Reducing the Overhead of Cloud MPC

A. Choudhury*, A. Patra and N. P. Smart*****

Abstract. We present a secure multi-party computation (MPC) protocol in the honest-majority setting, which aims to reduce the communication costs in the situation where there are a large number of parties (as in a cloud scenario). Our goal is to reduce the usage of point-to-point channels, so as to enable the cloud to be used for multiple different protocol executions. We assume that the number of adversarially controlled parties is relatively small, and that an adversary is unable to target the proactive corruption of a subset of the parties (technically we assume a static corruption model for simplicity). As well as enabling a cloud provider to run multiple MPC protocols, our protocol also has highly efficient theoretical communication costs as a general MPC protocol when compared with other protocols in the literature; in particular the communication cost, for circuits of a suitably large depth, is $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$, for security parameter κ and circuit size $|\mathsf{ckt}|$.

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 and the third author was supported in part by a Royal Society Wolfson Merit Award.

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.

* Department of Computer Science and Engineering, Jadavpur University, India. Email: achoudhury@cse.jdvu.ac.in, partho31@gmail.com.

** Applied Statistics Unit, Indian Statistical Institute Kolkata. Email: arpitapatra10@gmail.com.

*** Department of Computer Science, University of Bristol, United Kingdom. Email: nigel@cs.bris.ac.uk.

1 Introduction

Threshold secure multi-party computation (MPC) [29, 21] is a fundamental problem in secure distributed computing. On a very high level 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 the situation of an organization performing a multi-party computation on a cloud infrastructure. The cloud infrastructure is likely to be very large; if the number of machines is n then one can expect $n \geq 1000$. However, the proportion of compromised machines (namely the ration t/n) is likely to be rather small in comparison, say 5 percent. Using the entire cloud infrastructure to perform an MPC calculation is wasteful in this situation, as typical (secret-sharing based) MPC protocols (such as [3, 5, 6, 9, 13]) 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 adversary. In particular we present a protocol in which the main computation is performed by a "smallish" subset of the cloud infrastructure, with the whole cloud only being used occasionally so as to "checkpoint" the computation. Note that by not utilizing the entire cloud infrastructure all the time enables the cloud infrastructure to be utilized for many MPC calculations at once.

Asymptotically MPC protocols have been obtained which provide poly-log communication complexity, in both the computational and the information-theoretic setting[1] [10, 11]. These protocols become applicable (and efficient) when n is very large, and the ratio of corrupted parties $\epsilon = \frac{t}{n}$ is relatively small, which is exactly the situation described above. But all these prior works require communications between all parties at all gate evaluations[2]. Our protocol not only reduces the need for the parties to be communicating with all others at all stages in the protocol, but also asymptotically provides a better communication complexity than these prior works. Specifically, for evaluating circuits of suitably large depth, we require a communication complexity of $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$, where κ is the security parameter and $|\mathsf{ckt}|$ is the circuit size. Note there is no dependence here on the number of parties n involved or the corruption threshold t; this is unlike the previous best protocol of [11], which also considers a computationally bounded adversary and has the same resilience as our protocol, but requires a communication complexity of $\mathcal{O}(|\mathsf{ckt}| \cdot \mathsf{poly}(\kappa, \log n, \log |\mathsf{ckt}|))$.

We remark that the efficiency improvement of our protocol holds for circuits of sufficiently "large" depth and for values of ϵ in $[1/100, 1/2]$. For smaller values of ϵ simpler protocols may be possible (more on this later). However, we do not consider it a limitation because we view our result as asymptotic improvement.

Protocol Overview: Let $0 \leq \epsilon < \frac{1}{2}$ and κ be the security parameter (for example $\kappa = 80$ or 128). 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 [27] 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 [28], amongst a well-defined subset $\mathcal{C}$ of the parties.

The inputs to the protocol are assumed to be $[\cdot]$ shared amongst the parties at the start of the protocol. We first divide ckt to be evaluated 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 $\mathfrak{c}$. If $\mathfrak{c}$ is such that $\epsilon^{\mathfrak{c}} < 2^{-\kappa}$

[1] The computational setting considers a computationally bounded adversary, while the information-theoretic setting considers a computationally unbounded adversary.

[2] Informally, in any generic MPC protocol, the function to be computed securely is expressed as a publicly known arithmetic circuit and the goal of the protocol is to "securely" evaluate each gate of the circuit.

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[3] 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 checkpointing 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 checkpointing prevents from re-evaluating the entire circuit from scratch after every abortion 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, which also maps into our cloud scenario above.

The main technical difficulty is in providing actively-secure translations 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. The requirement of a dishonest-majority MPC protocol which enables identification of cheaters, without sacrificing privacy, leads us to the utilization of the protocol in [12]. This results in us needing to use double-trapdoor homomorphic commitments as a basic building block. To ensure greater asymptotic efficiency we also apply the packing technique from [10] 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.

Our protocol is not adaptively secure [20]. Whilst we only prove security in a static setting, we feel that adaptive[4] 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 more complex situation to a future work.

We note that the idea of using small committees instead of involving the whole set of parties in the protocol has been used earlier in the literature for the leader election [22, 25] and Byzantine agreement problem [24, 23]. The above approach was also used for MPC in [14]; however the committee size there

[3] 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.

[4] A static adversary decides the set of t parties to corrupt at the beginning of the execution of a protocol. An adaptive adversary can dynamically decide the set of parties to corrupt during the execution of a protocol based on the information obtained so far during the execution, subject to the condition that at any point no more than t parties are under the control of the adversary.

is polylog(n) (which is dependent on n) and the communication complexity of their protocol is $\mathcal{O}(|\mathsf{ckt}| \cdot \text{polylog}(n))$. So asymptotically our protocol is better as there is no dependence on n.

Notice that in our protocol we select committees of size $\mathfrak{c}$ satisfying $\epsilon^{\mathfrak{c}} < 2^{-\kappa}$, which ensures that statistically the selected committee has dishonest majority with overwhelming probability. One may think that instead we can randomly select committees of larger size and with overwhelming probability the selected committee will have honest majority, and this is indeed true for very small values of ϵ. Thus for small ϵ our protocol is sub-optimal. For example, at the $\kappa = 80$ security level and large n, with $\epsilon = 1/3$ we need a committee of size 51 to obtain committee with at least one honest member, whilst we need one of size 1388 to obtain a committee with honest majority (both with overwhelming probability). These figures reduce to 24 and 100 for $\epsilon = 1/10$ and 12 and 32 for $\epsilon = 1/100$. That is, our protocol is interesting when the fraction of corrupted parties is not too small, in which case a simpler protocol working with a committee with honest majority may be possible (where we need not have to worry about aborts).

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 [7]. 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. This is exactly what is required in cloud scenario where multiple protocols may run in parallel. In the UC 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 participants 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 ideal functionality. For more details, see Appendix A.

We do not assume a physical broadcast channel among the parties. Instead, we formalize broadcast using an ideal broadcast functionality $\mathcal{F}_{\text{BC}}$ (see Fig. 4; Appendix B) 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 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 $\mathcal{BC}\big(\ell, |\mathcal{X}|\big)$ to denote that in total, $\mathcal{O}(\ell)$ bits is broadcasted in the associated protocol to a set of parties of size $|\mathcal{X}|$. For details about the instantiation of $\mathcal{F}_{\text{BC}}$, see Appendix C.

As mentioned in the introduction, 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 [26]. 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 as "input independence". It has been shown in [8, 7] that a UC secure commitment scheme is impossible to achieve without setup assumptions. The standard methods 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 [12], 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 $\mathbb{F}_p$ [12]). *It is a collection of the following five PPT algorithms* $(\mathsf{Gen}, \mathsf{Comm}, \mathsf{Open}, \mathsf{Equivocate}, \mathsf{TDExtract}, \odot)$*:*
(1): $\mathsf{Gen}(1^\kappa) \to (\mathsf{ck}, \tau_0, \tau_1)$*: the generation algorithm outputs a commitment key* ck*, along with trapdoors* τ_0 *and* τ_1. **(2):** $\mathsf{Comm}_{\mathsf{ck}}(x; r_0, r_1) \to \mathbf{C}_{x,r_0,r_1}$*: the commitment algorithm takes a message* $x \in \mathbb{F}_p$ *and randomness* r_0, r_1 *from the commitment randomness space* $\mathcal{R}$ [5] *and outputs a commitment* $\mathbf{C}_{x;r_0,r_1}$ *of* x *under the randomness* r_0, r_1. **(3):** $\mathsf{Open}_{\mathsf{ck}}(\mathbf{C}, (x; r_0, r_1)) \to \{0, 1\}$*: 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. **(4):** $\mathsf{Equivocate}(\mathbf{C}_{x,r_0,r_1}, x, r_0, r_1, \overline{x}, \tau_i) \to (\overline{r}_0, \overline{r}_1) \in \mathcal{R}$*: 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* $r_{1-i} = \overline{r}_{1-i}$.**(5):** $\mathsf{TDExtract}(\mathbf{C}, x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1, \tau_i) \to \tau_{1-i}$*: 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* $\mathsf{Comm}(x; r_0, r_1)^c = \mathsf{Comm}(cx; cr_0, cr_1)$ *for any publicly known constant* c*. We require the following properties to be satisfied:*

- **Trapdoor Security***: There exists no PPT algorithm* A *such that* $A(1^\kappa, \mathsf{ck}, \tau_i) \to \tau_{1-i}$*, for* $i \in \{0, 1\}$.
- **Computational Binding***: There exists no PPT algorithm* A *such that* $A(1^\kappa, \mathsf{ck}) \to (x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1)$*, with* $(x, r_0, r_1) \neq (\overline{x}, \overline{r}_0, \overline{r}_1)$*, 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 randomness* $r_0, r_1 \in \mathcal{R}$*, 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 [12]. 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 $\overline{r}_i = \tau_i^{-1}(x - \overline{x}) + r_i$.
- $\mathsf{TDExtract}(\mathbf{C}, x, r_0, r_1, \overline{x}, \overline{r}_0, \overline{r}_1, \tau_i) \to \tau_{1-i}$, 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. $\mathsf{Comm}(x; r_0, r_1) \odot \mathsf{Comm}(\overline{x}; \overline{r}_0, \overline{r}_1) = g^x h_0^{r_0} h_1^{r_1} \cdot g^{\overline{x}} h_0^{\overline{r}_0} h_1^{\overline{r}_1} = g^{x+\overline{x}} \cdot h_0^{r_0+\overline{r}_0} \cdot h_1^{r_1+\overline{r}_1} = \mathsf{Comm}(x + \overline{x}; r_0 + \overline{r}_0, r_1 + \overline{r}_1)$.

We can now define the various types of secret-shared data used in our protocols. We assume that there exist n publicly known non-zero, distinct values $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_p$, where α_i is associated with P_i as the *evaluation point*. The $[\cdot]$ sharing scheme is the standard Shamir-sharing [27], 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.

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

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 the commitment key* ck. *Moreover, the collection of $[s]_i$'s, corresponding to $P_i \in \mathcal{P}$ is denoted by $[s]$.*

Our second type of secret-sharing scheme, namely $\langle\cdot\rangle$-sharing (which is a variation of additive sharing), will be 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 the commitment key* 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

In this section, we 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 the party P_i for every $i \in \{1, \ldots, n\}$ 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[a] $\mathcal{S}$ and halts.

[a] The symbol $\mathcal{S}$ is used to denote the adversary in the ideal-world functionality in the UC model; see Appendix A.

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

The underlying idea of our MPC protocol is discussed below. 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 we have the following two cases:

1. Single Elimination: Here $\epsilon_{\mathsf{new}} = \frac{\mathfrak{t}-1}{\mathfrak{n}-1}$ and so $\epsilon_{\mathsf{new}} \leq \epsilon_{\mathsf{old}}$ if and only if $\mathfrak{n} \geq \mathfrak{t}$, which will always hold.
2. Double Elimination: Here $\epsilon_{\mathsf{new}} = \frac{\mathfrak{t}-1}{\mathfrak{n}-2}$ 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 [12], 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 [12] 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

Our protocol makes use of ideal functionalities for **(a):** generating the CRS[6] ($\mathcal{F}_{\text{CRS}}$), **(b):** for broadcast ($\mathcal{F}_{\text{BC}}$), **(c):** for committee selection ($\mathcal{F}_{\text{COMMITTEE}}$) and **(d):** for generating $[\cdot]$-sharing of values ($\mathcal{F}_{\text{GEN}[\cdot]}$). In Appendix B we present these functionalities and then in Appendix C, we describe the protocols implementing them and give proofs of their security in the UC framework.

[6] The CRS is used to prove security in the UC model.

3.2 Supporting Sub-protocols

Our MPC protocol also makes use of the following four sub-protocols. In Appendix D we present these sub-protocols and estimate their communication complexity. We do not prove the security of these sub-protocols. Since we 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 $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ and $\mathcal{F}_{\mathrm{BC}}$. **(B) Protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$:** 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. This protocol further uses the following sub-protocol $\Pi_{\langle\cdot\rangle}$. **(C) Protocol $\Pi_{\langle\cdot\rangle}$:** the protocol enables a designated party 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. **(D) Protocol $\Pi_{\mathbf{RANDZERO}[\cdot]}$:** 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 $\mathcal{F}_{\mathrm{ZK.BC}}$ (given in Appendix D) to publicly prove a commitment to zero.

Apart from the above sub-protocols, we use a non-robust, dishonest-majority MPC protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ with the capability of fault-detection. The protocol presented in Figure 18 of Appendix E 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 $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ is the dishonest majority MPC protocol of [12]. The MPC protocol of [12] takes $\langle\cdot\rangle$-shared inputs of a given circuit, from a set of parties, say $\mathcal{X}$, having 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 misbehavior by the corrupted parties and abort the protocol.

We observe that for an aborted execution of the protocol of [12], 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 $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ to enable the parties in $\mathcal{P}$ identify a pair of parties from $\mathcal{X}$ with at least one of them being corrupted.

Protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ proceeds in two stages, the *preparation stage* and the *evaluation stage*, each involving various other sub-protocols (details available in the sequel). 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 is easy and does not require any interaction among the parties in $\mathcal{X}$, 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] (see protocol Π_{BEA} in Appendix E). While applying the Beaver's trick, the parties in $\mathcal{X}$ need to publicly open two $\langle\cdot\rangle$-shared values using a reconstruction protocol $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ (presented in Appendix E). It may be possible that the opening is non-robust[7], in which case the

[7] As we may not have honest majority in $\mathcal{X}$, we could not always ensure robust reconstruction; see protocol $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ for the details.

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 the protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ is provided in Appendix E. The protocol invokes two ideal functionalities $\mathcal{F}_{\mathrm{GENRAND}\langle\cdot\rangle}$ and $\mathcal{F}_{\mathrm{BC}}$ where the functionality $\mathcal{F}_{\mathrm{GENRAND}\langle\cdot\rangle}$ is used to generate $\langle\cdot\rangle$-sharing of random values (see Figure 16 in Appendix E). For our purpose we note that the protocol provides a statistical security of 2^{-s} and has communication complexity as stated in Lemma 1 and proved in Appendix E. Note that there are two types of broadcast involved: among the parties in $\mathcal{X}$ and among the parties in $\mathcal{P}$.

Lemma 1. *For statistical security parameter s, protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ 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 $\mathcal{F}_{\mathrm{GEN}[\cdot]}$. The parties then select a random committee of parties by calling $\mathcal{F}_{\mathrm{COMMITTEE}}$ for evaluating the lth sub-circuit via the dishonest majority MPC protocol of [12]. 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 $\mathcal{P}$).

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 $\Pi_{\mathrm{RANDZERO}[\cdot]}$ 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 Appendix F 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 $\mathcal{F}_{\mathrm{GEN}[\cdot]}$), 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 proof is available in Appendix F. The (circuit-dependent) communication complexity in the theorem is derived after substituting the calls to the

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 input (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 $f^{(i)}(0) = x^{(i)}$.

