On Linear Communication Complexity for (Maximally) Fluid MPC

Alexander Bienstock,[1] Daniel Escudero[2] and Antigoni Polychroniadou[2]

[1] New York University, New York, U.S.A.
[2] J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, New York, U.S.A.

Abstract. Secure multiparty computation protocols with dynamic parties, which assume that honest parties do not need to be online throughout the whole execution of the protocol, have recently gained a lot of traction for computations of *large scale* distributed protocols, such as blockchains. More specifically, in *Fluid* MPC, introduced in (Choudhuri *et al.* CRYPTO 2021), parties can dynamically join and leave the computation from round to round. The best known Fluid MPC protocol in the honest majority setting communicates $O(n^2)$ elements per gate where n is the number of parties online at a time. While Le Mans (Rachuri and Scholl, CRYPTO 2022) extends Fluid MPC to the dishonest majority setting with preprocessing, it still communicates $O(n^2)$ elements per gate.

In this work we present alternative Fluid MPC solutions that require $O(n)$ communication per gate for both the information-theoretic honest majority setting and the information-theoretic dishonest majority setting with preprocessing. Our solutions also achieve *maximal fluidity* where parties only need to be online for a single communication round. Additionally, we show that a protocol in the information-theoretic dishonest majority setting with sub-quadratic $o(n^2)$ overhead per gate requires for each of the N parties who may ever participate in the (later) execution phase, $\Omega(N)$ preprocessed data per gate.

1 Introduction

Secure multiparty computation (MPC) is a promising set of techniques that has been studied since the 80s [Yao86, GMW87, CCD87], and aims at enabling a set of mutually distrustful parties to securely compute a given function on their private inputs, without leaking anything but the output of the function. This should hold even if an unknown subset of the parties is corrupted by an adversary which tries to compromise the privacy of the remaining honest parties.

A protocol may be designed to tolerate an arbitrary amount of corruptions, refereed to as the the *dishonest majority* setting, or this can be relaxed to only require security as long as a minority of the parties are corrupted, *honest majority* setting. Even though dishonest majority MPC protocols offer stronger security guarantees tolerating a higher corruption threshold, protocols in the honest majority setting do not require any computational assumptions and tend to be computationally more efficient than the cryptographic machinery required for the dishonest majority setting. Ever since the introduction of MPC, big efforts have been made at improving its efficiency, which has resulted in a rich and fruitful line of works, including [DPSZ12, BENO19] for the dishonest majority setting, and [DN07, GIP$^+$14, CGH$^+$18, GS20, BGIN20, GLO$^+$21, EGPS22] for honest majority.

The narrow set of use-cases that MPC has seen in practice, in spite of the major push to improve its efficiency and the amount of prototype implementations available, may be a direct effect of some of the other limitations present in MPC which are not precisely related to efficiency metrics such as running times or communication complexities. MPC protocols are distributed interactions that obey a set of rules and, importantly, take place over a communication network, such as the internet. In the real world, networks are unstable, nodes join and leave, computers crash, software has bugs and messages take variable times to reach their destination—or they may even not arrive at all. Moreover, in the cloud setting, cloud resiliency is essential given that networks are dynamic and need to tolerate power and regional outages. Unfortunately, most MPC protocols are not designed to tolerate such unstable networking settings which appear in practice. One may argue that *actively* secure protocols do tolerate unstable environments, since, ultimately, they tolerate arbitrary behavior from the set of corrupted parties. This, however, is unsatisfactory: every MPC protocol has a limited amount of active corruptions it can tolerate, and treating genuinely honest parties as corrupt due to, say, networking or software errors is unacceptable. In particular, not only it reduces the amount of actual malicious corruptions the protocol can tolerate, but also deprives the flagged honest party from any security guarantee, since from a definitional point of view corrupt parties do not need any protection.

Fluid MPC. The limitations of MPC mentioned above have been already identified by several previous works [CGG$^+$21, GHK$^+$21b, BJMS20, DEP21, GPS19, RS22]. These works aim at developing MPC solutions for more unstable settings where the parties or the network may fail to accommodate real world conditions. We discuss these in more detail in the related work section (Section 1.2). In this work we focus on the *Fluid MPC setting*, introduced in [CGG$^+$21]. This approach aims at making MPC more suitable for practical settings by reducing the connectivity requirements that the set of computing parties need to have. Instead of requiring all parties to stay 100% online and with no failures during the whole duration of the computation (which, depending on the protocol, can range from a few minutes to hours, to a whole day), the fluid MPC model allows the parties to join only at parts of the computation, which can be made as "small" as required. More specifically, parties are called online to participate in a committee for a given set of communication rounds. Each committee computes over a specific number of rounds, and once the task is completed the parties in the committee need to *transfer* the state of the computation to the next committee in line, who continues with the computation.

In [CGG$^+$21] it was shown how to instantiate fluid MPC in the *maximal fluidity setting* where parties only need to be online for a single communication round. The parties receive the previous state of the computation, advance one step, and transfer the new state to the next committee. This is done in the setting where the adversary corrupts at most a minority in each chosen committee, or in other words, each committee contains an honest majority. Later, in [RS22], it was shown how to obtain fluid MPC with maximal fluidity in the setting where each committee contains an arbitrary amount of corruptions, or put differently, the corruptions in each committee may constitute of a dishonest majority. The authors aim for an information-theoretic online phase and thus require to assume certain "global preprocessing" among the pool of all parties who will eventually form the different computation committees. The preprocessing is "consumed" at execution time in order to accelerate the computation and is independent of the inputs to the computation and the evaluation function. Both of these works make a noticeable step in making MPC more suitable for practical settings where participants do not need to guarantee stability for extended periods of time.

However, for both the protocols in [CGG$^+$21] and [RS22], the fluidity feature comes at the expense of asymptotically increasing communication complexity with respect to state-of-the art solutions in the non-fluid setting, such as [DPSZ12, BENO19] for dishonest majority, and [DN07, GIP$^+$14, CGH$^+$18, GS20, BGIN20, GLO$^+$21, EGPS22] for honest majority. Existing non-fluid techniques achieve *linear communication complexity*, which is to say that the communication complexity per party does not increase in the average as the set of computing parties increases. In contrast, the protocols from [CGG$^+$21] and [RS22] require *quadratic* communication, which means the communication complexity per party grows linearly with the total number of parties. Such an overhead deprives the scalability of the protocols (*e.g.* if there are twice the amount of parties, then each party needs to send twice the amount of messages). In a way, it seems that achieving the flexibility of the fluid MPC setting comes at the expense of lowering the performance of the resulting protocol. That said, there is no known MPC protocol in the fluid setting which only requires constant communication overhead per party for any corruption threshold (for both honest and dishonest majority).

1.1 Our Contribution

In this work we aim at achieving the efficiency of the non-fluid MPC protocols (*i.e.* overall linear communication complexity), while simultaneously achieving maximal fluidity[3]where a participating party stays online for a single communication round of the secure computation. The goal is to achieve the best of both worlds: flexible enough computation that can somehow mitigate the instability of networks present in the real world without significantly degrading the communication complexity. In particular, we propose fluid MPC protocols with constant communication overhead per party in a committee of size n for any corruption threshold.

[3] Note that even for fluidity $f \geq 2$ the works of [RS22, CGG$^+$21] still only achieve communication complexity $\Omega(n^2 \cdot |C|/f)$; in particular quadratic if f is constant.

