byzantine Agreement and Leader Election with Full Information. *ACM Transactions on Algorithms*, 6(4), 2010.
18. V. King, S. Lonargan, J. Saia, and A. Trehan. Load Balanced Scalable Byzantine Agreement through Quorum Building, with Information. In *ICDCN*, volume 6522 of *LNCS*, pages 203–214, 2011.
19. V. King and J. Saia. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary. *J. ACM*, 58(4):18, 2011.
20. V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable Leader Election. In *SODA*, pages 990–999, 2006.
21. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91*, volume 576 of *LNCS*, pages 129–140, 1992.
22. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.