- Compute the commitment to the input as $\mathbf{C}_{x^{(i)}, g_0^{(i)}, h_0^{(i)}} = \mathsf{Comm}_{\mathsf{ck}}(x^{(i)}; g_0^{(i)}, h_0^{(i)})$ where $g_0^{(i)} = g^{(i)}(0)$, $h_0^{(i)} = h^{(i)}(0)$, and 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 $(\mathsf{sid}, i, [x_k]_i, \mathcal{C})$.
  - For $k = 1, \ldots, \mathsf{in}_l$, 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 obtained as the output during $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$, then update $\mathcal{L}$ as $\mathcal{L} = \mathcal{L} \setminus \{P_a, P_b\}$, $\mathfrak{t}$ as $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n}$ as $\mathfrak{n} = \mathfrak{n} - 2$, set $\mathsf{NewCom} = 1$ 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 from $\Pi_{\langle\cdot\rangle\to[\cdot]}$ for *some* $k \in \{1, \ldots, \mathsf{out}_l\}$, then update $\mathcal{L}$ as $\mathcal{L} = \mathcal{L}\setminus\{P_a, P_b\}$, $\mathfrak{t}$ as $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n}$ as $\mathfrak{n} = \mathfrak{n} - 2$, set $\mathsf{NewCom} = 1$ and go to the **Committee Selection** step.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a)$ is the output from $\Pi_{\langle\cdot\rangle\to[\cdot]}$ for *some* $k \in \{1, \ldots, \mathsf{out}_l\}$, then update $\mathcal{L}$ as $\mathcal{L} = \mathcal{L} \setminus \{P_a\}$, $\mathfrak{t}$ as $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n}$ as $\mathfrak{n} = \mathfrak{n} - 1$, set $\mathsf{NewCom} = 1$ 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 $(f_i, g_i, h_i, \{\mathbf{C}_{f_j, g_j, h_j}\}_{P_j\in\mathcal{P}})$.

- Send $(\mathsf{sid}, i, j, f_i, g_i, h_i)$ to every $P_j \in \mathcal{P}$.
- Initialize a set $\mathcal{T}_i$ to $\emptyset$. 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 the polynomial $f(\cdot)$ 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, then output (sid, i, z) and halt, where $z = f(0)$; else output $(\mathsf{sid}, i, \mathsf{Failure})$ and halt.

Fig. 2. Protocol for UC-secure realizing $\mathcal{F}_f$ 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 Model

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 model[8] in the UC security framework. The circuit-dependent communication complexity of the protocol is $\mathcal{O}(|\mathsf{ckt}| \cdot (\frac{n\cdot t}{d} + \kappa) \cdot \kappa^2)$, $\mathcal{BC}\left(|\mathsf{ckt}| \cdot \frac{n\cdot t\cdot\kappa^2}{d}, n\right)$ and $\mathcal{BC}\left(|\mathsf{ckt}| \cdot \kappa^3, \kappa\right)$.*

[8] See Appendix A for the meaning of g-hybrid model in the UC framework.

The following corollary to Theorem 1 follows easily when we instantiate the broadcast primitive with the Dolev-Strong (DS) broadcast protocol (see Appendix C) over the point-to-point channels.

Corollary 1. *If $d = \omega(\frac{n^4 \cdot t}{\kappa^4})$ and if the calls to $\mathcal{F}_{\mathrm{BC}}$ are realized via the DS broadcast protocol, then the circuit-dependent communication complexity of the protocol Π_f is $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$.*

When we restrict to widths w of the form $w = \omega(n^2 \cdot (n + \kappa))$, 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 [17] (see Appendix C). This is because, setting $w = \omega(n^2 \cdot (n+\kappa))$ 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, which allows us to obtain the same result but with a milder restriction on d.

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

We propose two optimizations for our MPC protocol that improves its communication complexity and enables us to reduce the dependence on d in the previous corollaries.

$[\cdot]$-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 $2t + 1$ parties among $\mathcal{P}$ for the same. 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 $2t + 1$ parties. This optimization replaces n by t in the complexity expressions mentioned in Theorem 1, Corollaries 1 and 2. By noting that $n = t/\epsilon$ with $0 \leq \epsilon < \frac{1}{2}$, we remark that replacing n by t can give a big saving when ϵ becomes closer to zero.

Packed Secret-Sharing. We can employ packed secret-sharing technique of [18] 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. The corollaries are then translated as follows:

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 the protocol 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 $\mathcal{O}(|\mathsf{ckt}| \cdot \kappa^7)$.*

References

1. M. Abe and S. Fehr. Adaptively Secure Feldman VSS and Applications to Universally-Composable Threshold Cryptography. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 317–334. Springer Verlag, 2004.

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 J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.
4. Z. BeerliováTrubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.
5. R. Bendlin, I. Damgård, C. Orlandi, and S. Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188, 2011.
6. Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.
8. Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.
9. R. Cramer, I. Damgård, and U. M. Maurer. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer Verlag, 2000.
10. I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.
11. I. Damgård, Y. Ishai, M. Krøigaard, J.B. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 241–261, 2008.
12. I. Damgård and C. Orlandi. Multiparty Computation for Dishonest Majority: From Passive to Active Security at Low Cost. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 558–576. Springer Verlag, 2010.
13. I. Damgård, V. Pastro, N.P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
14. 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 *PODC*, pages 227–228, 2012.
15. D. Dolev and H. R. Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
16. M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2002. `ftp://ftp.inf.ethz.ch/pub/crypto/publications/Fitzi03.pdf`.
17. Matthias Fitzi and Martin Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 163–168. ACM, 2006.
18. M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.
19. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In B. A. Coan and Y. Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.
20. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
21. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.
22. 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.
23. V. King, S. Lonargan, J. Saia, and A. Trehan. Load balanced scalable Byzantine agreement through quorum building, with information. In *ICDCN*, volume 6522 of *Lecture Notes in Computer Science*, pages 203–214. Springer Verlag, 2011.
24. 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.
25. V. King, J. Saia, V. Sanwalani, and E. Vee. Scalable leader election. In *SODA*, pages 990–999, 2006.
26. T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 1992.
27. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.
28. D. R. Stinson. *Cryptography - Theory and Practice*. Discrete mathematics and its applications series. CRC Press, 2005.

29. A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164, 1982.

Appendices

A The UC Security Model

We work in the standard Universal Composability (UC) framework of Canetti [7], with static corruption. The UC framework introduces a PPT environment $\mathcal{Z}$ that is invoked on the computational security parameter κ, the statistical security parameter s and an auxiliary input z and oversees the execution of a protocol in one of the two worlds. The "ideal" world execution involves dummy parties $P_1, \ldots, P_n$, an ideal adversary $\mathcal{S}$ who may corrupt some of the dummy parties, and a functionality $\mathcal{F}$. The "real" world execution involves the PPT parties $P_1, \ldots, P_n$ and a real world adversary $\mathcal{A}$ who may corrupt some of the parties. In either of these two worlds, a PPT adversary can corrupt t parties out of the n parties. The environment $\mathcal{Z}$ chooses the input of the parties and may interact with the ideal world/real world adversary during the execution. At the end of the execution, it has to decide upon and output whether a real or an ideal world execution has taken place.

We let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, s, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting with the ideal execution with adversary $\mathcal{S}$, the functionality $\mathcal{F}$, on the computational security parameter κ, the statistical security parameter s and z. Let $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, s, z)\}_{\kappa,s\in\mathbb{N},z\in\{0,1\}^*}$. Similarly let $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, s, z)$ denote the random variable describing the output of the environment $\mathcal{Z}$ after interacting in a real execution of a protocol Π with adversary $\mathcal{A}$, the parties $\mathcal{P}$, on the computational security parameter κ, the statistical security parameter s and z. Let $\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa, s, z)\}_{\kappa,s\in\mathbb{N},z\in\{0,1\}^*}$.

Definition 4. *For $n \in \mathbb{N}$, let $\mathcal{F}$ be an n-ary functionality and let Π be an n-party protocol. We say that Π (κ, s)-securely realizes $\mathcal{F}$ in the UC security framework, if for every PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, corrupting the same parties, such that the following two distributions are computationally indistinguishable in κ, with all but 2^{-s} probability:*

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \stackrel{c}{\approx} \mathrm{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}.$$

We consider the above definition where it quantifies over different adversaries: passive or active, that corrupts only certain number of parties. Note that the security offered by the statistical security parameter s does not depend upon the computational power of the adversary.

Modular Composition: A great advantage of the UC model is that it allows to prove the security of the protocols in a modular fashion. Specifically, the sequential modular composition theorem [7] states that in order to analyze the security of a protocol π_f for computing a function f that uses a subprotocol π_g for computing another function g, it suffices to consider the execution of π_f in a model where a trusted third party is used to ideally compute g (instead of the parties running the real subprotocol π_g). This facilitates a modular analysis of security: we first prove the security of π_g (as per the UC definition) and then prove the security of π_f assuming an ideal party (functionality) for g. This model in which π_f is analyzed using ideal calls to g, instead of executing π_g, is called the *g-hybrid model* because it involves both a real protocol execution (for computing f) and an ideal trusted third party computing g.

B Supporting Functionalities

Below, we present a number of ideal functionalities defining sub-components of our main protocol. We start with three basic functionalities for generating the CRS, for broadcast and for committee selection. We then present the functionality to enable the sharing of values via $[\cdot]$-sharings. In Appendix C, we describe the protocols realizing these functionalities and give proofs of their security in the UC framework.

Basic Functionalities: The functionality $\mathcal{F}_{\text{CRS}}$ for generating the common reference string (CRS) for our main MPC protocol is given in Figure 3. 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 [12]; see Appendix E).

The functionality $\mathcal{F}_{\text{BC}}$ for group broadcast is given in Figure 4. 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. In the rest of the paper, we will denote by $\mathcal{BC}(\ell, |\mathcal{X}|)$ the complexity of broadcasting a message of size ℓ to a set of parties $\mathcal{X}$.

The functionality $\mathcal{F}_{\text{COMMITTEE}}$ for a random committee selection is given in Figure 5. 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 [12]

Fig. 3. 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. 4. The Ideal Functionality for Broadcast

Functionality Related to $[\cdot]$-sharings: In Figure 6 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

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

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

- Upon receiving $(\mathsf{sid}, i, \mathcal{Y})$ from every party $P_i \in \mathcal{P}$, 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 then halts.

Fig. 5. The Ideal Functionality for Selecting a Random Committee of Given Size $\mathfrak{c}$

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 $\mathcal{P}$ containing at most t corrupted parties, 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.

- 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 the following:
  - $f(\cdot), g(\cdot)$ and $h(\cdot)$ are of degree at most t;
  - $\mathbf{C} \stackrel{?}{=} \mathsf{Comm}_{\mathsf{ck}}(f(0); g(0), h(0))$ holds; where $\mathbf{C}$ is received from the majority of the parties in $\mathcal{P}$.
- 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 then sets $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ and sends $(\mathsf{sid}, i, \mathsf{D}, [s]_i)$ to every party $P_i \in \mathcal{P}$ and halts.

Fig. 6. The Ideal Functionality for Verifiably Generating $[\cdot]$-sharing

We remark that the functionality $\mathcal{F}_{\text{GEN}[\cdot]}$ 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 [26], a public commitment of all the shares and the secret are available to the parties without violating any privacy. In order to make the commitments of the shares 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.

C UC-secure Instantiation of Various Functionalities

C.1 Protocol for Realizing $\mathcal{F}_{\text{GEN}[\cdot]}$

We design a protocol $\Pi_{[\cdot]}$, presented in Figure 7, for realizing the functionality $\mathcal{F}_{\text{GEN}[\cdot]}$ in the UC framework. We closely follow the standard Pedersen VSS scheme [26] against a threshold static adversary. Specifically, let $\mathbf{C}$ be the existing commitment available to the parties in $\mathcal{P}$ such that $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(s; g, h)$ and let (s, g, h) be available to D. To $[\cdot]$-share s, the dealer D selects three random polynomials $f(\cdot), g(\cdot)$ and $h(\cdot)$ each of degree at most t such that $f(0) = s, g(0) = g$ and $h(0) = h$. To every party P_i in $\mathcal{P}$, D distributes the share $f_i = f(\alpha_i)$ and opening information $g_i = g(\alpha_i)$ and $h_i = h(\alpha_i)$. Additionally, D publicly commits to the shares of all the share-holders, with the corresponding opening information acting as the randomness for the commitments. Namely D broadcasts $\{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}}$ via $\mathcal{F}_{\mathrm{BC}}$.

Every honest party P_i then verifies three conditions: **(1).** if the commitments correspond to polynomials of degree at most t **(2).** if the commitments are consistent with $\mathbf{C}$ in the sense that the constant terms of the polynomials committed via the commitments $\{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}}$ are indeed embedded in $\mathbf{C}$ **(3).** if f_i, g_i, h_i received over the point-to-point channel is consistent with $\mathbf{C}_{f_i,g_i,h_i}$ received via $\mathcal{F}_{\mathrm{BC}}$. The first two tests can be done appealing to the homomorphic property of the commitment scheme. If any of the first two tests fails, then P_i concludes that D is corrupted and outputs $\mathsf{Failure}$. If the last test fails (but first two tests succeed), then P_i complains D (publicly) who resolves the complain by revealing f_i, g_i, h_i via $\mathcal{F}_{\mathrm{BC}}$. Subsequently, the third test is checked with f_i, g_i, h_i received from D publicly. If the test is successful, P_i accepts the new f_i, g_i, h_i and outputs $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$. Else P_i outputs $\mathsf{Failure}$.

Intuitively the privacy of the shared secret s for an honest D follows from the fact that $\mathcal{A}$ may learn at most t shares, which constitute t distinct points on $f(\cdot)$ having degree t; so from adversary's point of view, we have one "degree of freedom"; i.e. for every possible choice of s, there exists a unique $f(\cdot)$ polynomial of degree t, which is consistent with the shares received by $\mathcal{A}$. Note that the publicly known commitment of the shares do not provide any additional information about the unknown shares to $\mathcal{A}$, thanks to the (statistical) hiding property of the commitment scheme and the fact that the corresponding randomness lie on polynomials of degree at most t and $\mathcal{A}$ will be provided with at most t points on them, again implying one degree of freedom.

The properties of the protocol $\Pi_{[\cdot]}$ are formally stated in Lemma 2.

Lemma 2. *Let $\mathsf{D} \in \mathcal{P}$ be a dealer with secret s and randomness pair (g, h) and let $\mathbf{C}$ be a publicly known commitment available to the parties in $\mathcal{P}$. Then the protocol $\Pi_{[\cdot]}$ UC-securely realizes the functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ in the $\mathcal{F}_{\mathrm{BC}}$-hybrid model. The protocol has communication complexity $\mathcal{O}(n\kappa)$ bits and $\mathcal{BC}\big(n\kappa, n\big)$.*

PROOF: The communication complexity follows easily from the protocol. We next prove the security, considering the following two cases.

Case I — When D is honest: We first claim that in this case, no honest party will output $\mathsf{Failure}$; this easily follows from the fact that an honest D will distribute consistent shares and only the corrupted share-holders (at most t) will accuse D and such accusations will be resolved correctly by D. Let $T \subset \mathcal{P}$ be the set of parties under the control of $\mathcal{A}$ during the protocol $\Pi_{[\cdot]}$; we present a simulator $\mathcal{S}_{[\cdot]}^{\mathrm{HD}}$ (interacting with the functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$) for $\mathcal{A}$ in Figure 8. The high level idea behind the simulator is the following: the simulator interacts with $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ and obtains the shares and opening information of the corrupted parties, along with all the committed shares and sends the same to the real-world adversary; the simulator then simulates the rest of the protocol steps of $\Pi_{[\cdot]}$ on the behalf of the honest parties (including D). Any accusation by a (corrupted) share-holder can be easily resolved by the simulator, as it knows the corresponding share and opening information (as obtained from the functionality), which it can reveal. It follows easily that the simulated view has exactly the same distribution as the view of the real-world adversary in $\Pi_{[\cdot]}$.

Case 2 — When D is Corrupted: We first note that there exists at least $t+1$ honest parties in $\mathcal{P}$ and that there exists only a unique polynomial of degree at most t passing through a set of $t+1$ or more distinct points. With these facts, we next prove the security with respect to a corrupted D. Let $T \subset \mathcal{P}$ be the set of parties under the control of $\mathcal{A}$ including D, during the protocol $\Pi_{[\cdot]}$; we present a simulator $\mathcal{S}_{[\cdot]}^{\mathrm{CD}}$ (interacting with the functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$) for $\mathcal{A}$ in Figure 9.

Protocol $\Pi_{[\cdot]}$

The public input to the protocol is a publicly known commitment $\mathbf{C}$ available to the parties in $\mathcal{P}$, while the private input for D is a secret s and randomness pair (g, h), such that $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(s; g, h)$ holds. For session ID sid, D and the parties in $\mathcal{P}$ do the following:

Round 1 (Share Distribution and Broadcasting Commitments) — D does the following:
- Select three random polynomials $f(\cdot), g(\cdot)$ and $h(\cdot)$ of degree at most t, subject to the condition that $f(0) = s, g(0) = g$ and $h(0) = h$.
- Corresponding to every $P_i \in \mathcal{P}$, compute the *share* $f_i = f(\alpha_i)$ and the *opening information* $g_i = g(\alpha_i), h_i = h(\alpha_i)$ and the commitment $\mathbf{C}_{f_i,g_i,h_i} = \mathsf{Comm}_{\mathsf{ck}}(f_i; g_i, h_i)$.
- Corresponding to every $P_i \in \mathcal{P}$, send $(\mathsf{sid}, i, (f_i, g_i, h_i))$ to the party P_i. In addition, call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, \mathsf{D}, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}}, \mathcal{P})$.