In the following we let C be a *layered* arithmetic circuit over a finite field $\mathbb{F}$ with $|C|$ multiplication gates and depth $\mathsf{depth}(C)$. A layered circuit is a type of arithmetic circuit comprised of input, output, addition, multiplication, and identity gates, where all output wires of a given layer go only to the immediately next layer. As is common in MPC protocols that achieve linear communication complexity (e.g., [DN07, DIK10]), we assume that the width of all layers of considered circuits is at least proportional to n. We remark, however, that even with smaller widths, our result is a strict improvement over prior works.

- *Information-theoretic honest majority without preprocessing:* We show that, in the case where each committee contains an honest majority (*i.e.* the setting from [CGG$^+$21]), the circuit C can be evaluated with maximal fluidity by using $(2 \cdot \mathsf{depth}(C) + 14)$ different committees in the execution stage with a total communication of $O(n|C|)$ for the largest n. The input and output stages require a small constant of committees. This protocol requires no preprocessing among the parties for the circuit computation and is information-theoretic.
- *Information-theoretic dishonest majority with preprocessing:* For the case where each committee may contain a dishonest majority (*i.e.* the setting from [RS22]), we show that the circuit C can be evaluated with maximal fluidity by using $(2 \cdot \mathsf{depth}(C) + 5)$ different committees during the execution stage, with a total communication of $O(n|C|)$ for the largest n. This is done starting from a global preprocessing among the pool of all parties that, crucially, does not assume knowledge of the committee assignments ahead of the computation. The online phase is information-theoretic. Similarly to the honest majority case, the input and output stages require a small constant of committees.
- *Lower bound on the amount of preprocessed data:* Finally, on the negative side, we prove a lower bound on the amount of preprocessing required in the information-theoretic dishonest majority setting in order to transfer the secret-shared execution state st from one committee to the next, which is a major building block used in fluid protocols. We show that, in order to achieve sub-quadratic communication for the transfer, it must be the case that each party in the global pool of parties has preprocessing whose size is proportional to $\Omega(N \cdot |\mathsf{st}|)$, where N is the total amount of parties in the global pool. In particular, if each committee computes the output of at most one circuit layer (possibly including multiplication gates) at a time, and each party may participate in a constant fraction of committees in the worst-case, then the total preprocessing per party must be $\Omega(N \cdot |C|)$. Such large preprocessing is not needed for the case in which quadratic communication suffices, since in this case the parties can perform a *resharing* step where each party secret-shares their share to each other party in the next committee, without making use of any preprocessing.
  This result shows that linear communication complexity in the dishonest majority setting comes at a price: if more parties join the global pool, the preprocessing held by each party must necessarily grow.[4]

[4] We remark however that, even in [RS22], the preprocessing per party grows as $\Omega(N)$, even when quadratic communication is achieved. This preprocessing is of a different

As in [CGG$^+$21, RS22], both of our protocols are actively secure and offer security with abort (unanimous abort, if the clients have access to broadcast). A crucial property of the model for fluid computation is that each committee only knows the identities of the parties it is directly connected to on-the-fly without the need to commit to a specific online time way ahead of time. Both [CGG$^+$21, RS22] provide ample motivation for this setting, which includes, for example, applications to computing via distributed systems such as blockchains.

We summarize our results in the following theorems. First some extra notation: given a layered circuit C, we let w_ℓ be the width of the ℓ-th layer in C and for some gate g in C, we let $\ell(g)$ be the index of the layer that g belongs to in C.

Theorem 1 (Informal). *For a layered arithmetic circuit C over a finite field $\mathbb{F}$, there exists an information-theoretic fluid MPC protocol with maximal fluidity which securely computes C in the presence of an active adversary controlling up to $t < n/2$ parties. The protocol uses $2 \cdot \mathsf{depth}(C) + 14$ n-party committees and the communication cost per gate g is $O(n^2/w_{\ell(g)})$ for the largest n. In particular, if the width of all layers is $w = \Omega(n)$, then the total cost is $O(n|C|)$ elements of communication for the largest n.*

Theorem 2 (Informal). *For an arithmetic circuit C over a finite field $\mathbb{F}$, there exists an information-theoretic fluid MPC protocol with maximal fluidity in the preprocessing model which securely computes C in the presence of an active adversary controlling up to $t \geq n/2$ parties. The protocol uses $(2 \cdot \mathsf{depth}(C) + 5)$ n-party committees in the execution stage and the communication cost per gate g is $O(n^2/w_{\ell(g)})$ for the largest n. In particular, if the width of all layers is $w = \Omega(n)$, then the total cost is $O(n|C|)$ elements of communication for the largest n.*

Theorem 3 (Informal). *A secure message transmission protocol for messages of length λ with two n-party committees that uses $o(n^2 \cdot \lambda)$ total communication must have $\Omega(N \cdot \lambda)$ preprocessed data.*

At the crux of our techniques for the protocols are methods to allow reconstruction and resharing of shares in the presence of an active adversary with linear communication in the fluid setting while dealing with the challenge of dynamic committees which are not known ahead of time and announced one by one on the fly. Paradoxically, we adapt the common "king" technique, originated in the non-fluid honest majority setting [DN07], to obtain our results in the dishonest majority setting. And at the same time, we start with the non-fluid SPDZ-like techniques (additive secret sharing and authenticated shares), originated in the non-fluid dishonest majority setting [DPSZ12], to obtain our honest majority

nature though and is not related to resharing, as it comes in the form of pairwise products that are used to build multiplication triples once the exact committees are known. Our result implies that, even if multiplication triples are pre-distributed within each committee, transferring the state from one committee to the next will still require $\Omega(N)$ preprocessing, unless $O(n^2)$ communication is allowed. We discuss this in more detail in Section 6.

results without the need of preprocessing. Finally, we cast the problem of transferring the secret-shared execution state from one committee to the next into the secure message transmission setting and use information theoretic arguments to prove our lower bound.

1.2 Related Work

As mentioned above, our work expands on the work of [RS22, CGG$^+$21] for MPC in the fluid model. There are several works that study other alternative models to Fluid MPC, that also aim at making MPC solutions more resilient to unstable networking settings. Early works like [FHM98a] study MPC in the setting where the adversary may "fail-stop" corrupt some parties, meaning these parties can crash upon an adversarial command, but they do not reveal their state to the adversary. Other models such as Lazy MPC [BJMS20] or sleepy MPC [GPS19] aim at enabling dropouts, but they do so in different ways to Fluid MPC: in Lazy MPC the parties can drop but they cannot return to the computation, and in sleepy in MPC parties can return, but they are assumed to receive all messages sent to them while being offline. In the recent work of [DEP21], a model is presented where the parties can return to the computation while enabling lost messages. Last but not least, the YOSO work of [GHK$^+$21a] splits the computation to committees where each committee passes the control of the computation to the next committee. However, their work requires all parties to be online at all times. Moreover, unlike [GHK$^+$21a], we have no restrictions on the size of committees nor the overlap between them. We refer the reader to Section B in the Supplementary Material for a more thorough description of these works.

2 Technical Overview

We begin by presenting the general idea of our fluid dishonest majority protocol with linear communication complexity, in the preprocessing model. It turns out some of the ideas present in this part of our work will also be helpful for the construction of our fluid honest majority protocol. We then provide a shorter overview for the honest majority protocol in Section 2.3.