Round 2 (Consistency Verification and Complaints) — Every party $P_i \in \mathcal{P}$ does the following:
- Receive $(\mathsf{sid}, i, \mathsf{D}, (f_i, g_i, h_i))$ from D and $(\mathsf{sid}, i, \mathsf{D}, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ from $\mathcal{F}_{\mathrm{BC}}$.
- Using the homomorphic property of commitments, verify
  - if there exists polynomials of degree at most t, say $f'(\cdot), g'(\cdot)$ and $h'(\cdot)$ such that $\mathbf{C}_{f_j,g_j,h_j}$ is $\mathsf{Comm}_{\mathsf{ck}}(f'(\alpha_j); g'(\alpha_j), h'(\alpha_j))$ [a] for every $P_j \in \mathcal{P}$.
  - whether the $\mathbf{C}$ is same as $\mathsf{Comm}_{\mathsf{ck}}(f'(0); g'(0), h'(0))$.

  If any of the above tests fail then output $(\mathsf{sid}, i, \mathsf{Failure})$ and halt.
- Verify whether $\mathbf{C}_{f_i,g_i,h_i} \stackrel{?}{=} \mathsf{Comm}_{\mathsf{ck}}(f_i; g_i, h_i)$. If the verification fails then call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, i, (\mathtt{Unhappy}, \mathsf{D}), \mathcal{P})$.

Local Computation (at the end of Round 2) — Every party P_i in $\mathcal{P}$ does the following:
- Construct a set $\mathcal{W}_i$ initialized to $\emptyset$ and add $P_j \in \mathcal{P}$ to $\mathcal{W}_i$ if corresponding to party P_j the message $(\mathsf{sid}, i, j, (\mathtt{Unhappy}, \mathsf{D}))$ is received from[b] $\mathcal{F}_{\mathrm{BC}}$.
- If $|\mathcal{W}_i| > t$, then output $(\mathsf{sid}, i, \mathsf{Failure})$ and halt.

Round 3 (Resolving Complaints) — D does the following:
- Corresponding to each $P_i \in \mathcal{W}_{\mathsf{D}}$, call $\mathcal{F}_{\mathrm{BC}}$ with the message $(\mathsf{sid}, \mathsf{D}, (\mathtt{Resolve}, P_i, f_i, g_i, h_i), \mathcal{P})$.

Local Computation (at the end of Round 3) — every party $P_i \in \mathcal{P}$ does the following:
- If there exists a $P_k \in \mathcal{W}_i$ corresponding to which the message $(\mathsf{sid}, i, \mathsf{D}, (\mathtt{Resolve}, P_k, f_k, g_k, h_k))$ is received from $\mathcal{F}_{\mathrm{BC}}$ such that $\mathbf{C}_{f_k,g_k,h_k} \neq \mathsf{Comm}_{\mathsf{ck}}(f_k; g_k, h_k)$, then output $(\mathsf{sid}, i, \mathsf{Failure})$ and halt.
- Else output $[s]_i$ computed as follows and halt:
  - If $P_i \in \mathcal{P} \setminus \mathcal{W}_i$, then $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ where f_i, g_i, h_i is received from D in **Round 1**.
  - If $P_i \in \mathcal{W}_i$, then $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ where f_i, g_i, h_i is received from D in **Round 3**.

[a] This is done using a standard procedure based on the properties of Vandermonde matrix; see for example [19].
[b] The contents of $\mathcal{W}_i$ will be the same for each honest party P_i in $\mathcal{P}$.

Fig. 7. Protocol for UC-secure realizing $\mathcal{F}_{\mathrm{GEN}[\cdot]}$

Simulator $\mathcal{S}^{\mathbf{HD}}_{[\cdot]}$

The simulator plays the role of the honest parties (including D) and simulates each step of the protocol $\Pi_{[\cdot]}$ as follows. The communication of the $\mathcal{Z}$ with the adversary $\mathcal{A}$ is handled as follows: Every input value received by the simulator from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to the simulator's output tape (to be read by the environment $\mathcal{Z}$). The simulator then does the following for the session ID sid:

- Interact with $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ and obtain $(\mathsf{sid}, i, (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}}))$ corresponding to every corrupted party $P_i \in T$.
- On the behalf of D, send $(\mathsf{sid}, i, (f_i, g_i, h_i))$ to $\mathcal{A}$, corresponding to every $P_i \in T$. In addition, send $(\mathsf{sid}, \mathsf{D}, i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$ to $\mathcal{A}$ on the behalf of $\mathcal{F}_{\mathrm{BC}}$, corresponding to each $P_i \in T$.
- On receiving $(\mathsf{sid}, i, \mathcal{P}, (\mathtt{Unhappy}, \mathsf{D}))$ as the message to $\mathcal{F}_{\mathrm{BC}}$ from $\mathcal{A}$ on the behalf of any $P_i \in T$, send $(\mathsf{sid}, \mathsf{D}, i, (\mathtt{Resolve}, P_i, f_i, g_i, h_i))$ to $\mathcal{A}$ as the message from $\mathcal{F}_{\mathrm{BC}}$ on the behalf of D.

The simulator then outputs $\mathcal{A}$'s output and terminate.

Fig. 8. Simulator for the adversary $\mathcal{A}$ corrupting at most t parties in the set $T \subset \mathcal{P} \setminus \mathsf{D}$ in the protocol $\Pi_{[\cdot]}$.

It follows easily that the simulated view is computationally indistinguishable from the view of the real-world adversary; otherwise we can use the corresponding distinguisher to break the binding property of the underlying commitment scheme. □

Simulator $\mathcal{S}_{[\cdot]}^{\mathbf{CD}}$

The simulator plays the role of the honest parties and simulates each step of the protocol $\Pi_{[\cdot]}$ as follows. The communication of the $\mathcal{Z}$ with the adversary $\mathcal{A}$ is handled as follows: Every input value received by the simulator from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to the simulator's output tape (to be read by the environment $\mathcal{Z}$). The simulator then does the following for the session ID sid:

- Play the role of $n - |T|$ honest parties and interact with $\mathcal{A}$ as per the protocol $\Pi_{[\cdot]}$.
- If $\mathsf{Failure}$ is obtained during the simulated execution of the protocol due to the fact that the committed shares and the corresponding randomness do not lie on polynomials of degree at most t, then send three arbitrary polynomials of degree more than t on the behalf of D to the functionality $\mathcal{F}_{\text{GEN}[\cdot]}$.
- Else define three polynomials $\hat{f}(\cdot), \hat{g}(\cdot)$ and $\hat{h}(\cdot)$ of degree at most t, such that $\hat{f}(\alpha_i) = f_i, \hat{g}(\alpha_i) = g_i$ and $\hat{h}(\alpha_i) = h_i$ holds for every honest party $P_i \notin T$, where f_i and (g_i, h_i) are the corresponding share and opening information respectively which are obtained by P_i during the simulated run of [a] of $\Pi_{[\cdot]}$. Then send the polynomials $\hat{f}(\cdot), \hat{g}(\cdot)$ and $\hat{h}(\cdot)$ on the behalf of D to $\mathcal{F}_{\text{GEN}[\cdot]}$.

The simulator then outputs $\mathcal{A}$'s output and terminate.

[a] Note that $\hat{f}(\cdot), \hat{g}(\cdot)$ and $\hat{h}(\cdot)$ are well defined as there exists $|n| - |T| > t + 1$ honest parties in $\mathcal{P}$.

Fig. 9. Simulator for the adversary $\mathcal{A}$ corrupting at most t parties in the set $T \subset \mathcal{P}$ including D during the protocol $\Pi_{[\cdot]}$.

C.2 Protocols for Realizing $\mathcal{F}_{\text{COMMITTEE}}$ and $\mathcal{F}_{\text{BC}}$

The Committee Selection Protocol: Functionality $\mathcal{F}_{\text{COMMITTEE}}$ can be realized using various standard ways; moreover, the functionality will be invoked at most $(t + 1)$ times in our MPC protocol; t times corresponding to t failed sub-circuit evaluations plus once for initial selection of a committee. As this cost is independent of the circuit size $|\mathsf{ckt}|$ (but rather $\mathsf{Poly}(n)$), we give only a high level sketch of one of the possible instantiations of $\mathcal{F}_{\text{COMMITTEE}}$, based on a computationally secure pseudo-random number generator (PRNG) [28]. Assume we have a PRNG $\mathcal{R}_k(\cdot)$ with seed k, which outputs values in the range $1, \ldots, n$. Then each time a committee needs to be formed, the parties in $\mathcal{P}$ can agree on a random seed k; this can be done via standard method, say by coin-flipping (or executing an instance of $\Pi_{[\cdot]}$ on the behalf of each party). Then the parties can (locally) run $\mathcal{R}$ with the obtained key, till they obtain the desired committee. It follows via the security of $\mathcal{R}$, that the committee selected like this is indeed a uniformly random committee of parties with high probability. We can simplify further by putting up a random seed in the CRS, rather than sampling a random seed on the fly every time a committee needs to be formed.

The Broadcast Protocol: Assuming a PKI set-up, the well known Dolev-Strong (DS) broadcast protocol [15] allows a sender $\mathsf{Sen} \in \mathcal{P}$ to reliably broadcast a message m of size ℓ to a set of parties $\mathcal{X} \subseteq \mathcal{P}$, provided $\mathcal{X}$ has at least one honest party; the protocol can be used to realize $\mathcal{F}_{\text{BC}}$. As stated in [17], using the DS protocol, it costs the parties in $\mathcal{X} \cup \{\mathsf{Sen}\}$ a total communication of $\mathcal{O}(|\mathcal{X}|^3 \cdot \ell \cdot \kappa)$ bits over the point-to-point channels to enable the Sen to broadcast m to the parties in $\mathcal{X}$. As the protocol is well known in the literature, we avoid giving the details here and instead refer the interested readers to [16] for the details. We also note that [17] suggests an improved proposal for realizing $\mathcal{F}_{\text{BC}}$ with a communication complexity of $\mathcal{O}(|\mathcal{X}| \cdot \ell + |\mathcal{X}|^4 \cdot (|\mathcal{X}| + \kappa))$ bits, but with a restriction on the size of ℓ, namely $\ell = \omega(|\mathcal{X}|^2 \cdot (|\mathcal{X}| + \kappa))$. We make use of this proposal for estimating the communication complexity of our MPC protocol in Section 3.3.

D Supporting Sub-Protocols

In this appendix, we present the details for the sub-protocols which enable a number of tasks such as conversion from $[\cdot]$-sharing to $\langle\cdot\rangle$-sharing and vice-versa, generating a random $[\cdot]$-sharing of 0 and a protocol for dishonest-majority sub-circuit evaluation in the player elimination framework. These sub-protocols are used in our main protocol and we do not explicitly prove any security property of these sub-protocols. Rather we focus on the security proof for our MPC protocol that invokes these sub-protocols. We note that it is therefore not necessary to prove any form of security for these sub-protocols separately. To modularize the complexity analysis of our main protocol, we however analyze the communication complexity of each of the subprotocols in turn.

D.1 Protocol for Converting a $\langle\cdot\rangle$-sharing to a $[\cdot]$-sharing

The protocol in Figure 10 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 functionalities $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ and $\mathcal{F}_{\mathrm{BC}}$.

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 $\mathcal{X}$ as a part of $\langle s\rangle_i$, 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 $\mathcal{F}_{\mathrm{GEN}[\cdot]}$. 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 $\mathcal{F}_{\mathrm{BC}}$ 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 3, 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 3. *The communication complexity of protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$ is $\mathcal{BC}(|\mathcal{X}|^2\kappa, n)$ plus the complexity of $\mathcal{O}(|\mathcal{X}|)$ invocations to the realization of the functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$.*

Protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$

For session id sid, every party $P_i \in \mathcal{P}$ participates with either $(\mathsf{sid}, i, \langle s\rangle_i)$ or (sid, i) and does the following:

- If $P_i \in \mathcal{X}$, interpret $\langle s\rangle_i$ as $(s_i, u_i, v_i, \{\mathbf{C}^{P_i}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}})$ and invoke $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, i, \{\mathbf{C}^{P_i}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}}, \mathcal{P})$
- Receive $(\mathsf{sid}, i, k, \{\mathbf{C}^{P_k}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}})$ from $\mathcal{F}_{\mathrm{BC}}$ for every $P_k \in \mathcal{X}$ (who acted as the sender).
- If there exists a pair of parties $P_a, P_b \in \mathcal{X}$, such that $\{\mathbf{C}^{P_a}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}} \neq \{\mathbf{C}^{P_b}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ and halt; if there are multiple such pairs (P_a, P_b) the select the one with the least index a and b. Else set $\{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}} = \{\mathbf{C}^{P_\alpha}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}}$ to be the reference set of commitments, where P_α is the least indexed party in $\mathcal{P}$.
- If $P_i \in \mathcal{X}$, act as a D and call $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ with $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$ where $f^{(i)}(\cdot), g^{(i)}(\cdot)$ and $h^{(i)}(\cdot)$ are random polynomials of degree at most t, subject to the condition that $f^{(i)}(0) = s_i, g^{(i)}(0) = u_i$ and $h^{(i)}(0) = v_i$. If $P_i \in \mathcal{P} \setminus \mathcal{X}$, invoke $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ with $(\mathsf{sid}, i, k, \mathbf{C}_{s_k,u_k,v_k})$ for every $P_k \in \mathcal{X}$, where $\mathbf{C}_{s_k,u_k,v_k}$ is obtained from the reference set of commitments. Receive $(\mathsf{sid}, i, k, \mathsf{Failure})$ or $(\mathsf{sid}, i, k, [s_k]_i)$ from $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ for every $P_k \in \mathcal{X}$
- Output $(\mathsf{sid}, i, \mathsf{Failure}, P_k)$ and halt if $(\mathsf{sid}, i, P_k, \mathsf{Failure})$ is received from $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ corresponding to any $P_k \in \mathcal{X}$. Otherwise, locally compute $[s]_i = \sum_{P_k\in\mathcal{X}} [s_k]_i$, output $(\mathsf{sid}, i, \mathsf{Success}, [s]_i)$ and halt.

Fig. 10. Protocol for Converting $\langle\cdot\rangle$-sharing to $[\cdot]$-sharing in the $(\mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\mathrm{GEN}[\cdot]})$-hybrid Model

D.2 Protocol for Generating $\langle\cdot\rangle$-sharing of a Committed Secret

To enable a 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, we follow the same three round *share-accuse-resolve* approach as used in the protocol $\Pi_{[\cdot]}$ (for realizing $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ in Figure 8), except that now the resultant sharing is an additive sharing instead of a Shamir-sharing. More specifically, every $P_i \in \mathcal{P}$ holds a (publicly known) commitment $\mathbf{C}_{f,g,h}$ (see Figure 11). The dealer D holds the secret $f \in \mathbb{F}_p$ and randomness pair (g, h), such that $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(f; g, h)$; 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 $\Pi_{\langle\cdot\rangle}$ stated in Lemma 4 follows from the protocol steps.

Lemma 4. *The communication complexity of protocol $\Pi_{\langle\cdot\rangle}$ is $\mathcal{O}(|\mathcal{X}|\kappa)$ and $\mathcal{BC}(|\mathcal{X}|\kappa, n)$.*

Protocol $\Pi_{\langle\cdot\rangle}$

For session ID sid, every $P_i \in \mathcal{P}$ participates with $(\mathsf{sid}, i, \mathsf{D}, \mathbf{C}_{f,g,h}, \mathcal{X})$ where $\mathbf{C}_{f,g,h}$ is a (publicly known) commitment. The dealer D participates with $(\mathsf{sid}, \mathsf{D}, f, g, h, \mathcal{X})$ where f is the secret and (g, h) is the randomness pair, such that $\mathbf{C} = \mathsf{Comm}_{\mathsf{ck}}(f; g, h)$. The parties in $\mathcal{P}$ do the following:

Round 1 (Share Distribution and Broadcasting Commitments): Only D does the following:
- Corresponding to every $P_i \in \mathcal{X}$, select a random *share* s_i and a random pair of *opening information* u_i, v_i, subject to the condition that $\sum_{P_i \in \mathcal{X}} s_i = f, \sum_{P_i \in \mathcal{X}} u_i = g$ and $\sum_{P_i \in \mathcal{X}} v_i = h$, and compute the commitment $\mathbf{C}_{s_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(s_i; u_i, v_i)$. Send $(\mathsf{sid}, i, \mathsf{D}, s_i, u_i, v_i)$ to the party P_i.
- Call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, \mathsf{D}, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}}, \mathcal{P})$ to broadcast $\{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}}$ to all the parties in $\mathcal{P}$.