2.1 Our Starting Point: Le Mans [RS22]

We first present the high level ideas of the protocol from [RS22], which achieves fluid MPC in the dishonest majority setting with quadratic communication complexity. The overall idea is the following. The circuit at hand is considered to be a *layered circuit*. As in [CGG$^+$21], the invariant that will be kept is that the parties in committee $\mathcal{C}_i$ will hold certain sharings of all the current intermediate values in layer i. Eventually, the last committee obtains shares of the outputs of the circuit, which are then transmitted to the clients. Unlike [CGG$^+$21] however, Le Mans makes use of additive secret-sharing in contrast to Shamir's, due to the setting being dishonest majority in contrast to honest majority.

Resharing and Openings. To maintain the aforementioned invariant, Le Mans makes use of two major blocks. Let us denote by $[x]^{\mathcal{C}}$ additive shares held by committee $\mathcal{C}$ of some secret x. First, to preserve the invariant for addition and identity gates, the parties make use of a *resharing* procedure which enables the parties in committee $\mathcal{C}_i$ to transfer additive sharings of some given values $[x_1]^{\mathcal{C}_i}, \ldots, [x_m]^{\mathcal{C}_i}$ to committee $\mathcal{C}_{i+1}$ so that, as long as there is at least one honest party in each of these two committees, the adversary learns no information about the underlying secrets, and the parties in $\mathcal{C}_{i+1}$ obtain fresh-looking shares $[x_1]^{\mathcal{C}_{i+1}}, \ldots, [x_m]^{\mathcal{C}_{i+1}}$. As we will see later, this resharing can be achieved quite efficiently (in particular, with linear communication complexity) by preprocessing most of the shares that the receiving committee $\mathcal{C}_{i+1}$ should hold.

The second major block used in Le Mans is that of Beaver triples, which is preprocessed material of the form $([a]^{\mathcal{C}_i}, [b]^{\mathcal{C}_i}, [c]^{\mathcal{C}_i})$ where $c = a \cdot b$, held by a committee $\mathcal{C}_i$.[5] This enables sharings $[x]^{\mathcal{C}_i}, [y]^{\mathcal{C}_i}$ held by $\mathcal{C}_i$ to be "multiplied", so that committee $\mathcal{C}_{i+1}$ obtains sharings $[x \cdot y]^{\mathcal{C}_{i+1}}$. This is done by the parties in $\mathcal{C}_i$ locally computing $[x+a]^{\mathcal{C}_i}$ and $[y+b]^{\mathcal{C}_i}$, followed by *opening* these sharings towards $\mathcal{C}_{i+1}$ by each party in $\mathcal{C}_i$ revealing their shares to each party in $\mathcal{C}_{i+1}$. Notice that this takes quadratic communication, and in fact, opening shared values are in essence the exact quadratic bottleneck in Le Mans.

Let us assume temporarily that committee $\mathcal{C}_{i+1}$ has sharings of the same multiplication triple, that is, $([a]^{\mathcal{C}_{i+1}}, [b]^{\mathcal{C}_{i+1}}, [c]^{\mathcal{C}_{i+1}})$. After $\mathcal{C}_{i+1}$ receives $x+a$ and $y+b$, they can locally compute $[x \cdot y]^{\mathcal{C}_{i+1}} = (x+a)(y+b) - (y+b)[a]^{\mathcal{C}_{i+1}} - (x+a)[b]^{\mathcal{C}_{i+1}} + [c]^{\mathcal{C}_{i+1}}$, as required. Now, one way in which committee $\mathcal{C}_{i+1}$ could have obtained the multiplication triple is by assuming they obtain it from the preprocessing. However, notice that this triple has to coincide with the one held by $\mathcal{C}_i$, which is harder to achieve while maintaining the requirement of committee-agnostic preprocessing. Instead, in Le Mans the following approach is taken: the parties in $\mathcal{C}_i$ *reshare* their triple $([a]^{\mathcal{C}_i}, [b]^{\mathcal{C}_i}, [c]^{\mathcal{C}_i})$ to committee $\mathcal{C}_{i+1}$, which enables the latter committee to obtain $([a]^{\mathcal{C}_{i+1}}, [b]^{\mathcal{C}_{i+1}}, [c]^{\mathcal{C}_{i+1}})$. In principle, using the resharing method sketched above, this can be done with linear communication complexity. However, as we will discuss below, active security demands that besides additive sharings the parties also hold sharings of certain MACs. In Le Mans this is handled by using a different resharing method named key-switching, which makes use of openings and hence it suffers from quadratic communication.

Authenticated Sharings. To prevent a corrupt party from breaking security, Le Mans, as all of the dishonest majority MPC protocols, relies on the SPDZ paradigm [DPSZ12] of adding authentication to every shared value. This consists of additive sharings of a global MAC key $[\Delta]$, and for each shared value $[x]$,

[5] Recall that a requirement in the fluid preprocessing model is that the correlations the parties receive have to be agnostic to the specific committee assignments. It may not be clear now, but it turns out multiplication triples are committee-agnostic, if the parties start with BeDOZa-style correlations [BDOZ11]. This will be made clearer.

additive sharings of the MAC of this value, computed as $[\Delta \cdot x]$. In the fluid setting, each committee $\mathcal{C}_i$ who has shares of a value $[x]^{\mathcal{C}_i}$ must also have shares of its MAC $[\Delta_{\mathcal{C}_i} \cdot x]^{\mathcal{C}_i}$, together with shares of the global key $[\Delta_{\mathcal{C}_i}]^{\mathcal{C}_i}$.

The shared MAC key $[\Delta_{\mathcal{C}_i}]^{\mathcal{C}_i}$ can be preprocessed, but it may be different from committee to committee as a result of the committee-agnostic preprocessing condition. Due to this, if the first committee has sharings $([x]^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i} \cdot x]^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i}]^{\mathcal{C}_i})$, and the second committee $\mathcal{C}_{i+1}$ wants to obtain authenticated sharings $([x]^{\mathcal{C}_{i+1}}, [\Delta_{\mathcal{C}_{i+1}} \cdot x]^{\mathcal{C}_{i+1}}, [\Delta_{\mathcal{C}_{i+1}}]^{\mathcal{C}_{i+1}})$ under the different key $\Delta_{\mathcal{C}_{i+1}}$, this cannot be achieved with the simple resharing from before, given that the secret $\Delta_{\mathcal{C}_i} \cdot x$ changes to $\Delta_{\mathcal{C}_{i+1}} \cdot x$. This is addressed in Le Mans by using a key switching method (Protocol $\Pi_{\mathsf{Key\text{-}Switch}}$ in [RS22]) that enables an authenticated value under one committee's key to be transferred to the next committee so that it remains authenticated, but under the key of the next committee.

In a bit more detail, assume preprocessed sharings $([r]^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_{i+1}} \cdot r]^{\mathcal{C}_{i+1}})$.[6] With this, given $[\Delta_{\mathcal{C}_i} \cdot x]^{\mathcal{C}_i}$, committee $\mathcal{C}_{i+1}$ can obtain $[\Delta_{\mathcal{C}_{i+1}} \cdot x]^{\mathcal{C}_{i+1}}$ by letting $\mathcal{C}_i$ first compute locally $[x - r]^{\mathcal{C}_i}$ and then *opening* this value towards committee $\mathcal{C}_{i+1}$. Then, committee $\mathcal{C}_{i+1}$ computes locally $[\Delta_{\mathcal{C}_{i+1}} \cdot x]^{\mathcal{C}_{i+1}} = (x - r) \cdot [\Delta_{\mathcal{C}_{i+1}}]^{\mathcal{C}_{i+1}} + [\Delta_{\mathcal{C}_{i+1}} \cdot r]^{\mathcal{C}_{i+1}}$. Again, because this requires opening shared values from one committee to another, the resulting communication complexity is quadratic.