Round 2 (Consistency Verification and Complaints): Every party $P_i \in \mathcal{P}$ does the following:
- Receive $(\mathsf{sid}, i, \mathsf{D}, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$ from $\mathcal{F}_{\mathrm{BC}}$. Additionally if $P_i \in \mathcal{X}$, then receive $(\mathsf{sid}, i, \mathsf{D}, s_i, u_i, v_i)$ from D.
- Verify if $\odot_{P_j \in \mathcal{X}} \mathbf{C}_{s_j,u_j,v_j} \stackrel{?}{=} \mathbf{C}_{f,g,h}$ (homomorphically). If the verification fails, then output $(\mathsf{sid}, i, \mathsf{D}, \mathsf{Failure})$ and halt.
- If $P_i \in \mathcal{X}$ then verify whether $\mathbf{C}_{s_i,u_i,v_i} \stackrel{?}{=} \mathsf{Comm}_{\mathsf{ck}}(s_i; u_i, v_i)$. If the verification fails then call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, i, (\texttt{Unhappy}, i, \mathsf{D}), \mathcal{P})$.
- Construct a set $\mathcal{W}_i$ initialized to $\emptyset$ and add $P_j \in \mathcal{X}$ to $\mathcal{W}_i$ if $(\mathsf{sid}, i, j, (\texttt{Unhappy}, j, \mathsf{D}))$ is received from[a] $\mathcal{F}_{\mathrm{BC}}$ corresponding to P_j.

Round 3 (Resolving Complaints): Only D does the following:
- Corresponding to each $P_i \in \mathcal{W}_{\mathsf{D}}$, call $\mathcal{F}_{\mathrm{BC}}$ with the message $(\mathsf{sid}, \mathsf{D}, (\texttt{Resolve}, i, s_i, u_i, v_i), \mathcal{P})$.

Local Computation (at the end of Round 3): Every party $P_i \in \mathcal{P}$ does the following:
- If there exists a $P_k \in \mathcal{W}_i$ corresponding to which the message $(\mathsf{sid}, i, \mathsf{D}, (\texttt{Resolve}, k, s_k, u_k, v_k))$ is received from $\mathcal{F}_{\mathrm{BC}}$ such that $\mathbf{C}_{s_k,u_k,v_k} \neq \mathsf{Comm}_{\mathsf{ck}}(s_k; u_k, v_k)$, then output $(\mathsf{sid}, i, \mathsf{D}, \mathsf{Failure})$ and halt.
- Else every $P_i \in \mathcal{P} \setminus \mathcal{X}$ outputs $(\mathsf{sid}, i, \mathsf{D}, \mathsf{Success})$ and halts, while every $P_i \in \mathcal{X}$ does the following:
  - If $P_i \in \mathcal{X} \setminus \mathcal{W}_i$, then set $\langle f \rangle_i = (s_i, u_i, v_i, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$, where (s_i, u_i, v_i) was received from D and $\{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}}$ was received from $\mathcal{F}_{\mathrm{BC}}$ at the end of **Round 1**. Output $(\mathsf{sid}, i, \mathsf{D}, \mathsf{Success}, \langle f \rangle_i)$ and halt.
  - Else if $P_i \in \mathcal{W}_i$, then set $\langle f \rangle_i = (s_i, u_i, v_i, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$, where (s_i, u_i, v_i) was received from $\mathcal{F}_{\mathrm{BC}}$ (corresponding to D) at the end of **Round 3** and $\{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}}$ was received from $\mathcal{F}_{\mathrm{BC}}$ at the end of **Round 1**. Output $(\mathsf{sid}, i, \mathsf{Success}, \mathsf{D}, \langle f \rangle_i)$ and halt.

[a] The contents of $\mathcal{W}_i$ will be the same for each honest party P_i in $\mathcal{P}$.

Fig. 11. Protocol $\Pi_{\langle\cdot\rangle}$ for Verifiably $\langle\cdot\rangle$-sharing an Existing Committed Secret

D.3 Protocol for Transforming $[\cdot]$-sharing to $\langle\cdot\rangle$-sharing

Protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$ in Figure 12 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 $\Pi_{[\cdot]\to\langle\cdot\rangle}$ is stated in Lemma 5 which follows from the fact that n invocations to $\Pi_{\langle\cdot\rangle}$ are done in the protocol.

Lemma 5. *The communication complexity of protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$ is $\mathcal{O}(n|\mathcal{X}|\kappa)$ and $\mathcal{BC}(n|\mathcal{X}|\kappa, n)$.*

Protocol $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$

For session ID sid, every party $P_i \in \mathcal{P}$ participates in the protocol with $(\mathsf{sid}, i, [s]_i, \mathcal{X})$, where $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j\in\mathcal{P}})$ and does the following:

Verifiably $\langle\cdot\rangle$-sharing the Share and Opening Information in $[s]_i$. Act as a dealer D and participate in an instance of $\Pi_{\langle\cdot\rangle}$ with input $(\mathsf{sid}, i, f_i, g_i, h_i, \mathcal{X})$. For every $P_k \in \mathcal{P}$, participate in the instance of $\Pi_{\langle\cdot\rangle}$ corresponding to the dealer P_k with input $(\mathsf{sid}, i, k, \mathbf{C}_{f_k,g_k,h_k}, \mathcal{X})$.

Identifying the Correctly $\langle\cdot\rangle$-shared Shares of s and Generating $\langle s\rangle_{\mathcal{X}}$. If $P_i \in \mathcal{P} \setminus \mathcal{X}$, then output (sid, i) and halt. Else construct a set $\mathcal{H}$ initialized to $\emptyset$.

- Include $P_k \in \mathcal{P}$ to $\mathcal{H}$ if $(\mathsf{sid}, i, k, \mathsf{Success}, \langle f_k\rangle_i)$ is the output for the instance of $\Pi_{\langle\cdot\rangle}$ where P_k acted as the dealer.
- Without loss of generality, let[a] $\mathcal{H} = \{P_1, \ldots, P_{|\mathcal{H}|}\}$ and let $c_1, \ldots, c_{|\mathcal{H}|}$ be the publicly known Lagrange interpolation coefficients, such that $c_1 f_1 + \ldots + c_{|\mathcal{H}|} f_{|\mathcal{H}|} = s$. Then locally compute $\langle s\rangle_i = c_1\langle f_1\rangle_i + \ldots + c_{|\mathcal{H}|}\langle f_{|\mathcal{H}|}\rangle_i$, output $(\mathsf{sid}, i, \langle s\rangle_i)$ and halt.

[a] The set $\mathcal{H}$ will be of size more than $t+1$.

Fig. 12. Protocol $\Pi_{[\cdot]\rightarrow\langle\cdot\rangle}$ for Converting an $[\cdot]$-sharing to $\langle\cdot\rangle$-sharing.

D.4 Protocol for Generating Random $[\cdot]$-sharing of 0

We present a protocol $\Pi_{\textsc{RandZero}[\cdot]}$ to generate a random $[\cdot]$-sharing of 0. Before presenting the protocol $\Pi_{\textsc{RandZero}[\cdot]}$, we first describe the "zero-knowledge" (ZK) functionality $\mathcal{F}_{\mathrm{ZK.BC}}$, used in protocol $\Pi_{\textsc{RandZero}[\cdot]}$.

Functionality $\mathcal{F}_{\mathrm{ZK.BC}}$: The functionality (see Figure 13) 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.

Functionality $\mathcal{F}_{\mathrm{ZK.BC}}$

$\mathcal{F}_{\mathrm{ZK.BC}}$ interacts with a prover $P_j \in \mathcal{P}$ and the set of n verifiers $\mathcal{P} = \{P_1, \ldots, P_n\}$ and the adversary $\mathcal{S}$ and is parameterized by the commitment key ck of a double-trapdoor homomorphic commitment scheme.

- Upon receiving $(\mathsf{sid}, j, \mathbf{C}, u, v)$ from the prover P_j and (sid, j, i) from every party $P_i \in \mathcal{P} \setminus \{P_j\}$, the functionality sends $(\mathsf{sid}, i, \mathbf{C})$ to every party $P_i \in \mathcal{P}$ and $\mathcal{S}$ and halts if $\mathbf{C} \stackrel{?}{=} \mathsf{Comm}_{\mathsf{ck}}(0; u, v)$ is true. Else the functionality sends $(\mathsf{sid}, i, \perp)$ to every party $P_i \in \mathcal{P}$ and $\mathcal{S}$ and halts.

Fig. 13. The Ideal Functionality for ZK Proof of Committing Zero

A protocol $\Pi_{\mathrm{ZK.BC}}$ realizing $\mathcal{F}_{\mathrm{ZK.BC}}$ can be designed in the CRS model using standard techniques (see for example, [20]). Since we invoke it only n times (independent of $|\mathsf{ckt}|$) in the MPC protocol Π_f[9], we skip the details of $\Pi_{\mathrm{ZK.BC}}$, except confirming that it costs $\mathcal{O}(\mathsf{Poly}(n)\kappa)$ bits of communication.

Our protocol $\Pi_{\textsc{RandZero}[\cdot]}$ invokes the following ideal functionalities: $\mathcal{F}_{\mathrm{ZK.BC}}, \mathcal{F}_{\textsc{Gen}[\cdot]}$. 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 $\mathcal{F}_{\textsc{Gen}[\cdot]}$ 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 $\mathcal{F}_{\textsc{Gen}[\cdot]}$. Since there exists at least one honest party in this set of dealers, the resultant sharing will be indeed a random sharing of 0. Since this protocol is invoked only once in our main MPC protocol Π_f, we avoid giving details of the communication complexity of the protocol. However assuming standard realization of $\mathcal{F}_{\mathrm{ZK.BC}}$, the protocol has complexity $\mathcal{O}(\mathsf{Poly}(n)\kappa)$.

[9] Recall that we require only one call to $\Pi_{\textsc{RandZero}[\cdot]}$ in our MPC protocol for the output gate, which would in turn require n invocations of $\Pi_{\mathrm{ZK.BC}}$.

Protocol $\Pi_{\text{RANDZERO}[\cdot]}$

For the session id sid, every party $P_i \in \mathcal{P}$ participates with (sid, i) and does the following:

Publicly Committing 0:

- Set $r_i = 0$ and randomly select $u_i, v_i \in \mathbb{F}_p$ and compute $\mathbf{C}_{r_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(r_i; u_i, v_i)$.
- Act as a prover and call $\mathcal{F}_{\text{ZK.BC}}$ with $(\mathsf{sid}, i, \mathbf{C}_{r_i,u_i,v_i}, u_i, v_i)$. Corresponding to every prover $P_j \in \mathcal{P} \setminus P_i$, participate in $\mathcal{F}_{\text{ZK.BC}}$ with (sid, j, i).
- Construct a set $\mathcal{T}_i$, initialized to $\emptyset$ and include P_j in $\mathcal{T}_i$ if corresponding to the prover P_j, $(\mathsf{sid}, i, \mathbf{C}_{r_i,u_j,v_j})$ is received from $\mathcal{F}_{\text{ZK.BC}}$.

$[\cdot]$**-sharing** 0:

- Select three random polynomials $f^{(i)}(\cdot), g^{(i)}(\cdot)$ and $h^{(i)}(\cdot)$ each of degree at most t, subject to the condition that $f^{(i)}(0) = r_i, g^{(i)}(0) = u_i$ and $h^{(i)}(0) = v_i$.
- Act as a D and call $\mathcal{F}_{\text{GEN}[\cdot]}$ with $(\mathsf{sid}, P_i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$. Corresponding to every dealer $P_j \in \mathcal{T}_i$, participate in $\mathcal{F}_{\text{GEN}[\cdot]}$ with $(\mathsf{sid}, i, j, \mathbf{C}_{r_j,u_j,v_j})$.
- If corresponding to any $P_j \in \mathcal{T}_i$, $(\mathsf{sid}, i, P_j, \mathsf{Failure})$ is received from $\mathcal{F}_{\text{GEN}[\cdot]}$, then remove P_j from $\mathcal{T}_i$.
- Locally compute $[0]_i = \sum_{P_j \in \mathcal{T}_i} [r_j]_i$, where $(\mathsf{sid}, i, j, [r_j]_i)$ is received from $\mathcal{F}_{\text{GEN}[\cdot]}$ corresponding to $P_j \in \mathcal{T}_i$. Output $(\mathsf{sid}, i, [0]_i)$ and halt.

Fig. 14. Protocol $\Pi_{\text{RANDZERO}[\cdot]}$ for generating a random $[\cdot]$-sharing of 0.

E Protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ for $\langle\cdot\rangle$-shared Evaluation of a Circuit

As evident from the high level description of protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ in Section 3.2, the major step of the protocol $\Pi_{\mathsf{C}}^{\text{NR}}$ is the preparation stage for generating the shared-triplets. Towards constructing the preparation stage protocol and protocol $\Pi_{\mathsf{C}}^{\text{NR}}$, we begin with the building blocks and sub-protocols most of which are taken from [12] and rest are modified according to our need. Many of the sub-protocols are described with respect to a set of parties $\mathcal{X} \subset \mathcal{P}$, where we assume that $\mathcal{X}$ contains at least one honest party.

Strong Semi-honest Secure Two-party Multiplication Protocol. Protocol $\Pi_{\text{MULT}}(a, b) \rightarrow (c_1, c_2)$ is a two-party protocol. The inputs of the first and second party are a and b respectively. The outputs to the first and second party are c_1 and c_2 respectively. It holds that c_1 is random in $\mathbb{F}_p$ and $c_1 + c_2 = a \cdot b$. Informally the protocol satisfies the following properties (for the complete formal details see [12]):

- The protocol is secure even if the adversary maliciously chooses the randomness for the corrupted parties (this is the reason [12] calls the protocol as *strong semi-honest* secure).
- The view of the protocol commits the adversary to his randomness and given the view and the randomness it is possible to verify whether any party deviated from the protocol.

In our context, the second property of Π_{MULT} is very crucial, as it enables an honest party involved in Π_{MULT} to identify any malicious behavior of its partner in the protocol when the individual randomness are revealed. There are various standard ways for instantiating $\Pi_{\text{MULT}}(a, b)$, based on variety of standard assumptions, such as homomorphic encryption, oblivious transfer (OT), etc. An instantiation based on Paillier encryption with communication complexity $\mathcal{O}(\kappa)$ is provided in [12] (for details see [12]).

Semi-honest Secure Triple Generation Protocol. The protocol Π_{TRIPLE} (see Figure 15) uses the two party protocol Π_{MULT} as a sub-protocol and allows a set of parties $\mathcal{X} \subset \mathcal{P}$ to generate one $\langle\cdot\rangle$-shared multiplication triplet $(\langle a\rangle_{\mathcal{X}}, \langle b\rangle_{\mathcal{X}}, \langle c\rangle_{\mathcal{X}})$. The protocol is executed assuming semi-honest adversary. The protocol is based on the following idea: every party $P_i \in \mathcal{X}$ selects a random a_i and b_i and commits the same. Then we set a and b to be the sum of all a_is and b_is. For setting c as $a \cdot b$, every pair of parties $P_i, P_j \in \mathcal{X}$ need to securely compute the "cross-terms" $a_i \cdot b_j$ and $a_j \cdot b_i$, for which they execute two instances of Π_{MULT}. Once P_i computes its c_i, it publicly commits the same. Instantiating the calls to Π_{MULT} with that of [12] (based on the Paillier encryption), protocol Π_{TRIPLE} has communication complexity of $\mathcal{O}(|\mathcal{X}|^2\kappa)$ and $\mathcal{BC}\big(|\mathcal{X}|\kappa, |\mathcal{X}|\big)$.