2.2 The "King Idea" in the Fluid Setting

The reconstruction of a value d requires n^2 communication if all parties just send shares to each other, but it can be done with communication $O(n)$ based on the "king idea" from [DN07]. This is achieved as follows: in a first round, all share owners send their shares to a single party, a "king", who reconstructs d and sends this value to the intended receiving parties in a second round.

Given that, as we have highlighted above, opening shared values is the bottleneck in Le Mans, a natural approach to achieving linear communication complexity in that protocol is to replace all-to-all openings, which have quadratic communication complexity, by the king idea above. However, this imposes a major complication: all-to-all openings require quadratic communication, but only make use of *one single round*, while in contrast, the king idea has linear communication complexity but requires *two rounds*. As a result, using the king idea does not allow committee $\mathcal{C}_i$ to open shared values to committee $\mathcal{C}_{i+1}$, but rather, these can be opened towards a committee $\mathcal{C}_{i+2}$ (by making use of a king in $\mathcal{C}_{i+1}$). At first sight, one may think that the techniques from Le Mans carry over when using this king idea by simply using two committees per circuit layer, instead of one, to accommodate for the extra round required for the reconstruction of shared values. Unfortunately, as we will argue below, such approach is much more complicated than how it looks at a high level.

[6] This form of preprocessing is not committee-agnostic, but a simpler form of it is, and the actual tuple required is obtained by adding an extra resharing step. This is not relevant for our discussion.

Problems with key switching. Recall the key switching protocol from Le Mans sketched above. In that protocol, the parties start with a pair $([r]^{\mathcal{C}_i}, \left[\Delta_{\mathcal{C}_{i+1}} \cdot r\right]^{\mathcal{C}_{i+1}})$, and this enables committee $\mathcal{C}_i$ to "transfer" shares of MACs $[\Delta_{\mathcal{C}_i} \cdot x]^{\mathcal{C}_i}$ to committee $\mathcal{C}_{i+1}$ so that the latter obtains $\left[\Delta_{\mathcal{C}_{i+1}} \cdot x\right]^{\mathcal{C}_{i+1}}$. This approach works for the one-round openings used in the key switching, but if instead we want to use two-round openings with a king, the king would have to be a member of $\mathcal{C}_{i+1}$ itself, and the key switching would have to be done towards committee $\mathcal{C}_{i+2}$ instead. This raises a number of complications. First, such approach would require an initial pair $([r]^{\mathcal{C}_i}, \left[\Delta_{\mathcal{C}_{i+2}} \cdot r\right]^{\mathcal{C}_{i+2}})$, but unfortunately such pair is not easily obtainable. The reason is that $\left[\Delta_{\mathcal{C}_{i+1}} \cdot r\right]^{\mathcal{C}_{i+1}}$ for the inefficient key-switch protocol is obtained in part by the parties of committee $\mathcal{C}_i$ using preprocessed "local MACs" of their shares of r under some keys that each party in committee $\mathcal{C}_{i+1}$ has. This is allowed in the fluid model, since committee $\mathcal{C}_i$ *does* learn the parties of committee $\mathcal{C}_{i+1}$ at some point (so that they know to whom to send their sharings of intermediate circuit values). However, committee $\mathcal{C}_i$ *never* learns the parties of committee $\mathcal{C}_{i+2}$, so we do not have the preprocessing required to obtain $\left[\Delta_{\mathcal{C}_{i+2}} \cdot r\right]^{\mathcal{C}_{i+2}}$.

Instead, our approach is to let committee $\mathcal{C}_{i+2}$ obtain sharings of the MAC of the secret x, but under a MAC key corresponding to the previous committee $\mathcal{C}_{i+1}$, that is, $\left[\Delta_{\mathcal{C}_{i+1}} \cdot x\right]^{\mathcal{C}_{i+2}}$. The Le Mans key switching protocol is naturally extended to achieve this by using the king of committee $\mathcal{C}_{i+1}$ to reconstruct $(x - r)$ to the parties of committee $\mathcal{C}_{i+2}$ and also having committee $\mathcal{C}_{i+1}$ reshare $\left[\Delta_{\mathcal{C}_{i+1}} \cdot r\right]^{\mathcal{C}_{i+1}}$ and $\left[\Delta_{\mathcal{C}_{i+1}}\right]^{\mathcal{C}_{i+1}}$ with committee $\mathcal{C}_{i+2}$. Committee $\mathcal{C}_{i+2}$ can then perform the same computation as committee $\mathcal{C}_{i+1}$ did before with these sharings to obtain $\left[\Delta_{\mathcal{C}_{i+1}} \cdot x\right]^{\mathcal{C}_{i+2}}$. However, with this key switching protocol, we need to take some extra care in our protocol to ensure that MACs of intermediate circuit values do not "fall behind".

In particular, we maintain the invariant that the inputs to the gates at the circuit layer which some committee $\mathcal{C}_i$ processes must be authenticated under the MAC key of committee $\mathcal{C}_{i-2}$. For example, $([x]^{\mathcal{C}_{i+2}}, [\Delta_{\mathcal{C}_i} \cdot x]^{\mathcal{C}_{i+2}})$. This is achieved by preprocessing a multiplication triple where the sharings of a and b are authenticated under the MAC keys of both committees $\mathcal{C}_{i-2}$ and $\mathcal{C}_{i-1}$. For example, the a sharing is of the form: $([a]^{\mathcal{C}_{i-2}}, \left[\Delta_{\mathcal{C}_{i-2}} \cdot a\right]^{\mathcal{C}_{i-2}}, \left[\Delta_{\mathcal{C}_{i-1}} \cdot a\right]^{\mathcal{C}_{i-1}})$.[7] Now, committee $\mathcal{C}_{i-2}$ can first reshare the triple that is authenticated under their MAC key Δ_{i-2} to committee $\mathcal{C}_{i-1}$, who can then reshare it to committee $\mathcal{C}_i$. Assuming the invariant holds for committee $\mathcal{C}_i$, it can then successfully compute authenticated sharings of $(x + a)$ and $(y + b)$ under MAC key Δ_{i-2} needed to multiply x and y. Also, from the above resharing by committee $\mathcal{C}_{i-2}$, and the MACs on the triple that committee $\mathcal{C}_{i-1}$ already holds, committee $\mathcal{C}_{i-1}$ obtains the triple authenticated under their MAC key Δ_{i-1}. Committee $\mathcal{C}_{i-1}$ can then key switch the triple with committee $\mathcal{C}_{i+1}$ using a king in committee $\mathcal{C}_i$. From

[7] This kind of triple authenticated under the MAC keys of both committees $\mathcal{C}_{i-2}$ and $\mathcal{C}_{i-1}$ can indeed still be computed from our actual committee-agnostic preprocessing.

this key switching, committee $\mathcal{C}_{i+1}$ receives the triple authenticated under the MAC key of committee $\mathcal{C}_i$, e.g., for the a part: $([a]^{\mathcal{C}_{i+1}}, [\Delta_{\mathcal{C}_i} \cdot a]^{\mathcal{C}_{i+1}})$. Committee $\mathcal{C}_{i+1}$ can then reshare this triple to committee $\mathcal{C}_{i+2}$, who can then use the above multiplication technique to obtain $([xy]^{\mathcal{C}_{i+2}}, [\Delta_{\mathcal{C}_i} \cdot (xy)]^{\mathcal{C}_{i+2}})$, also authenticated under the MAC key of committee $\mathcal{C}_i$. Thus the invariant is preserved.

Authenticating multiplication triples. There is a second and perhaps more subtle problem that arises when using an intermediate king for linear reconstruction. Note that the above preprocessed triples that we can obtain are such that the sharing $[c]^{\mathcal{C}_i}$ is *not* authenticated. This is addressed in Le Mans by letting committee $\mathcal{C}_i$ learn the authentication of $[c]^{\mathcal{C}_i}$, and in fact the whole multiplication triple, from the previous committee $\mathcal{C}_{i-1}$. In a bit more detail, $\mathcal{C}_{i-1}$ obtains $([a]^{\mathcal{C}_{i-1}}, [\Delta_{\mathcal{C}_{i-1}} \cdot a]^{\mathcal{C}_{i-1}}, [b]^{\mathcal{C}_{i-1}}, [\Delta_{\mathcal{C}_{i-1}} \cdot b]^{\mathcal{C}_{i-1}}, [c]^{\mathcal{C}_{i-1}})$ from the preprocessing, and they perform key switching so that committee $\mathcal{C}_i$ obtains the multiplication triple with the MAC shares of the factors, only missing the shares of the MAC of c. To obtain $[\Delta_{\mathcal{C}_i} \cdot c]^{\mathcal{C}_i}$, a pair $([v]^{\mathcal{C}_{i-1}}, [\Delta_{\mathcal{C}_i} \cdot v]^{\mathcal{C}_i})$ is generated using the key switch protocol on a preprocessed pair $([v]^{\mathcal{C}_{i-1}}, [\Delta_{\mathcal{C}_{i-1}} \cdot v]^{\mathcal{C}_{i-1}})$.[8] With the former pair at hand, the parties in $\mathcal{C}_{i-1}$ can open $[c-v]^{\mathcal{C}_{i-1}}$ to $\mathcal{C}_i$, who can then compute locally $[\Delta_{\mathcal{C}_i} \cdot c]^{\mathcal{C}_i} = (c-v) \cdot [\Delta_{\mathcal{C}_i}]^{\mathcal{C}_i} + [\Delta_{\mathcal{C}_i} \cdot v]^{\mathcal{C}_i}$.

We can easily enough tweak our multiplication procedure sketched above so that committee $\mathcal{C}_{i-2}$ instead uses a king in committee $\mathcal{C}_{i-1}$ to open $(c-v)$ to committee $\mathcal{C}_i$. However, recall that using our key-switch procedure, committee $\mathcal{C}_i$ can only obtain sharings from committee $\mathcal{C}_{i-2}$ that are authenticated under the MAC key of committee $\mathcal{C}_{i-1}$. But, we need c to be authenticated under the MAC key of committee $\mathcal{C}_i$ in order to preserve the invariant described above, since these shares of c are used to compute the shares of the output. Thus, we must wait until committee $\mathcal{C}_{i+1}$ to authenticate c under the key of committee $\mathcal{C}_i$. However, since $(x+a)$ and $(y+b)$ are opened to the (possibly corrupt) king of committee $\mathcal{C}_{i+1}$, the adversary could then add errors dependent on x and y to c while authenticating it. The adversary could thus mount a selective failure attack using these errors. To solve this we still use the king technique so that committee $\mathcal{C}_{i-2}$ can open $(c-v)$ to committee $\mathcal{C}_i$. We then have *only* some king in committee $\mathcal{C}_i$ (to preserve linear communication) again forward $(c-v)$ to committee $\mathcal{C}_{i+1}$, who can then authenticate c. To ensure that this king does not cheat as above, we also have the parties of committee $\mathcal{C}_i$ hash the received $(c-v)$ values for *all* multiplication gates at this circuit layer, using a universal hash function. Then the parties of committee $\mathcal{C}_i$ send these hashes to *each* party of $\mathcal{C}_{i+1}$, who use them to check consistency of their received openings. Since these hashes are short, in fact independent of the number of gates at this layer, communication is still efficient. We use a similar hashing technique as part of the procedure that checks the MACs of shared values.

[8] [RS22] uses 'l' instead of our 'v' here.

2.3 Fluid Honest Majority MPC with Linear Communication

Here we comment briefly on how we obtain our results in the honest majority setting. We remark that, for the purpose of this overview, we present our results using as a starting point the previous discussion on dishonest majority. In this section, let us denote by $[x]_t^{\mathcal{C}}$ Shamir shares of degree t, held by the parties in committee $\mathcal{C}$. In the work of [CGG+21], honest majority fluid MPC is achieved by letting the parties in a given committee $\mathcal{C}_i$ hold *Shamir* sharings of the intermediate circuit values $[x_1]_t^{\mathcal{C}_i}, \dots, [x_\ell]_t^{\mathcal{C}_i}$ in the i-th layer, where $t < n/2$. To preserve the invariant observe that, because of the multiplicative properties of Shamir secret-sharing, the parties in $\mathcal{C}_i$ can *locally* obtain sharings of every intermediate value $[y_1]_{t'_1}^{\mathcal{C}_i}, \dots, [y_{\ell'}]_{t'_1}^{\mathcal{C}_i}$ in the next layer, where each degree t'_j is either equal to t (for addition and identity gates), or $2t$, which is less than n (for multiplication gates). At this point, the parties in $\mathcal{C}_i$ can *reshare* these shared values towards committee $\mathcal{C}_{i+1}$, who obtains $[y_1]_{t'_1}^{\mathcal{C}_{i+1}}, \dots, [y_{\ell'}]_{t'_{\ell'}}^{\mathcal{C}_{i+1}}$, hence maintaining the invariant.

While in the dishonest majority setting resharing additively shared values (with no authentication) can be achieved with linear communication complexity assuming certain form of committee-agnostic preprocessing, such approach does not work in our current setting. Here, Shamir secret-sharing is used, and resharing in one round requires a quadratic amount of communication as it is done by each party in $\mathcal{C}_i$ distributing shares of their Shamir share to each party in $\mathcal{C}_{i+1}$, which can be aggregated to obtain Shamir shares of the underlying secret. This is indeed the approach taken in [CGG+21], and this is one of the fundamental reasons for the quadratic communication in that work. A second reason is also similar to the one in the dishonest majority setting, and it is related to the reconstruction of secret-shared values.

We can interpret our protocol in the honest majority setting as addressing the two issues highlighted above using some techniques from the dishonest majority case as a base, while adding other new ones, and for the purpose of this section, we describe our protocol in these terms. In a bit more detail, we overcome the issue of resharing with squared communication by, instead of using Shamir secret-sharing with degree $t < n/2$, using a larger degree $n - 1$, which is in essence equivalent to additive secret-sharing, as used in the dishonest majority setting. In principle, this would enable us to perform resharing with linear communication by using preprocessed data as sketched in Section 2.1. However, an important challenge in the honest majority setting is that we should not use any preprocessing whatsoever since, unlike the dishonest majority setting, it is not required.