Protocol Π_{TRIPLE}

The public input to the protocol is a set of parties $\mathcal{X} \subset \mathcal{P}$ containing at least one honest party. For the session id sid, every party $P_i \in \mathcal{X}$ participates with (sid, i) and does the following:

- Randomly select shares a_i, b_i.
- For all $P_j \in \mathcal{X} \setminus P_i$, run $\Pi_{\text{MULT}}(a_i, b_j) \rightarrow (d_{ij}, e_{ji})$ as party 1.
- For all $P_j \in \mathcal{X} \setminus P_i$, run $\Pi_{\text{MULT}}(a_j, b_i) \rightarrow (d_{ji}, e_{ij})$ as party 2.
- Set $c_i = a_i \cdot b_i + \sum_{P_j \in \mathcal{X} \setminus P_i} d_{ij} + \sum_{P_j \in \mathcal{X} \setminus P_i} e_{ij}$.
- Randomly select q_i, r_i, s_i, t_i, u_i and v_i and compute the commitments $\mathbf{C}_{a_i,q_i,r_i} = \mathsf{Comm}_{\mathsf{ck}}(a_i; q_i, r_i), \mathbf{C}_{b_i,s_i,t_i} = \mathsf{Comm}_{\mathsf{ck}}(b_i; s_i, t_i)$ and $\mathbf{C}_{c_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(c_i; u_i, v_i)$. Call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, \mathbf{C}_{a_i,q_i,r_i}, \mathbf{C}_{b_i,s_i,t_i}, \mathbf{C}_{c_i,u_i,v_i}, \mathcal{X})$.
- Corresponding to each $P_j \in \mathcal{X}$, receive $(\mathsf{sid}, i, j, \mathbf{C}_{a_j,q_j,r_j}, \mathbf{C}_{b_j,s_j,t_j}, \mathbf{C}_{c_j,u_j,v_j})$ from $\mathcal{F}_{\text{BC}}$.
- Set $\langle a \rangle_i = (a_i, q_i, r_i, \{\mathbf{C}_{a_j,q_j,r_j}\}_{P_j \in \mathcal{X}}), \langle b \rangle_i = (b_i, s_i, t_i, \{\mathbf{C}_{b_j,s_j,t_j}\}_{P_j \in \mathcal{X}})$ and $\langle c \rangle_i = (c_i, u_i, v_i, \{\mathbf{C}_{c_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$. Output $(\mathsf{sid}, i, \langle a \rangle_i, \langle b \rangle_i, \langle c \rangle_i)$ and halt.

Fig. 15. Protocol for Generating One $\langle\cdot\rangle$-shared Multiplication Triple Assuming No Active Corruptions.

Functionality for Generating $\langle\cdot\rangle$-sharing of a Random Value. Functionality $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ (presented in Figure 16) generates an $\langle\cdot\rangle$-shared random value within a designated set of parties $\mathcal{X} \subset \mathcal{P}$, where each party in $\mathcal{X}$ "contributes" its "part" of the share and opening information for the shared random value.

Functionality $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$

The functionality interacts with a designated set of parties $\mathcal{X} \subset \mathcal{P}$ containing at least one honest party and the adversary $\mathcal{S}$ and is parametrized by the commitment key ck of a double-trapdoor commitment scheme. For the session id sid, the functionality does the following:

- On receiving $(\mathsf{sid}, i, s_i, u_i, v_i)$ from each party $P_i \in \mathcal{X}$, compute $s = \sum_{P_i \in \mathcal{X}} s_i, u = \sum_{P_i \in \mathcal{X}} u_i$ and $v = \sum_{P_i \in \mathcal{X}} v_i$ and $\mathbf{C}_{s,u,v} = \mathsf{Comm}_{\mathsf{ck}}(s; u, v)$. In addition, for each $P_i \in \mathcal{X}$, compute $\mathbf{C}_{s_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(s_i; u_i, v_i)$. Finally send $(\mathsf{sid}, i, \mathbf{C}_{s,u,v}, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$ to every $P_i \in \mathcal{X}$ and halt.

Fig. 16. Functionality for Generating $\langle\cdot\rangle$-shared Random Value for a Designated Set $\mathcal{X} \subset \mathcal{P}$ with Dishonest-majority.

In [12], a realization of $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ based on UC-secure multi-party commitment scheme was presented in the common reference string (CRS) model. The UC secure multi-party commitment scheme is further constructed using a CCA-secure encryption and the double-trapdoor homomorphic commitment scheme introduced in Section 2. Specifically, the following was shown; we refer to [12] for the details of the instantiation of $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$.

Lemma 6 ([12]). *Assuming CCA-secure encryption and double-trapdoor homomorphic commitment scheme, it is possible to (κ, s)-securely realize $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ in the $(\mathcal{F}_{\text{CRS}}, \mathcal{F}_{\text{BC}})$-hybrid model in the UC framework. The protocol generates ℓ $\langle\cdot\rangle$-shared random values and has communication complexity $\mathcal{BC}\big(|\mathcal{X}|(\ell + s)\kappa, |\mathcal{X}|\big)$.*

Protocol for Reconstructing $\langle\cdot\rangle$-shared Value. Protocol $\Pi_{\text{REC}\langle\cdot\rangle}$ takes as input an $\langle\cdot\rangle$-sharing, say $\langle s \rangle_{\mathcal{X}}$ and either allows the honest parties in $\mathcal{X}$ to robustly reconstruct s or ensures that the honest parties in $\mathcal{X}$ can (locally) identify at least one corrupted party in $\mathcal{X}$. The protocol is based on the following standard idea: let $\langle s \rangle_i = (s_i, u_i, v_i, \{\mathbf{C}_{s_j,u_j,v_j}\}_{P_j \in \mathcal{X}})$ be the information available to party $P_i \in \mathcal{X}$ corresponding to $\langle s \rangle_{\mathcal{X}}$. Then each P_i broadcasts s_i, u_i, v_i to the parties in $\mathcal{X}$ via $\mathcal{F}_{\mathrm{BC}}$. Let each party P_i receive $\bar{s}_j, \bar{u}_j, \bar{v}_j$ from P_j. P_i then verifies if $\mathbf{C}_{s_j,u_j,v_j} = \mathsf{Comm}_{\mathsf{ck}}(\bar{s}_j; \bar{u}_j, \bar{v}_j)$. If the verification fails, P_i *identifies* P_j to be corrupted and outputs $(\mathsf{Failure}, i, j)$; otherwise P_i sums up all the shares to obtain s and outputs $(\mathsf{Success}, i, s)$. In the rest of the description, we will say that *the parties in $\mathcal{X}$ participate in $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ with $\langle s \rangle_{\mathcal{X}}$ and each $P_i \in \mathcal{X}$ outputs either* $(\mathsf{Success}, i, s)$ *or* $(\mathsf{Failure}, i, j)$ to mean the above. The protocol has communication complexity $\mathcal{BC}\big(|\mathcal{X}|\kappa, |\mathcal{X}|\big)$.

Beaver's Multiplication Protocol. Protocol $\Pi_{\mathrm{BEA}}(\langle x \rangle_{\mathcal{X}}, \langle y \rangle_{\mathcal{X}}, \langle a \rangle_{\mathcal{X}}, \langle b \rangle_{\mathcal{X}}, \langle c \rangle_{\mathcal{X}})$ is a standard protocol for securely computing $\langle x \cdot y \rangle_{\mathcal{X}}$ from $\langle x \rangle_{\mathcal{X}}$ and $\langle y \rangle_{\mathcal{X}}$, at the cost of two public reconstruction. The protocol assumes that the parties in $\mathcal{X} \subset \mathcal{P}$ have access to an $\langle \cdot \rangle_{\mathcal{X}}$-shared random multiplication triple (a, b, c) unknown to the adversary, with $c = a \cdot b$. The protocol is based on the principle that $x \cdot y = (x - a + a) \cdot (y - b + b) = de + db + ae + c$, where $d = (x - a)$ and $e = (y - b)$. Hence if the parties in $\mathcal{X}$ reconstruct d and e, then they can locally compute $\langle x \cdot y \rangle_{\mathcal{X}} = de + d \cdot \langle b \rangle_{\mathcal{X}} + e \cdot \langle a \rangle_{\mathcal{X}} + \langle c \rangle_{\mathcal{X}}$. The security of x and y follows even after the reconstruction of d and e, as x and y are masked by random and private a and b respectively. To reconstruct d and e, the parties in $\mathcal{X}$ first locally compute $\langle d \rangle_{\mathcal{X}} = \langle x - a \rangle_{\mathcal{X}}$ and $\langle e \rangle_{\mathcal{X}} = \langle y - b \rangle_{\mathcal{X}}$, followed by invoking $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ with inputs $\langle d \rangle_{\mathcal{X}}$ and $\langle e \rangle_{\mathcal{X}}$. Depending on whether the instances of $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ are successful or not, an honest party in $\mathcal{X}$ may output $(\mathsf{Success}, i, \langle x \cdot y \rangle_i)$ or $(\mathsf{Failure}, i, j)$. In the rest of the description, we will say that *the parties in $\mathcal{X}$ participate in $\Pi_{\mathrm{BEA}}(\langle x \rangle_{\mathcal{X}}, \langle y \rangle_{\mathcal{X}}, \langle a \rangle_{\mathcal{X}}, \langle b \rangle_{\mathcal{X}}, \langle c \rangle_{\mathcal{X}})$ and each P_i output either* $(\mathsf{Success}, i, \langle x \cdot y \rangle_i)$ *or* $(\mathsf{Failure}, i, j)$ to mean the above. The protocol has communication complexity $\mathcal{BC}\big(|\mathcal{X}|\kappa, |\mathcal{X}|\big)$.

E.1 The Preparation Stage of Protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$

We are now ready to discuss the preparation stage of our protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$. We pursue the same outline as followed by the preparation stage of the MPC protocol of [12] and describe the same briefly below. This is followed by the required adaptations in our context. The preparation stage of [12] provides security with abort. Namely the protocol generates the required $\langle \cdot \rangle$-shared triplets if all the parties behave honestly; otherwise if the honest parties identify any wrong-doing then they simply abort.

- **Triple Generation:** The involved parties generate many random $\langle \cdot \rangle$-shared triplets by executing many instances of Π_{TRIPLE}, assuming no active corruptions.
- **Verification of the Triples via Cut-and-choose:** A random fraction of the triplets are verified via cut-and-choose to detect any cheating attempts. Specifically, a random subset of generated triplets are selected and the parties are asked to disclose the randomness that they used in the instances of Π_{TRIPLE} for generating the selected triplets. If any cheating is detected then the involved parties abort, otherwise they proceed to the next step. If the test passes then with high probability it is ensured that the majority of the remaining untested triplets are "good" in the sense that they are honestly generated.
- **Proof of Knowledge:** The goal of this test is to ensure that for each remaining triplet, every party has the knowledge of their shares, thus ensuring independence required for UC security. More specifically, during the generation of an untested triplet, a corrupted party P_i could broadcast an arbitrary $\mathbf{C}_{a_i,\star,\star}, \mathbf{C}_{b_i,\star,\star}$ or $\mathbf{C}_{c_i,\star,\star}$, being oblivious to a_i, b_i and c_i. This is prevented by the following steps: First parties generate random $\langle \cdot \rangle$-shared values (by calling $\mathcal{F}_{\mathrm{GENRAND}\langle\cdot\rangle}$) and then they open the difference of the triplets and those random shared values via protocol $\Pi_{\mathrm{REC}\langle\cdot\rangle}$. Opening these differences is indeed a very simple proof of knowledge (see [12]). A cheating is detected if some of the opening fail. In that case the involved parties abort, otherwise they proceed to the next step.

– **Verification of the Triplets via Sacrificing Trick:** At this stage, the remaining triplets are verified for correctness via the well-known "sacrificing" trick [13]. Namely for every pair of remaining shared triplets (a, b, c) and (x, y, z), the parties generate a random r and recompute an $\langle\cdot\rangle$-sharing of $a \cdot b$, by assuming rx, ry, r^2z as a multiplication triplet; protocol Π_{BEA} is used for the same. Ideally if (a, b, c) and (x, y, z) are multiplication triplets, then the difference of the sharing of c and the recomputed ab should be a sharing of zero, which is verified by the parties publicly (using protocol $\Pi_{\text{REC}\langle\cdot\rangle}$). If any cheating is detected then the parties abort, else they proceed to the next step after discarding (x, y, z), whose security is sacrificed during the verification of (a, b, c). It follows that if the test passes then except with probability $1/p$ over the choice of r, the triplet (a, b, c) is indeed a correct multiplication triplet (see [12] for the details).
– **Privacy Amplification:** At this stage, the parties jointly perform privacy amplification and "distill" $\mathsf{C_M} + \mathsf{C_R}$ fully random private triplets from a set of $\mathcal{O}((\mathsf{C_M} + \mathsf{C_R}) + X)$ triplets, where X of them might not be private[10]; recall that $\mathsf{C_M}$ and $\mathsf{C_R}$ are the number of multiplication and random gates respectively in the circuit C. For this, $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ along with Π_{BEA} is used. If any cheating is detected during Π_{BEA}, then the parties abort.

In our context, it is not enough to abort when a wrong-doing is detected. If some party $P_i \in \mathcal{X}$ identifies any party $P_j \in \mathcal{X}$ cheating in any of the steps for preparation stage, P_i alarms the parties in $\mathcal{P}$ by raising a complaint against P_j. This allows the parties in $\mathcal{P}$ to localize the fault to a pair of parties (P_i, P_j). To simplify the fault-localization, we set a designated party $P_{\mathsf{Ref}} \in \mathcal{X}$ with the smallest index Ref as the *referee* to locally identify any fault and report the same to the parties in $\mathcal{P}$. The fault localization step in each stage of the preparation stage is emphasized below.

– **Fault Localization During the Verification of the Triples via Cut-and-choose:** The parties in $\mathcal{X}$ first run the steps for the cut-and-choose triple-verification as in [12]. If any party P_i locally identifies any fault then it raises an alarm for the parties in $\mathcal{P}$. On receiving the alarm, every party in $\mathcal{X}$ broadcasts (to the parties in $\mathcal{X}$) their entire view (including the randomness used) in the generation of the triplets under testing. The referee P_{Ref} then "recomputes" every message a party $P_i \in \mathcal{X}$ should send to every other party $P_j \in \mathcal{X}$ and compares them with what P_i claims to send and what P_j claimed to receive. In case there is any mis-match, then P_{Ref} raises a complaint against both P_i and P_j among $\mathcal{P}$ and urges P_i and P_j to respond. Now depending upon the response, the parties can localize the fault to either (P_{Ref}, P_i) or (P_{Ref}, P_j) or (P_i, P_j). The important observation is that fault will never be localized to a pair of honest parties from $\mathcal{X}$. This is because the property of Π_{MULT} ensures that if both the participating parties are honest then they never conflict with each other. A located pair will contain at least one corrupted party.
– **Fault Localization in Proof of Knowledge:** The parties in $\mathcal{X}$ execute the same steps as in [12] for proving the knowledge of their shares. If any party $P_i \in \mathcal{X}$ locally identifies any fault during the instances of $\Pi_{\text{REC}\langle\cdot\rangle}$ (used to open the differences of triplets and random shared values), then P_i raises an alarm among the set $\mathcal{P}$, while the referee P_{Ref} is assigned the task of publicly reporting the identity of the party P_j it has caught cheating. The fault is then localized to (P_j, P_{Ref}). If an honest P_i raises an alarm, but a corrupted P_{Ref} does not identify any cheater, then the fault is localized to (P_i, P_{Ref}). It is easy to note that a located pair will contain at least one corrupted party.
– **Fault Localization During the Verification of the Triplets via Sacrificing Trick:** Here the parties in $\mathcal{X}$ first apply the sacrificing trick on each pair of remaining triplets. Now there are *three* situations under which a party $P_i \in \mathcal{X}$ can detect a fault. **(a)** The instances of Π_{BEA} is unsuccessful. In this case, the

[10] For the specific instantiation of Π_{MULT} based on Paillier encryption, this is indeed the case if one of the participating parties in Π_{MULT} is corrupted; see [12] for the details.

parties in $\mathcal{P}$ localize the fault in the same way as in the previous step. Namely P_i raises an alarm while P_{Ref} is asked to identify the cheating party. **(b)** The instances of $\Pi_{\mathrm{REC}\langle\cdot\rangle}$ to open the difference of ab and c fails; the fault-localization in this case is also the same as in the previous step. **(c)** The difference of ab and c is non-zero. Clearly in this case, at least one of the involved triplet (in the pair) is not generated correctly and so the parties in $\mathcal{P}$ perform the fault-localization in the same way as in the cut-and-choose step. Namely, all the parties in $\mathcal{X}$ publicly open (to the parties in $\mathcal{X}$) their entire view produced during the generation of the two triplets and P_{Ref} is then asked to find a pair of "conflicting" parties.

– **Fault Localization in Privacy Amplification:** The parties in $\mathcal{X}$ execute the steps for privacy amplification [12]. If any cheating is detected by a party $P_i \in \mathcal{X}$ during the involved instances of Π_{BEA}, then the parties in $\mathcal{P}$ perform the fault localization in the same way as it is done during for a failed instance of Π_{BEA} in the previous stage.

The protocol steps for the preparation stage are given in Figure 17, where we give the formal steps for the fault localization with respect to only the first two phases; the formal steps for the fault localization for the remaining phases is not provided to avoid repetition.

In the protocol, B and λ are two parameters. In [12], it was shown that their preparation stage provides a statistical security of $2^{-B\log_2(1+\lambda)}$. They set B and λ as $B = 3.6s$ and $\lambda = 1/4$ to achieve a statistical security of 2^{-s}. Since our preparation stage is almost the same as that of [12] bar the fault-localization steps (which does not affect the statistical security at all), it follows easily via [12] that our preparation stage also provides a statistical security of $2^{-B\log_2(1+\lambda)}$. Intuitively this is due to the following reason: define a triplet to be a *good* one if the adversary could open it correctly during the **Cut-and-choose** step and make an honest party accept (this implies that such a triplet is generated honestly), otherwise call the triplet a *bad* triplet (i.e. such triplets are not generated honestly and so adversary may know some information about honest partys' shares for such triplets). Then it follows from [12] that if the protocol reaches the **Privacy Amplification** phase, then the probability that the triplets considered during this phase has more than B bad (and hence non-private) triplets is at most $(1+\lambda)^{-B}$. As a result, adversary may know at most B points on the polynomials $F(\cdot)$ and $G(\cdot)$ of degree at most d, implying $\mathsf{C_M} + \mathsf{C_R}$ degree of freedom in the view of the adversary. Note that as suggested in [12], instead of creating "big" polynomials $F(\cdot)$ and $G(\cdot)$ of huge degrees, we can partition the remaining triplets in $\mathcal{M}$ into batches of smaller size and accordingly use many polynomials of small degree, without affecting the security properties; we prefer to present the **Privacy Amplification** phase the way presented in [12].

Preparation Stage of $\Pi_{\mathsf{C}}^{\mathbf{NR}}$

The public input to the protocol is a set of parties $\mathcal{X} \subset \mathcal{P}$ containing at least one honest party and a referee $P_{\mathsf{Ref}} \in \mathcal{X}$, with the smallest index Ref. For the session id sid, every party $P_i \in \mathcal{P}$ participates with (sid, i) and does the following:

Triple-generation Assuming No Active Corruption — If $P_i \in \mathcal{X}$ then participate in the protocol Π_{TRIPLE} $(1+\lambda)(4(\mathsf{C_M}+\mathsf{C_R})+4B-2)$ times to generate a set $\mathcal{M}$ of $(1+\lambda)(4(\mathsf{C_M}+\mathsf{C_R})+4B-2)$ $\langle\cdot\rangle$-shared triplets.

Testing the Triplets via Cut-and-Choose — If $P_i \in \mathcal{X}$ then do the following:
- Call $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ to sample a random string str that determines a subset $\mathcal{T} \subset \mathcal{M}$ of size $\lambda(4(\mathsf{C_M}+\mathsf{C_R})+4B-2)$. Set $\mathcal{M} = \mathcal{M} \setminus \mathcal{T}$. Let $\mathsf{View}_i^{\mathcal{T}}$ denote the randomness used by P_i and the messages received from the other parties in $\mathcal{X}$, during the instances of Π_{TRIPLE} used for generating the triplets in $\mathcal{T}$. Reveal $\mathsf{View}_i^{\mathcal{T}}$ to the parties in $\mathcal{X}$ by calling $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, \mathsf{View}_i^{\mathcal{T}}, \mathcal{X})$.
- Corresponding to each $P_j \in \mathcal{X}$, receive $(\mathsf{sid}, i, j, \mathsf{View}_j^{\mathcal{T}})$ from $\mathcal{F}_{\text{BC}}$. Using $\{\mathsf{View}_j^{\mathcal{T}}\}_{P_j \in \mathcal{X}}$, reproduce every message that should have been sent by every sender $P_a \in \mathcal{X}$ to every receiver $P_b \in \mathcal{X}$ during the generation of the triplets in $\mathcal{T}$, and compare it with the corresponding value that the recipient P_b claims to have received. If any conflict is detected, then do the following for the smallest indexed conflicting parties P_a, P_b:
  - If $P_i \neq P_{\mathsf{Ref}}$, then call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, \mathtt{Err}, \mathcal{P})$ to indicate to the parties in $\mathcal{P}$ that a conflict has been detected.
  - Else call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, \mathsf{Ref}, \mathtt{Err}, P_a, P_b, l, x, \overline{x}, \mathcal{P})$ to indicate that referee P_i identified $P_a, P_b \in \mathcal{X}$ the least indexed conflicting parties and a message with index l where P_a should have sent x but P_b claimed to receive $\overline{x} \neq x$.
- If the message $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, P_a, P_b, l, x, \overline{x})$ is received from $\mathcal{F}_{\text{BC}}$ and if $P_a = P_i$ or $P_b = P_i$, then call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, \mathsf{Agree}, P_{\mathsf{Ref}}, \mathcal{P})$ to indicate that you agree with P_{Ref}, else call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, \mathsf{Disagree}, P_{\mathsf{Ref}}, \mathcal{P})$.

Fault Localization —
- If the message $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, P_a, P_b, l, x, \overline{x})$ is received from $\mathcal{F}_{\text{BC}}$ and subsequently **(a)** if $(\mathsf{sid}, i, a, \mathsf{Disagree}, P_{\mathsf{Ref}})$ is received from $\mathcal{F}_{\text{BC}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_a)$ and halt **(b)** if $(\mathsf{sid}, i, b, \mathsf{Disagree}, P_{\mathsf{Ref}})$ is received from $\mathcal{F}_{\text{BC}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_b)$ and halt. Else output $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ and halt.
- If no message of the form $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, \star, \star, \star, \star, \star)$ is received from $\mathcal{F}_{\text{BC}}$, but corresponding to some $P_j \in \mathcal{X}$ the message $(\mathsf{sid}, i, j, \mathtt{Err})$ is received from $\mathcal{F}_{\text{BC}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_j)$ and halt.

Proof of Knowledge — If $P_i \in \mathcal{X}$ then do the following for every (untested) triplet $(\langle a\rangle_{\mathcal{X}}, \langle b\rangle_{\mathcal{X}}, \langle c\rangle_{\mathcal{X}})$ in $\mathcal{M}$: Sample three random $\langle\cdot\rangle$-shared values $\langle r\rangle_{\mathcal{X}}, \langle s\rangle_{\mathcal{X}}, \langle u\rangle_{\mathcal{X}}$ by invoking $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$. Participate in instances of $\Pi_{\text{REC}\langle\cdot\rangle}$ with $\langle r-a\rangle_{\mathcal{X}}, \langle s-b\rangle_{\mathcal{X}}$ and $\langle u-c\rangle_{\mathcal{X}}$. If $(\mathsf{sid}, \mathsf{Failure}, i, j)$ is the output in any of the instances of $\Pi_{\text{REC}\langle\cdot\rangle}$, then do the following:
- If $P_i \neq P_{\mathsf{Ref}}$ then call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, i, j, \mathtt{Err}, \mathcal{P})$ to indicate that a cheating has been detected.
- Else if $P_i = P_{\mathsf{Ref}}$ then call $\mathcal{F}_{\text{BC}}$ with $(\mathsf{sid}, \mathsf{Ref}, \mathtt{Err}, j, \mathcal{P})$ to indicate P_j is identified as a cheater; if there are several such P_js then select the one with the minimum index j.

Fault Localization —If a message $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, j, \mathcal{P})$ is received from $\mathcal{F}_{\text{BC}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_j)$ and halt. Else if a message $(\mathsf{sid}, i, j, \mathtt{Err})$ is received from $\mathcal{F}_{\text{BC}}$, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_i, P_j)$ and halt.

Verification Via sacrificing Trick — If $P_i \in \mathcal{X}$ then do the following for every pair of triplets $(\langle a\rangle_{\mathcal{X}}, \langle b\rangle_{\mathcal{X}}, \langle c\rangle_{\mathcal{X}})$ and $(\langle x\rangle_{\mathcal{X}}, \langle y\rangle_{\mathcal{X}}, \langle z\rangle_{\mathcal{X}})$ in $\mathcal{M}$: Call $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ and sample a random[a] r. Participate in $\Pi_{\text{BEA}}(\langle a\rangle_{\mathcal{X}}, \langle b\rangle_{\mathcal{X}}, \langle rx\rangle_{\mathcal{X}}, \langle ry\rangle_{\mathcal{X}}, \langle r^2 z\rangle_{\mathcal{X}})$ for computing $\langle \overline{c}\rangle_{\mathcal{X}}$ followed by participation in $\Pi_{\text{REC}\langle\cdot\rangle}$ with $\langle \overline{c}-c\rangle_{\mathcal{X}}$. If no cheating has been identified during $\Pi_{\text{BEA}}, \Pi_{\text{REC}\langle\cdot\rangle}$ and if $\overline{c}-c=0$, then store $(\langle a\rangle_{\mathcal{X}}, \langle b\rangle_{\mathcal{X}}, \langle c\rangle_{\mathcal{X}})$ for future use and drop $(\langle x\rangle_{\mathcal{X}}, \langle y\rangle_{\mathcal{X}}, \langle z\rangle_{\mathcal{X}})$ from $\mathcal{M}$. Else proceed to the fault-localization step.

Fault Localization — If the parties in $\mathcal{X}$ have raised a complaint due to the failure of Π_{BEA} or $\Pi_{\text{REC}\langle\cdot\rangle}$, then localize the fault in the same way as in the case of fault-localization for the **Proof of Knowledge** step. Else localize the fault in the same way as in the **Cut-and-Choose** step by asking the parties in $\mathcal{X}$ to open their entire view of the disputed triplet.

Privacy Amplification — The parties in $\mathcal{X}$ are now left with $2(\mathsf{C_M}+\mathsf{C_R})+2B-1$ triplets $\{(\langle a^k\rangle_{\mathcal{X}}, \langle b^k\rangle_{\mathcal{X}}, \langle c^k\rangle_{\mathcal{X}})\}_{k=1,\ldots,2(\mathsf{C_M}+\mathsf{C_R})+2B-1}$ in $\mathcal{M}$. Let $d = (\mathsf{C_M}+\mathsf{C_R})+B-1$. If $P_i \in \mathcal{X}$ then do the following:
- Invoke $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ $2(d+1)$ times to generate $\langle f^{(1)}\rangle_{\mathcal{X}}, \ldots, \langle f^{(d+1)}\rangle_{\mathcal{X}}$ and $\langle g^{(1)}\rangle_{\mathcal{X}}, \ldots, \langle g^{(d+1)}\rangle_{\mathcal{X}}$.
- Let $F(\cdot)$ and $G(\cdot)$ be the polynomials of degree at most d such that $F(\alpha_k) = f^{(k)}$ and $G(\alpha_k) = g^{(k)}$ for $k = 1, \ldots, d+1$. Locally compute $\langle F(\alpha_{d+2})\rangle_{\mathcal{X}}, \ldots, \langle F(\alpha_{2d+1})\rangle_{\mathcal{X}}$ and $\langle G(\alpha_{d+2})\rangle_{\mathcal{X}}, \ldots, \langle G(\alpha_{2d+1})\rangle_{\mathcal{X}}$. For $k = 1, \ldots, 2d+1$, participate in $\Pi_{\text{BEA}}(\langle F(\alpha_k)\rangle_{\mathcal{X}}, \langle G(\alpha_k)\rangle_{\mathcal{X}}, \langle a^{(k)}\rangle_{\mathcal{X}}, \langle b^{(k)}\rangle_{\mathcal{X}}, \langle c^{(k)}\rangle_{\mathcal{X}})$ for computing $\langle h^{(k)}\rangle_{\mathcal{X}} = \langle F(\alpha_k)\cdot G(\alpha_k)\rangle_{\mathcal{X}}$.
- If any cheating is identified during Π_{BEA}, then proceed to the fault localization step. Else let $H(\cdot)$ be the polynomial of degree at most $2d$ such that $H(\alpha_i) = h^{(i)}$ for $i = 1, \ldots, 2d+1$. Then output $(\mathsf{sid}, i, \mathsf{Success}, \{(\langle \mathbf{a}^{(k)}\rangle_{\mathcal{X}}, \langle \mathbf{b}^{(k)}\rangle_{\mathcal{X}}, \langle \mathbf{c}^{(k)}\rangle_{\mathcal{X}})\}_{k=1,\ldots,\mathsf{C_M}+\mathsf{C_R}})$ and halt, where $\mathbf{a}^{(k)} = F(-\alpha_k)$, $\mathbf{b}^{(k)} = G(-\alpha_k)$ and $\mathbf{c}^{(k)} = H(-\alpha_k)$.

Fault Localization — If any complaint is raised due to the failure of Π_{BEA}, then localize the fault as in the **Proof of Knowledge** step. Else every $P_i \in \mathcal{P} \setminus \mathcal{X}$ output $(\mathsf{sid}, i, \mathsf{Success})$ and halt.

[a] It is enough to sample a *single* r for all the pairs of available triplets.

Fig. 17. Generating $\mathsf{C_M}+\mathsf{C_R}$ $\langle\cdot\rangle$-shared Multiplication Triples with Statistical Security $2^{-B\log_2(1+\lambda)}$.

E.2 Protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$

In this section the protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$ is presented in Figure 18, where during the circuit-evaluation stage, we follow the idea outlined earlier in section 3.2. Note that during the circuit evaluation, an instance of Π_{BEA} may fail, in which case the parties in $\mathcal{P}$ localize the fault via the referee P_{Ref} in the same way as it was done in the preparation stage.

Protocol $\Pi_{\mathsf{C}}^{\mathrm{NR}}$

The public input to the protocol is a set of parties $\mathcal{X} \subset \mathcal{P}$ containing at least one honest party and an arithmetic circuit C over $\mathbb{F}_p$ consisting of in input gates, out output gates, $\mathsf{C_M}$ multiplication gates and $\mathsf{C_R}$ random gates. In addition, $\langle x_1 \rangle_{\mathcal{X}}, \ldots, \langle x_{\mathsf{in}} \rangle_{\mathcal{X}}$ are the $\langle\cdot\rangle$-shared inputs for C. Let $P_{\mathsf{Ref}} \in \mathcal{X}$ be the party with the smallest index Ref who is set as the referee to localize any fault occurred during the protocol.

For the session id sid, party $P_i \in \mathcal{P}$ participates with (sid, i) and does the following:

Preparation Stage: execute the steps of Figure 17.

Computation Stage: If $(\mathsf{sid}, i, \mathsf{Success}, \{(\langle \mathbf{a}^{(k)} \rangle_{\mathcal{X}}, \langle \mathbf{b}^{(k)} \rangle_{\mathcal{X}}, \langle \mathbf{c}^{(k)} \rangle_{\mathcal{X}})\}_{k=1,\ldots,\mathsf{C_M}+\mathsf{C_R}})$ or $(\mathsf{sid}, i, \mathsf{Success})$ is obtained at the end of preparation stage, then do the following:

- **$\langle\cdot\rangle$-shared Evaluation of the Circuit** C — If $P_i \in \mathcal{X}$ then do the following for every gate in the circuit C:
  - **Input Gate**: For $l = 1, \ldots, \mathsf{in}$, associate $\langle x_l \rangle_{\mathcal{X}}$ with the corresponding input gate of C.
  - **Random Gate**: For the k^{th} random gate in C where $k \in \{1, \ldots, \mathsf{C_R}\}$, associate $\langle \mathbf{a}^{(k)} \rangle_{\mathcal{X}}$ as the output of the random gate.
  - **Addition Gate**: If $\langle x \rangle_{\mathcal{X}}$ and $\langle y \rangle_{\mathcal{X}}$ are the $\langle\cdot\rangle$-shared inputs of the gate, then locally compute $\langle x+y \rangle_{\mathcal{X}} = \langle x \rangle_{\mathcal{X}} + \langle y \rangle_{\mathcal{X}}$ and associate it as the output of the addition gate.
  - **Multiplication Gate**: For the k^{th} multiplication gate in C with the $\langle\cdot\rangle$-shared inputs $\langle x \rangle_{\mathcal{X}}$ and $\langle y \rangle_{\mathcal{X}}$ where $k \in \{1, \ldots, \mathsf{C_M}\}$, associate the triplet $(\langle \mathbf{a}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}}, \langle \mathbf{b}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}}, \langle \mathbf{c}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}})$. Participate in $\Pi_{\mathrm{BEA}}(\langle x \rangle_{\mathcal{X}}, \langle y \rangle_{\mathcal{X}}, \langle \mathbf{a}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}}, \langle \mathbf{b}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}}, \langle \mathbf{c}^{(\mathsf{C_R}+k)} \rangle_{\mathcal{X}})$ to compute $\langle x \cdot y \rangle_{\mathcal{X}}$. If $(\mathsf{sid}, i, \mathsf{Failure}, j)$ with $P_j \in \mathcal{X}$ is obtained during the instance of Π_{BEA} then do the following:
    - ∗ If $P_i \neq P_{\mathsf{Ref}}$ then call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, i, \mathtt{Err}, \mathcal{P})$ to indicate that a cheating has been detected while executing Π_{BEA}.
    - ∗ Else if $P_i = P_{\mathsf{Ref}}$ then call $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, \mathsf{Ref}, \mathtt{Err}, j, \mathcal{P})$ to indicate that P_j is identified as a cheater while executing Π_{BEA}; if there are several such P_js then select the one with the minimum index j.
- **Fault Localization** —
  - If there exists a multiplication gate in C corresponding to which a message $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, j)$ is received on the behalf of P_{Ref} from $\mathcal{F}_{\mathrm{BC}}$ then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_j)$ and halt.
  - Else if there exists a multiplication gate in C corresponding to which a message $(\mathsf{sid}, i, j, \mathtt{Err})$ is received from $\mathcal{F}_{\mathrm{BC}}$ on the behalf of $P_j \in \mathcal{X}$, but no message of the form $(\mathsf{sid}, i, \mathsf{Ref}, \mathtt{Err}, \star, \star)$ is received from $\mathcal{F}_{\mathrm{BC}}$ on the behalf of P_{Ref}, then output $(\mathsf{sid}, i, \mathsf{Failure}, P_{\mathsf{Ref}}, P_j)$ and halt.
  - Else if $P_i \in \mathcal{P} \setminus \mathcal{X}$ then output $(\mathsf{sid}, i, \mathsf{Success})$ and halt; otherwise output $(\mathsf{sid}, i, \mathsf{Success}, \langle y_1 \rangle_{\mathcal{X}}, \ldots, \langle y_{\mathsf{out}} \rangle_{\mathcal{X}})$ and halt, where $\langle y_1 \rangle_{\mathcal{X}}, \ldots, \langle y_{\mathsf{out}} \rangle_{\mathcal{X}}$ are the $\langle\cdot\rangle$-shared outputs associated with the output gates of C.