Due to the above, our approach for resharing degree-$(n - 1)$ Shamir sharings *without preprocessing* with linear communication is different. Assume committee $\mathcal{C}_i$ has sharings $[x]_{n-1}^{\mathcal{C}_i}$, and the goal is for committee $\mathcal{C}_{i+1}$ to obtain $[x]_{n-1}^{\mathcal{C}_{i+1}}$. Let us write $\mathcal{C}_i = \{P_1, \dots, P_n\}$ and $\mathcal{C}_{i+1} = \{Q_1, \dots, Q_n\}$, and also $[x]_{n-1}^{\mathcal{C}_i} = (x_1, \dots, x_n)$. Assume the parties in $\mathcal{C}_i$ have preprocessed a sharing of zero $[0]_{n-1}^{\mathcal{C}_i} = (r_1, \dots, r_n)$.[9]

[9] As we elaborate on below, this type of preprocessing can in fact be generated "on the fly" by the different committees, so it is not considered preprocessing as such.

Our resharing protocol is summarized as follows: each party P_j sends $x_j + r_j$ to Q_j, and committee $\mathcal{C}_{i+1}$ defines $[x]_{n-1}^{\mathcal{C}_{i+1}}$ to be these received shares. In words, shares are transferred in a "straight line fashion" (after randomizing with shares of zero), and the new sharings are exactly *the same* as the previous ones. This approach does not work in the dishonest majority setting: the adversary can corrupt, say, $P_1, \ldots, P_{n-1}$ in the first committee, and by corrupting Q_n the adversary learns all shares. In contrast, in the honest majority setting, the adversary learns at most t shares in the first committee and t shares in the second, for a total of $\leq 2t < n$ shares, which maintain privacy of the underlying secret. This powerful observation turns out to be the enabling tool for linear communication.

Using degree-$(n-1)$ Shamir sharings means that the shares of the honest parties in a given committee do not determine the underlying secret anymore, which enables a corrupt party to cheat by modifying their share. Importantly, a similar issue was faced in the dishonest majority setting with additive secret-sharing, and fortunately we are able to take a similar approach here by using MACs in order to prevent cheating. We remark that these are not needed in [CGG$^+$21], since they use Shamir sharings of low degree. We do not elaborate on how MACs are used in our protocol to prevent cheating, but we mention that the approach is in spirit similar to the one sketched in the dishonest majority overview.

The final details we comment on are related to the "preprocessing" required in our protocol. As we mentioned initially, it is imperative that our honest majority protocol does not make use of any preprocessing material. However, we already mentioned some form of preprocessing (namely, shares of zero $[0]_{n-1}^{\mathcal{C}_i}$), plus, several ideas from the dishonest majority protocol require preprocessing such as authenticated values $([r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i} \cdot r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i}]_{n-1}^{\mathcal{C}_i})$, or authenticated multiplication triples. Fortunately, in our work we are able to leverage once more the fact that we have an honest majority in order to let committee $\mathcal{C}_{i-1}$ generate the "preprocessing" for committee $\mathcal{C}_i$ *on the fly*. For correlations that are "linear" such as sharings of zero, the approach from [DN07] can be easily adapted, where the parties in committee $\mathcal{C}_{i-1}$ distribute sharings to $\mathcal{C}_i$, and the latter perform randomness extraction using a Vandermonde matrix. On the other hand, for correlations that include a multiplication, like multiplication triples or authenticated values $([r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i} \cdot r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i}]_{n-1}^{\mathcal{C}_i})$, the parties in $\mathcal{C}_{i-1}$ can obtain the linear part $([r]_t^{\mathcal{C}_{i-1}}, [\Delta_{\mathcal{C}_i}]_t^{\mathcal{C}_{i-1}})$ from $\mathcal{C}_{i-2}$ using the ideas we just described for linear correlations (notice the degree is $t < n/2$). Then, the parties in $\mathcal{C}_{i-1}$ locally multiply these sharings, to obtain $[\Delta_{\mathcal{C}_i} \cdot r]_{2t}^{\mathcal{C}_{i-1}}$. Finally, the parties in $\mathcal{C}_{i-1}$ perform the "straight-line" resharing from before so that $\mathcal{C}_i$ obtains $([r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i} \cdot r]_{n-1}^{\mathcal{C}_i}, [\Delta_{\mathcal{C}_i}]_{n-1}^{\mathcal{C}_i})$.

2.4 Technical Overview of SMT Lower Bound

Now we provide an overview for the lower bound on the amount of preprocessed data, for the dishonest majority case. We do this in the context of secure message transmission (SMT). Assume we have some sender A who wants to send some secret value x to a receiver B through two committees that are not known ahead of time. This is related to fluid MPC: we can think of an identity function that is to be computed using two committees. That said, assume that between A and B, there are two (non-overlapping) committees $\mathcal{C}_1$ and $\mathcal{C}_2$, each of size n, that are chosen at random from the larger universe $\mathcal{U}$ of parties of size N. Furthermore, assume that some adversary $\mathcal{A}$ that is trying to learn x is able to (passively) corrupt all-but-one party in each of $\mathcal{C}_1$ and $\mathcal{C}_2$, as well as any other parties in $\mathcal{U}$. In such a setting, we also allow for some *global preprocessing protocol* that the parties of $\mathcal{U}$ can run amongst each other *before* the secret x or committees $\mathcal{C}_1$ and $\mathcal{C}_2$ are chosen. We show that if the size of each preprocessing state is $o(N \cdot |x|)$, then the total communication must be $\Omega(n^2 \cdot |x|)$. The intuition is as follows.

Suppose that $\mathcal{A}$ corrupts all but the first parties of each committee. Furthermore, suppose, towards contradiction, that the size of the message $c_{1,1}$ that the first party of $\mathcal{C}_1$ sends to the first party of $\mathcal{C}_2$ is small ($\ll |x|$). First, this means that $\mathcal{A}$ can guess this message with high probability. Now, suppose that the preprocessing $r_{1,1}$ of the first party of $\mathcal{C}_1$ is not anymore correlated with the preprocessing $r_{2,1}$ of the second party of $\mathcal{C}_2$, than the preprocessing of the rest of the corrupted parties of $\mathcal{C}_1$. This correlation is what the parties of $\mathcal{C}_1$ (perhaps, implicitly) use to construct messages that will eventually result in correct transmission to the receiver B. In particular, any possible preprocessing $r'_{2,1}$ that has non-zero probability weight conditioned on the preprocessing of the parties of $\mathcal{C}_1$, and thus by the above assumption, the corrupted parties of $\mathcal{C}_1$, must enable correct transmission. So, since the first party of $\mathcal{C}_2$ only uses $r_{2,1}$ along with the ciphertexts it receives to produce its message to the receiver B, $\mathcal{A}$ must be able to use a guess for $r_{2,1}$ conditioned on the preprocessing of corrupt parties of $\mathcal{C}_1$ to produce a valid such message. Together with the other messages to the receiver B from the corrupted parties of $\mathcal{C}_2$, $\mathcal{A}$ can reconstruct x with high probability.