Fig. 18. Protocol for Secure $\langle\cdot\rangle$-shared Evaluation of a Given Circuit C with Statistical Security $1 - 2^{-B\log_2(1+\lambda)}$.

The correctness of the protocol follows via the binding property of the commitment and the detailed informal discussion above, while we appeal to [12] for the proof of privacy in UC secure framework. We now prove Lemma 1 (the lemma statement is available in Section 3), by setting $\lambda = 1/4$ and $B = 3.6s$ as done in [12], so that the protocol provides a statistical security of 2^{-s}.

Proof of Lemma 1: We prove the communication complexity of the preparation stage, with the observation that $\mathsf{C_M} + \mathsf{C_R} = \mathcal{O}(|\mathsf{C}|)$. During the **Triple-generation** phase, $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ instances of Π_{TRIPLE} are executed by the parties in $\mathcal{X}$, thus requiring communication complexity of $\mathcal{O}(|\mathcal{X}|^2(|\mathsf{C}| + B)\kappa)$ and $\mathcal{BC}\big(|\mathcal{X}|(|\mathsf{C}| + B)\kappa, |\mathcal{X}|\big)$.

During the **Cut-and-Choose** phase, $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ calls to $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ are made for generating $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ random $\langle\cdot\rangle$-shared commitments with statistical security $2^{-B\log_2(1+\lambda)}$, incurring communication complexity of $\mathcal{BC}\big(|\mathcal{X}|(|\mathsf{C}| + B)\kappa, |\mathcal{X}|\big)$. In addition, the parties in $\mathcal{X}$ need to broadcast among themselves their entire view of Π_{TRIPLE} with respect to $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ triplets. This incurs a communication complexity of $\mathcal{BC}\big(|\mathcal{X}|^2(|\mathsf{C}| + B)\kappa, |\mathcal{X}|\big)$. During the fault-localization step, the parties in $\mathcal{X}$ need to broadcast $\mathcal{O}(\kappa)$ bits to the parties in $\mathcal{P}$, thus requiring communication complexity of $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$.

During the **Proof of Knowledge** phase, $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ calls to $\mathcal{F}_{\text{GENRAND}\langle\cdot\rangle}$ are made and $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ instances of $\Pi_{\text{REC}\langle\cdot\rangle}$ are executed by the parties in $\mathcal{X}$, thus requiring a communication complexity of $\mathcal{BC}\big(|\mathcal{X}|(|\mathsf{C}|+B)\kappa, |\mathcal{X}|\big)$. In addition, during the fault-localization step, the parties in $\mathcal{X}$ need to broadcast $\mathcal{O}(\kappa)$ bits to the parties in $\mathcal{P}$, thus requiring communication complexity of $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$.

During the **Correctness** phase, $\mathcal{O}(\mathsf{C_M} + \mathsf{C_R} + B)$ instances of Π_{BEA} and $\Pi_{\text{REC}\langle\cdot\rangle}$ are executed by the parties in $\mathcal{X}$. In addition, the parties in $\mathcal{X}$ may need to publicly open among themselves the entire view of Π_{TRIPLE} with respect to a disputed pair of triplet. Thus this phase has communication complexity of $\mathcal{BC}\big(|\mathcal{X}|(|\mathsf{C}| + B)\kappa, |\mathcal{X}|\big)$, with an additional communication complexity of $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$ for the fault-localization step. It follows easily that the **Privacy Amplification** phase as well as the circuit evaluation stage has communication complexity of $\mathcal{BC}\big(|\mathcal{X}|(|\mathsf{C}| + B)\kappa, |\mathcal{X}|\big)$ for executing the steps within $\mathcal{X}$ and has communication complexity of $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$ for any possible fault-localization.

During the computation stage, $\mathsf{C_M}$ instances of Π_{BEA} are executed and fault-localization is done at most once. It thus follows that setting $B = 3.6s$, the protocol has communication complexity $\mathcal{O}(|\mathcal{X}|^2(|\mathsf{C}| + s)\kappa)$, $\mathcal{BC}\big(|\mathcal{X}|^2(|\mathsf{C}| + s)\kappa, |\mathcal{X}|\big)$ and $\mathcal{BC}\big(|\mathcal{X}|\kappa, n\big)$. □

F Proof of Theorem 1

Proof. **Communication Complexity.** We start with the communication complexity analysis of the protocol. We analyze each phase of the protocol separately:

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 Lemma 2), this incurs 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 $\mathcal{O}(n \cdot \mathfrak{c} \cdot \mathsf{in}_l \cdot \kappa)$ and $\mathcal{BC}\big(n \cdot \mathfrak{c} \cdot \mathsf{in}_l \cdot \kappa, n\big)$; this follows from Lemma 5 by substituting $|\mathcal{X}| = \mathfrak{c}$.
   - Since the size of ckt_l is at most $\frac{|\mathsf{ckt}|}{L}$, evaluating the same via protocol $\Pi^{\text{NR}}_{\mathsf{ckt}_l}$ 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 1 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 3 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 $\mathcal{BC}\big(\mathfrak{c}^2(\frac{|\mathsf{ckt}|}{L}+s)\kappa, \mathfrak{c}\big)$. Now 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 $\mathcal{BC}\big((\mathfrak{c}^2 \cdot (\frac{|\mathsf{ckt}|}{L} + s))\kappa, \mathfrak{c}\big)$.

Now the total number of sub-circuit evaluations is at most $L + t$, with L successful evaluations and at most t failed evaluations. Now substituting $L = t$, we get the communication complexity $\mathcal{O}((|\mathsf{ckt}| \cdot (\frac{n \cdot t \cdot \mathfrak{c}}{d} + \mathfrak{c}^2) + n^2 t + \mathfrak{c}^2 s \cdot t)\kappa)$, $\mathcal{BC}\big((|\mathsf{ckt}| \cdot \frac{n \cdot t \cdot \mathfrak{c}}{d} + n^2 t)\kappa, n\big)$ and $\mathcal{BC}\big((|\mathsf{ckt}| \cdot \mathfrak{c}^2 + \mathfrak{c}^2 \cdot s \cdot t)\kappa, \mathfrak{c}\big)$ for the overall evaluations of all the sub-circuits.

4. **Output Rerandomization and Reconstruction:** It is easy to see that the cost of this step is $\mathcal{O}(\mathsf{Poly}(n, \kappa))$ bits.

Concentrating on the circuit-dependent complexity of the whole protocol, the complexity 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 of protocol Π_f 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 $\mathcal{BC}\big(|\mathsf{ckt}|\kappa^3, \kappa\big)$.

Security. We next prove the security by designing a simulator for the protocol Π_f. Let $T \subset \mathcal{P}$ be the set of parties under the control of $\mathcal{A}$ during the protocol Π_f; we present a simulator $\mathcal{S}_f$ (interacting with the functionality $\mathcal{F}_f$) for $\mathcal{A}$ in Figure 19. The high level idea for the simulator is the following: the simulator takes the input $\{x^{(i)}\}_{P_i \in T}$ and interacts with $\mathcal{F}_f$ to obtain the function output y. The simulator then invokes $\mathcal{A}$ with the inputs $\{x^{(i)}\}_{P_i \in T}$ and simulates each message that $\mathcal{A}$ would have received in the protocol Π_f from the honest parties and from the functionalities called therein, step by step. Notice that the simulator $\mathcal{S}_f$ also needs to simulate the protocol steps of the honest parties for the sub-protocols $\Pi_{[\cdot] \to \langle \cdot \rangle}$, $\Pi_{\langle \cdot \rangle \to [\cdot]}$, $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$ and $\Pi_{\textsc{RandZero}[\cdot]}$. Specifying the simulator steps for these subprotocols would make the description of $\mathcal{S}_f$ complicated. So for the ease of presentation, we define three sub-simulators $\mathcal{S}_{[\cdot] \to \langle \cdot \rangle}$ (Fig. 20), $\mathcal{S}_{\langle \cdot \rangle \to [\cdot]}$ (Fig. 21), and $\mathcal{S}_{\textsc{RandZero}[\cdot]}$ (Fig. 22) which are invoked by $\mathcal{S}_f$ for simulating the steps of the honest parties for the instances of $\Pi_{[\cdot] \to \langle \cdot \rangle}$, $\Pi_{\langle \cdot \rangle \to [\cdot]}$ and $\Pi_{\textsc{RandZero}[\cdot]}$ respectively; technically, the steps specified for $\mathcal{S}_{[\cdot] \to \langle \cdot \rangle}$, $\mathcal{S}_{\langle \cdot \rangle \to [\cdot]}$ and $\mathcal{S}_{\textsc{RandZero}[\cdot]}$ are actually done by the main simulator $\mathcal{S}_f$. While invoking these "sub-simulators", $\mathcal{S}_f$ will provide its entire internal state to them and the sub-simulators then return back their internal state (after the required simulation) to the main simulator. Similarly, we also assume the presence of a simulator $\mathcal{S}^{\mathrm{NR}}_{\mathsf{ckt}_l}$, which can be invoked by $\mathcal{S}_f$ to simulate the steps of the honest parties for the protocol $\mathcal{S}^{\mathrm{NR}}_{\mathsf{ckt}_l}$. We do not explicitly give the steps of $\mathcal{S}^{\mathrm{NR}}_{\mathsf{ckt}_l}$, but rather appeal to the simulator of the MPC protocol of [12] because the protocol steps of $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$ are almost the same as the MPC protocol of [12], bar the fault-localization steps. However, simulating the steps of fault-localization is straight forward, since the simulator will know the entire states of all the honest parties in $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$ and so any wrong-doings by the corrupted parties can be easily identified by the simulator exactly as it was identified by an honest party in $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$.

It is easy to show that $\mathrm{IDEAL}_{\mathcal{F}_f, \mathcal{S}_f, \mathcal{Z}} \overset{c}{\approx} \mathrm{REAL}_{\Pi_f, \mathcal{A}, \mathcal{Z}}$ in the ($\mathcal{F}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{BC}}, \mathcal{F}_{\textsc{Committee}}, \mathcal{F}_{\textsc{Gen}[\cdot]}, \mathcal{F}_{\textsc{GenRand}\langle \cdot \rangle}$, $\mathcal{F}_{\mathrm{ZK.BC}}$)-hybrid settings due to the privacy of the the secret sharing schemes and the statistical hiding property of the underlying commitment scheme. For the correctness of the protocol, we rely on the trapdoor security and binding properties of the underlying double trapdoor commitment scheme.

Simulator $\mathcal{S}_f$

The simulator plays the role of the honest parties and simulates each step of the protocol Π_f as follows. The communication of the $\mathcal{Z}$ with the adversary $\mathcal{A}$ is handled as follows: Every input value received by the simulator from $\mathcal{Z}$ is written on $\mathcal{A}$'s input tape. Likewise, every output value written by $\mathcal{A}$ on its output tape is copied to the simulator's output tape (to be read by the environment $\mathcal{Z}$). The simulator then does the following for the session ID sid:

Initialization. $\mathcal{S}_f$ sets its internal variables $\mathcal{L} = \mathcal{P}$, $\mathfrak{n} = n$, $\mathfrak{t} = t$ and $\mathsf{NewCom} = 1$.

CRS Generation. On receiving (sid, i) from every $P_i \in T$, simulator $\mathcal{S}_f$, on behalf of $\mathcal{F}_{\mathrm{CRS}}$, computes $\mathsf{Gen}(1^\kappa) \to (\mathsf{ck}, \tau_0, \tau_1)$ and $G(1^\kappa) \to (\mathsf{pk}, \mathsf{sk})$, sets $\mathsf{CRS} = (\mathsf{ck}, \mathsf{pk})$ and sends (sid, i, CRS) to every $P_i \in T$.

Input commitment. On behalf of every honest party $P_i \in \mathcal{P} \setminus T$, $\mathcal{S}_f$ picks three random polynomials over $\mathbb{F}_p$, $f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot)$ of degree t such that $f^{(i)}(0) = 0$ and imitates the behavior of the honest parties. That is, $\mathcal{S}_f$ computes the commitment $\mathbf{C}_{f^{(i)}(0), g^{(i)}(0), h^{(i)}(0)} = \mathsf{Comm}_{\mathsf{ck}}(f^{(i)}(0); g^{(i)}(0), h^{(i)}(0))$ and sends $(\mathsf{sid}, i, \mathbf{C}_{f^{(i)}(0), g^{(i)}(0), h^{(i)}(0)})$ to every corrupted $P_j \in T$ on behalf of $\mathcal{F}_{\mathrm{BC}}$. When a corrupted $P_i \in T$ invokes $\mathcal{F}_{\mathrm{BC}}$ with $(\mathsf{sid}, i, \mathbf{C}_{f^{(i)}(0), g^{(i)}(0), h^{(i)}(0)}, \mathcal{P})$, simulator $\mathcal{S}_f$ acts on behalf of $\mathcal{F}_{\mathrm{BC}}$ and sends $\mathbf{C}_{f^{(i)}(0), g^{(i)}(0), h^{(i)}(0)}$ to every $P_j \in T$.

$[\cdot]$-sharing of Inputs. For every honest $P_i \in \mathcal{P} \setminus T$, simulator $\mathcal{S}_f$ acts on behalf of functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$ with $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$ and hands $(\mathsf{sid}, j, i, [f^{(i)}(0)]_j)$ to every $P_j \in T$. Then for every corrupted $P_i \in T$, on receiving $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$ from P_i (as the dealer), $\mathcal{S}_f$, on behalf of $\mathcal{F}_{\mathrm{GEN}[\cdot]}$, sends $(\mathsf{sid}, j, i, [f^{(i)}(0)]_i)$ to every $P_j \in T$, after verifying the polynomials $f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot)$ with respect to the corresponding commitment $\mathbf{C}_{f^{(i)}(0), g^{(i)}(0), h^{(i)}(0)}$ (as done by the functionality $\mathcal{F}_{\mathrm{GEN}[\cdot]}$). Locally, simulator maintains the following information:

- $\mathcal{S}_f$ stores the input of corrupted $P_i \in T$ as $x^{(i)} = f^{(i)}(0)$, where $f^{(i)}(\cdot)$ is received from corrupted P_i. Further it sets the input of honest $P_i \in \mathcal{P} \setminus T$ as $x^{(i)} = 0$.
- For every $P_i \in \mathcal{P}$, it stores the entire $[x^{(i)}]$.