So, it must in fact be that $r_{1,1}$ provides some unique information on the preprocessing of $r_{2,1}$ that the corrupted parties of $\mathcal{C}_1$ do not already provide. However, it is just as likely that some other party in $\mathcal{U}$ could have been chosen to be the first party of $\mathcal{C}_1$, in some other execution of the protocol. So, in fact *every* party outside of $\mathcal{C}_1$ and $\mathcal{C}_2$ must provide some unique information on the preprocessing of $r_{2,1}$. Since $\mathcal{A}$ can corrupt as many of these parties as it wishes, if $r_{2,1}$ is small enough (in particular, $o(N \cdot |x|)$), then $\mathcal{A}$ will eventually be able to reconstruct $r_{2,1}$ completely, guess (short) $c_{1,1}$ and thus again reconstruct x with high probability, as above. Therefore, a contradiction is reached, and the size of each (n^2 total) ciphertext $c_{i,j}$ must be $\Omega(|x|)$.

3 Security Model and Preliminaries

We present some of the preliminaries required in our work. First we discuss the fluid model in Section 3.1, and then in Section 3.2 we present our security model. We utilize the universal composability framework of [Can01].

3.1 Modelling Fluid MPC

We first recall at a high level the modelling of Fluid MPC from [RS22, CGG$^+$21]. A more detailed description is given in Section A in the Supplementary Material. We consider the *client-server* model, where there is a universe $\mathcal{U}$ of parties, that includes both the clients, who provide inputs, and servers, who perform computation. Computation is composed of an optional preprocessing stage among all clients and servers, an input phase where clients provide inputs, an execution stage where the servers compute the function, and an output phase where the clients receive output. The execution step is itself divided into *epochs*, where each epoch i runs among a fixed set of servers, or committee $\mathcal{C}_i$. An epoch contains two parts, the computation phase, where the committee performs some computation local to itself, followed by a hand-off phase, where the current committee securely transfers some current state to the next committee. We assume that all parties have access to only point-to-point channels. For simplicity, throughout the paper we may refer to the set of clients, $\mathcal{C}_{\mathsf{clnt}}$, as $\mathcal{C}_0$ (the 0-th committee) and $\mathcal{C}_\ell$ (the last committee).

Fluidity. This is defined as the minimum number of rounds in any given epoch of the execution stage. We say a protocol achieves *maximal fluidity* if each epoch i only lasts for one total round. In this paper, as in [RS22, CGG$^+$21], we only consider maximal fluidity.

Committee formation. The committees used in each epoch are chosen on-the-fly throughout the execution stage. See [CGG$^+$21] for more motivation and details on committee selection. The model of [CGG$^+$21] specifies the formation process via an ideal functionality that samples and broadcasts committees according to the desired mechanism. However, as in [RS22], we desire to divorce the study of committee selection from the actual MPC and simply require that all parties of the current committee $\mathcal{C}_i$ somehow agree on the next committee $\mathcal{C}_{i+1}$. Specifically, the parties of committee $\mathcal{C}_i$ during the hand-off phase of epoch i (and not before) are informed by the environment $\mathcal{Z}$ of its choice of committee $\mathcal{C}_{i+1}$ (i.e., it is a worst-case choice by $\mathcal{Z}$). We make no assumptions or restrictions on the size of committees nor the overlap between committees. In particular, committees may consist of a large number (possibly constant fraction) of parties in the entire universe, $\mathcal{U}$.

Corruptions. We study two different settings for the number of parties that may be corrupted for our model to still require security:

- For *honest majority*, the adversary $\mathcal{A}$ may only corrupt any minority of servers in the committee of each epoch.[10] This is the setting that [CGG$^+$21] studies.
- For *dishonest majority*, the adversary $\mathcal{A}$ may corrupt all-but-one client and all-but-one server in the committee of each epoch. This is the setting that [RS22] studies.

[10] All-but-one client could be corrupted, however.

We consider a *malicious R-adaptive adversary* from [CGG$^+$21] and used in [RS22]. In short, if there is a preprocessing stage, the adversary *statically* chooses some parties to corrupt beforehand. Then, the adversary *statically* chooses a set of clients to corrupt. During each epoch i of the execution phase, after learning which servers are in committee $\mathcal{C}_i$, the adversary *adaptively* chooses a subset of $\mathcal{C}_i$ to corrupt. Upon such a corruption, the adversary learns the server's entire past state and can send messages on its behalf in epoch i. Therefore, when counting the number of corruptions for some epoch i, we must retroactively include those servers in committee $\mathcal{C}_i$ that are corrupted in some later epoch $j > i$. Furthermore, if there is a preprocessing stage, we count a server in committee $\mathcal{C}_i$ as corrupted also if they were corrupted during the preprocessing phase.

3.2 Security Model

To model Fluid MPC, we adapt the dynamic arithmetic black box (DABB) ideal functionality $\mathcal{F}_{\mathsf{DABB}}$ of [RS22]. First, we note that our protocols, as written, achieve *security with selective abort* (same as [RS22, CGG$^+$21]), where the adversary can prevent any clients of his or her choice from receiving output. However, similar to the protocol of [CGG$^+$21] (c.f. Appendix A), our protocols can easily achieve *unanimous abort* (in which honest clients either all receive the output or all abort) if the clients have access to a broadcast channel in the last round or if they implement a broadcast over their point-to-point channels. The same applies to the protocol of [RS22]. Functionality $\mathcal{F}_{\mathsf{DABB}}$, presented below, is parameterized by a finite field $\mathbb{F}_p$, and supports addition and multiplication operations over the field. It keeps track of the current epoch number in a variable i and the committee of the current epoch i in a variable $\mathcal{C}_i$. The functionality receives the identity of the first committee from the clients via input **Init**. During the execution stage, where the current committee may change, the functionality receives the identity of the next committee from the currently active parties via input **Next-Committee** (if it receives inconsistent committees for either of these two inputs, we assume it aborts).

Functionality 1: $\mathcal{F}_{\mathsf{DABB}}$

Parameters: Finite field $\mathbb{F}_p$, universe $\mathcal{U}$ of parties, and set of clients $\mathcal{C}_{\mathsf{clnt}} \subseteq \mathcal{U}$. The functionality assumes that all parties have agreed upon public identifiers id_x, for each variable x used in the computation.

Init: On input $(\mathsf{Init}, \mathcal{C})$ from every party $P_j \in \mathcal{C}_{\mathsf{clnt}}$, where each P_j sends the same set $\mathcal{C} \subseteq \mathcal{U}$, initialize $i = 1$, $\mathcal{C}_1 = \mathcal{C}$ as the first active committee. Send $(\mathsf{Init}, \mathcal{C}_1)$ to $\mathcal{S}$.

Input: On input $(\mathsf{Input}, \mathsf{id}_x, x)$ from some $P_j \in \mathcal{C}_{\mathsf{clnt}}$, and $(\mathsf{Input}, \mathsf{id}_x)$ from all other parties in $\mathcal{C}_{\mathsf{clnt}}$, store the pair (id_x, x). Send $(\mathsf{Input}, \mathsf{id}_x)$ to $\mathcal{S}$.

Next-Committee: On input $(\mathsf{Next\text{-}Committee}, \mathcal{C})$ from every party $P_j \in \mathcal{C}_i$, where each P_j sends the same set $\mathcal{C} \subseteq \mathcal{U}$, update $i = i + 1$, $\mathcal{C}_i = \mathcal{C}$. Send $(\mathsf{Next\text{-}Committee}, \mathcal{C}_i)$ to $\mathcal{S}$.

Add: On input $(\mathsf{Add}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$ from every party $P_j \in \mathcal{C}_i$, compute $z = x + y$ and store (id_z, z). Send $(\mathsf{Add}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$ to $\mathcal{S}$.