$\mathcal{S}_f$ hands $\{x^{(i)}\}_{P_i \in T}$ to the MPC functionality $\mathcal{F}_f$ on behalf of the corrupted parties and gets back the outputs y from the functionality. Next $\mathcal{S}_f$ computes the remaining circuit using 0s as the inputs of the honest parties and $\{x^{(i)}\}_{P_i \in T}$ as the inputs of the corrupted parties. For these inputs, it knows the value to be associated with each wire of the circuit. Thus it knows the circuit output $\bar{y}$ resulted from the above set of inputs, namely 0s as the inputs of the honest parties and $\{x^{(i)}\}_{P_i \in T}$ as the inputs of the corrupted parties.

Start of while loop over the sub-circuits. Set $l = 1$ and while $l < L$, $\mathcal{S}_f$ continues as follows:

- **Committee Selection.** If $\mathsf{NewCom} = 1$, on receiving $(\mathsf{sid}, i, \mathcal{L})$ from every party $P_i \in T$, $\mathcal{S}_f$ on behalf of $\mathcal{F}_{\mathrm{COMMITTEE}}$ picks $\mathfrak{c}$ parties from its local set $\mathcal{L}$ at random and assigns them to $\mathcal{C}$. It then sends $(\mathsf{sid}, P_i, \mathcal{C})$ to every $P_i \in T$.
- **$[\cdot]$ to $\langle\cdot\rangle_{\mathcal{C}}$ Conversion of Inputs of ckt_l.** Let $[x_1], \ldots, [x_{\mathsf{in}_l}]$ denote $[\cdot]$-sharing of the inputs to the sub-circuit ckt_l. For $k \in \{1, \ldots, \mathsf{in}_l\}$, $\mathcal{S}_f$ invokes the sub-simulator $\mathcal{S}_{[\cdot] \to \langle\cdot\rangle}$ (Fig. 20) that simulates the steps of the honest parties in $\Pi_{[\cdot] \to \langle\cdot\rangle}$, with $(\mathsf{sid}, \{[x_k]_i\}_{P_i \in \mathcal{P} \setminus T}, \mathcal{C})$ (namely with the shares corresponding to the honest parties). The sub-simulator returns $\mathcal{S}_f$ with $(\mathsf{sid}, \{\langle x_k\rangle_i\}_{P_i \in \mathcal{C} \wedge (\mathcal{P} \setminus T)})$.
- **Evaluation of the Sub-circuit ckt_l.** The simulator $\mathcal{S}_f$ invokes the simulator $\mathcal{S}^{\mathrm{NR}}_{\mathsf{ckt}_l}$ (namely the simulator of the MPC protocol of [12] with the appropriate modifications in our context to do fault localization) for simulating the steps of the honest parties in the protocol $\Pi^{\mathrm{NR}}_{\mathsf{ckt}_l}$.
- **$\langle\cdot\rangle_{\mathcal{C}}$ to $[\cdot]$ conversion of Outputs of ckt_l.** $\mathcal{S}_f$ invokes $\mathcal{S}_{\langle\cdot\rangle \to [\cdot]}$ with $(\mathsf{sid}, \{\langle y_k\rangle_i\}_{P_i \in \mathcal{C} \wedge (\mathcal{P} \setminus T)}, \mathcal{C})$ for every $k \in \{1, \ldots, \mathsf{out}_l\}$ and gets back either $(\mathsf{sid}, \{[y_k]_i\}_{P_i \in \mathcal{P} \setminus T})$ or $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ or $(\mathsf{sid}, i, \mathsf{Failure}, P_a)$ and does the following:
  - If $(\mathsf{sid}, \{[y_k]_i\}_{P_i \in \mathcal{P} \setminus T})$ is received for every k, increment $l = l + 1$, set $\mathsf{NewCom} = 0$, store the sharings and return to the while loop.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a, P_b)$ is received for *some* $k \in \mathsf{out}_l$, update $\mathcal{L}$ as $\mathcal{L} = \mathcal{L} \setminus \{P_a, P_b\}$, $\mathfrak{t}$ as $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n}$ as $\mathfrak{n} = \mathfrak{n} - 2$.
  - If $(\mathsf{sid}, i, \mathsf{Failure}, P_a)$ is received for *some* $k \in \mathsf{out}_l$, update $\mathcal{L}$ as $\mathcal{L} = \mathcal{L} \setminus \{P_a\}$, $\mathfrak{t}$ as $\mathfrak{t} = \mathfrak{t} - 1$, $\mathfrak{n}$ as $\mathfrak{n} = \mathfrak{n} - 1$.
  - Set $\mathsf{NewCom} = 1$ and go to **Committee Selection Step**.

Output Rerandomization Let $[\bar{y}]$ denote the $[\cdot]$-sharing of the output of ckt. $\mathcal{S}_f$ invokes $\mathcal{S}_{\mathrm{RANDZERO}[\cdot]}$ with input $(\mathsf{sid}, y - \bar{y})$. $\mathcal{S}_{\mathrm{RANDZERO}[\cdot]}$ simulates the honest parties in protocol $\Pi_{\mathrm{RANDZERO}[\cdot]}$ and returns to $\mathcal{S}_f$ $(\mathsf{sid}, \{[y - \bar{y}]_i\}_{P_i \in (\mathcal{P} \setminus T)})$. $\mathcal{S}_f$ locally computes $[y]_i = [\bar{y}]_i + [y - \bar{y}]_i$ for every $P_i \in \mathcal{P} \setminus T$.

Output Computation. On behalf of every honest P_i, $\mathcal{S}_f$ sends $(\mathsf{sid}, i, j, f_i, g_i, h_i)$ to every $P_j \in T$ where $[y]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j, g_j, h_j}\}_{P_j \in \mathcal{P}})$. Clearly every $P_i \in T$ will recover y at the end due to the output rerandomization step.

The simulator then outputs $\mathcal{A}$'s output and terminate.

Fig. 19. Simulator for the adversary $\mathcal{A}$ corrupting at most t parties in the set $T \subset \mathcal{P}$ in the protocol Π_f.

Simulator $\mathcal{S}_{[\cdot]\to\langle\cdot\rangle}$

For session id sid, on receiving $(\mathsf{sid}, \{[s]_i\}_{P_i \in \mathcal{P}\setminus T}, \mathcal{X})$ from $\mathcal{S}_f$, $\mathcal{S}_{[\cdot]\to\langle\cdot\rangle}$ interacts with the corrupted parties in T on behalf of the honest parties in an instance of Protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$. The simulator $\mathcal{S}_{[\cdot]\to\langle\cdot\rangle}$ is aware of the internal state of the honest parties corresponding to the $[s]$. The simulator proceeds as follows:

Verifiably $\langle\cdot\rangle$-sharing the Share and Opening Information in $[s]_i$. First, $\mathcal{S}_{[\cdot]\to\langle\cdot\rangle}$ acts on behalf of every honest P_i as the dealer in an instance of $\Pi_{\langle\cdot\rangle}$ with $(\mathsf{sid}, [s]_i, \mathcal{X})$ such that $[s]_i = (f_i, g_i, h_i, \{\mathbf{C}_{f_j,g_j,h_j}\}_{P_j \in \mathcal{P}})$. It also simulates every honest party P_i in an instance of $\Pi_{\langle\cdot\rangle}$ where a corrupted $P_k \in T$ acts as a dealer.

Identifying the Correctly $\langle\cdot\rangle$-shared Shares of s and Generating $\langle s\rangle_{\mathcal{X}}$. If a corrupted $P_i \in T$ is caught cheating during conflict resolution step, then exclude it from a set $\mathcal{H}$ that is initialized to $\mathcal{P}$. Otherwise, for every corrupted $P_i \in T$, let it receive $\langle f_i\rangle_k$ on behalf of every honest $P_k \in (\mathcal{P}\setminus T)\wedge\mathcal{X}$. Without loss of generality, let[a] $\mathcal{H} = \{P_1, \ldots, P_{|\mathcal{H}|}\}$ and let $c_1, \ldots, c_{|\mathcal{H}|}$ be the publicly known Lagrange interpolation coefficients, such that $c_1 f_1 + \ldots + c_{|\mathcal{H}|} f_{|\mathcal{H}|} = s$. Then the simulator locally computes $\langle s\rangle_i = c_1\langle f_1\rangle_i + \ldots + c_{|\mathcal{H}|}\langle f_{|\mathcal{H}|}\rangle_i$ on behalf of every *honest* $P_i \in \mathcal{X}$ and returns $(\mathsf{sid}, \{\langle s\rangle_i\}_{P_i \in \mathcal{X}\wedge(\mathcal{P}\setminus T)})$ to $\mathcal{S}_f$.

[a] The set $\mathcal{H}$ will be of size more than $t+1$.

Fig. 20. Simulator $\mathcal{S}_{[\cdot]\to\langle\cdot\rangle}$ to be Invoked by the MPC Simulator $\mathcal{S}_f$ for Simulating the Steps of Sub-protocol $\Pi_{[\cdot]\to\langle\cdot\rangle}$ in Π_f

Simulator $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$

For session id sid, on receiving $(\mathsf{sid}, \{\langle s\rangle_i\}_{P_i \in \mathcal{X}\wedge(\mathcal{P}\setminus T)}, \mathcal{X})$ from $\mathcal{S}_f$, $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ interacts with the corrupted parties in T on behalf of the honest parties in an instance of Protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$. Note that the simulator is aware of the internal state of the honest parties corresponding to the $\langle s\rangle$. The simulator proceeds as follows:

- $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$, on behalf of every *honest* $P_i \in \mathcal{X}$, interprets $\langle s\rangle_i$ as $(s_i, u_i, v_i, \{\mathbf{C}^{P_i}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}})$ and sends $(\mathsf{sid}, k, i, \{\mathbf{C}^{P_i}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}})$ to every $P_k \in T$ (acting on behalf of functionality $\mathcal{F}_{\text{BC}}$ that would have been called by P_i in the hybrid protocol). On receiving $(\mathsf{sid}, i, \{\overline{\mathbf{C}^{P_i}_{s_j,u_j,v_j}}\}_{P_j\in\mathcal{X}}, \mathcal{P})$ from every $P_i \in T$, $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ acts on behalf of $\mathcal{F}_{\text{BC}}$ and sends $(\mathsf{sid}, k, i, \{\overline{\mathbf{C}^{P_i}_{s_j,u_j,v_j}}\}_{P_j\in\mathcal{X}})$ to every $P_k \in T$.
- If there exists a pair of parties $P_a, P_b \in \mathcal{X}$, such that P_a is honest and $P_b \in T$ and $\{\mathbf{C}^{P_a}_{s_j,u_j,v_j}\}_{P_j\in\mathcal{X}} \neq \{\overline{\mathbf{C}^{P_b}_{s_j,u_j,v_j}}\}_{P_j\in\mathcal{X}}$, then return $(\mathsf{sid}, \mathsf{Failure}, P_a, P_b)$ to $\mathcal{S}_f$ and halt.
- On behalf of every *honest* $P_i \in \mathcal{X}$, $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ acts as a D and selects $f^{(i)}(\cdot), g^{(i)}(\cdot)$ and $h^{(i)}(\cdot)$ such that they are random polynomials of degree at most t, subject to the condition that $f^{(i)}(0) = s_i, g^{(i)}(0) = u_i$ and $h^{(i)}(0) = v_i$. Then on behalf of $\mathcal{F}_{\text{GEN}[\cdot]}$, $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ creates the $[s_i]$ exactly in the way $\mathcal{F}_{\text{GEN}[\cdot]}$ would compute on input $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$ from P_i as the dealer. Then it hands $(\mathsf{sid}, k, i, [s_i]_k)$ to every $P_k \in T$. On receiving $(\mathsf{sid}, i, \overline{f^{(i)}}(\cdot), \overline{g^{(i)}}(\cdot), \overline{h^{(i)}}(\cdot))$ from a corrupted $P_i \in T$ acting as D, $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ acts exactly as $\mathcal{F}_{\text{GEN}[\cdot]}$ and returns either $(\mathsf{sid}, i, k, \mathsf{Failure})$ or $(\mathsf{sid}, i, k, [s_k]_i)$ to every $P_i \in T$. $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ returns $(\mathsf{sid}, \mathsf{Failure}, P_k)$ to $\mathcal{S}_f$ when $(\mathsf{sid}, i, k, \mathsf{Failure})$ was generated for any $P_k \in T$. Otherwise, it locally computes $[s]_i = \sum_{P_k\in\mathcal{X}}[s_k]_i$ for every honest $P_i \in \mathcal{P}\setminus T$, returns $(\mathsf{sid}, \{[s]_i\}_{P_i \in Parties\setminus T})$ to $\mathcal{S}_f$ and halts.

Fig. 21. Simulator $\mathcal{S}_{\langle\cdot\rangle\to[\cdot]}$ to be Invoked by the MPC Simulator $\mathcal{S}_f$ for Simulating the Steps of Sub-protocol $\Pi_{\langle\cdot\rangle\to[\cdot]}$ in Π_f

Simulator $\mathcal{S}_{\textsc{RandZero}[\cdot]}$

For the session id sid, on receiving $(\mathsf{sid}, y - \bar{y})$ from $\mathcal{S}_f$, $\mathcal{S}_{\textsc{RandZero}[\cdot]}$ interacts with the corrupted parties in T on behalf of the honest parties in an instance of Protocol $\Pi_{\textsc{RandZero}[\cdot]}$. The simulator proceeds as follows:

Publicly Committing 0:

- On behalf of honest party P_h (it just chooses any honest party from the set $\mathcal{P}$) randomly selects $u_h, v_h \in \mathbb{F}_p$, sets $r_h = y - \bar{y}$ and computes $\mathbf{C}_{r_h,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(y - \bar{y}; u_i, v_i)$. On behalf of every other honest party P_i, it randomly selects $u_i, v_i \in \mathbb{F}_p$, sets $r_i = 0$ and computes $\mathbf{C}_{r_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(r_i; u_i, v_i)$. On behalf of $\mathcal{F}_{\text{ZK.BC}}$ corresponding to every honest P_i, it then sends $(\mathsf{sid}, i, \mathbf{C}_{r_i,u_i,v_i})$ to every $P_j \in T$. On receiving $(\mathsf{sid}, i, \mathbf{C}_{r_i,u_i,v_i}, u_i, v_i)$ from every corrupted $P_i \in T$, it acts as $\mathcal{F}_{\text{ZK.BC}}$ and verifies if $\mathbf{C}_{r_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(0; u_i, v_i)$. It the tests passes, then the simulator on behalf of $\mathcal{F}_{\text{ZK.BC}}$ sends $(\mathsf{sid}, i, \mathbf{C}_{r_i,u_i,v_i})$ to every $P_j \in T$. Otherwise, it sends $(\mathsf{sid}, i, \perp)$ to every $P_j \in T$.
- It then constructs a set $\mathcal{T}$, initialized to $\emptyset$ and include in $\mathcal{T}$ all the honest parties in $\mathcal{P}$ and $P_i \in T$ if $\mathbf{C}_{r_i,u_i,v_i} = \mathsf{Comm}_{\mathsf{ck}}(0; u_i, v_i)$ was true for P_i.

$[\cdot]$**-sharing** 0:

- On behalf of honest party P_i, it selects three random polynomials $f^{(i)}(\cdot), g^{(i)}(\cdot)$ and $h^{(i)}(\cdot)$ each of degree at most t, subject to the condition that $f^{(i)}(0) = r_i, g^{(i)}(0) = u_i$ and $h^{(i)}(0) = v_i$. On behalf of $\mathcal{F}_{\textsc{Gen}[\cdot]}$ for an honest P_i, it sends $(\mathsf{sid}, j, i, f_j^{(i)}, g_j^{(i)}, h_j^{(i)})$ to every $P_j \in T$. Then for every corrupted $P_i \in T$, on receiving $(\mathsf{sid}, i, f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot))$ from P_i (as the dealer), $\mathcal{S}_{\textsc{RandZero}[\cdot]}$, on behalf of $\mathcal{F}_{\textsc{Gen}[\cdot]}$, sends $(\mathsf{sid}, j, i, [f^{(i)}(0)]_i)$ to every $P_j \in T$, after verifying the polynomials $f^{(i)}(\cdot), g^{(i)}(\cdot), h^{(i)}(\cdot)$ with respect to the corresponding commitment $\mathbf{C}_{r_i,u_i,v_i}$ (as done by the functionality $\mathcal{F}_{\textsc{Gen}[\cdot]}$). If the polynomials fails the test, remove P_i from $\mathcal{T}$.
- It locally computes $[y - \bar{y}]_i = \sum_{P_j \in \mathcal{T}} [r_j]_i$ and returns $(\mathsf{sid}, \{[y - \bar{y}]_i\}_{P_i \in \mathcal{P} \setminus T})$ to $\mathcal{S}_f$ and halt.

Fig. 22. Simulator for $\Pi_{\textsc{RandZero}[\cdot]}$