Multiply: On input $(\mathsf{Mult}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$ from every party $P_j \in \mathcal{C}_i$, compute $z = x \cdot y$ and store (id_z, z). Send $(\mathsf{Mult}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$ to $\mathcal{S}$.

Output: On input $(\mathsf{Output}, \{\mathsf{id}_{z_m}\})$ from every party $P_j \in \mathcal{C}_{\mathsf{clnt}} \cup \mathcal{C}_i$, where a value z_m for each id_{z_m} has been stored previously, retrieve $\{(\mathsf{id}_{z_m}, z_m)\}$ and send $(\mathsf{Output}, \{(\mathsf{id}_{z_m}, z_m)\})$ to $\mathcal{S}$. Wait for input from $\mathcal{S}$, and if it is $\mathsf{Deliver}$, send the output to every $P_i \in \mathcal{C}_{\mathsf{clnt}}$. Otherwise, abort.

3.3 Preliminaries

Notation. We first note that we will often use $l \in \mathcal{C}_i$ as shorthand to refer to some party $P_l \in \mathcal{C}_{i+1}$. We use $x \leftarrow_\$ \mathcal{X}$ to denote sampling x randomly from distribution $\mathcal{X}$.

Universal hashing. We make use of universal hash function families in both our honest and dishonest majority protocols. A family of hash functions $\mathcal{H} = \{\mathcal{H}_s : \mathbb{F}_p^T \to \mathbb{F}_p\}$ is *universal* if for all $x \neq y \in \mathbb{F}_p^T$,

$$\Pr_s[\mathcal{H}_s(x) = \mathcal{H}_s(y)] \leq 1/p.$$

Functionalities, protocols and procedures. In this work we denote functionalities by $\mathcal{F}$ and some subscript, and protocols by Π and some subscript. We also consider *procedures*, denoted by π and some subscript. These are similar to protocols except that (1) they act like "macros" that can be called within actual protocols and (2) they are not intended to instantiate a given functionality. Instead, security is proven in the protocol where they are used.

Layered circuits. We refer the reader to [CGG$^+$21] for a more precise description on layered circuits. In short, these are arithmetic circuits composed of addition, multiplication and identity gates. The circuit is divided in *layers*, and for each such layer, the inputs to each gate on the layer come directly from the layer above. Every circuit can be made layered by adding enough identity gates.

4 Dishonest Majority

We first turn our attention to the dishonest majority setting, where there is only guaranteed to be at least one honest party in each committee. The protocol Le Mans from [RS22] is set in this setting, and they show how to achieve maximal fluidity by relying on preprocessed partially-authenticated multiplication triples, and using accumulators to both verify openings and multiplication correctness. Le Mans, just like the protocol from [CGG$^+$21], achieves a communication complexity that is quadratic in the size of the committees. However, interestingly, the source of quadratic communication is different for [RS22]. In [CGG^{+}21], as we discussed in the previous section, quadratic communication appears in the *resharing* step, where each committee reshares their status of the computation towards the next committee. In contrast, resharing is not a problem in Le Mans, which stems from the fact that they make use of additive secret sharing, which admits for a very efficient resharing protocol if one is willing to assume certain form of preprocessing. Instead, the quadratic complexity in Le Mans appears from the approach they take to secure multiplication, which we expand on below.

As we have already mentioned Le Mans makes use of preprocessed multiplication triples to make progress at every multiplication layer. This reduces the problem of secure multiplication to that of reconstructing a shared value, which they do by letting each party in a given committee send out their shares to every other party in the next committee, which leads to quadratic communication. Instead, we achieve linear communication by handling these reconstructions in a different way: sharings are reconstructed to a single "king", who sends the reconstructions to the parties in the next committee. This is indeed the standard way in which openings are handled in non-fluid dishonest majority such as SPDZ [DPSZ12], and its derivatives. The details of our protocol are presented below.

Throughout this section we will always use additive n-out-of-n secret sharings. We use the notation $[x]^{\mathcal{C}}$ to denote such a sharing of a value x between the parties of some committee $\mathcal{C}$.

Remark 1 (On the relevance of global preprocessing). We recall that our fluid modelling allows for the committees to be chosen on the fly, which means that any form of preprocessing has to be agnostic to concrete committee assignments. This is one of the major complications we deal with in our work. If this was not the case, that is, if the preprocessing was allowed to depend on the concrete committee choices, then a much simpler approach can be envisioned: the same authenticated triple is preprocessed across two alternate committees, the first uses it to mask the two secrets to be multiplied, a king in an intermediate committee is used for linear reconstruction, and the other committee uses the same triple to compute the final sharings of the product. This is not a possibility in our case.

Functionality for commitments. In this section we will make use of the following functionality, also appearing in [RS22].

Functionality 2: $\mathcal{F}_{\mathsf{commit}}$

The functionality runs between a set of parties $\mathcal{P}$ and an adversary $\mathcal{A}$.

Commit: On input $(\mathsf{commit}, P_i, x, \tau_x)$ from P_i, where τ_x is a previously unused identifier, store (P_i, x, τ_x) and sent (P_i, τ_x) to all parties.
Open: On input $(\mathsf{open}, P_i, \tau_x)$ from P_i, retrieve x and send (x, i, τ_x) to all parties.

4.1 Dishonest Majority Preprocessing

We begin by describing the preprocessing functionality $\mathcal{F}_{\mathsf{prep}}$ that is used for our dishonest majority construction. We let

$$\langle x\rangle^{i,j} := ((x^i, \Delta^i, M^{i,j}, K^{i,j}), (x^j, \Delta^j, M^{j,i}, K^{j,i}))$$

represent a pairwise BeDOZa [BDOZ11] sharing of x between parties P_i and P_j, MAC'd under their respective local MAC keys $K^{i,j}, K^{j,i}$ and shares of the global MAC key Δ^i, Δ^j:

$$M^{i,j} = K^{j,i} + \Delta^j \cdot x^i, \qquad M^{j,i} = K^{i,j} + \Delta^i \cdot x^j.$$

Functionality $\mathcal{F}_{\mathsf{prep}}$ is in charge of distributing first shares Δ^j of the global MAC key Δ to each party in the universe $\mathcal{U}$. It also distributes (i) pairwise BeDOZa sharings $\langle r\rangle^{i,j}$ between each pair of parties in $\mathcal{U}$, (ii) partial multiplication triple sharings $(\langle a\rangle^{i,j}, \langle b\rangle^{i,j}, [c]^{i,j})$ between each pair of parties in $\mathcal{U}$, and (iii) common random values $s_{i,j}$ shared between each pair of parties in $\mathcal{U}$. For (ii), the shared value c is computed as $a^i \cdot b^j + a^j \cdot b^i$, i.e., the cross terms in the product $(a^i + a^j) \cdot (b^i + b^j) = a \cdot b$ of the values a and b for which the two parties have pairwise BeDOZa sharings. Note that $[c]^{i,j}$ is not authenticated and in fact can have additive error of the form $\delta_a \cdot b^j + \delta_b \cdot a^j$, if P_i is corrupted and P_j is honest.

We remark that this is a functionality that we do not aim at instantiating in our work. We refer the reader to [RS22] on how this functionality can be instantiated by having the pool of all parties perform an MPC protocol among themselves. We note that this instantiation requires per-party communication and storage of $O(N \cdot \log |C|)$, if each party may participate in a constant fraction of committees in the worst-case during the execution phase. However, in order to have a purely *information-theoretic* execution phase, the stored preprocessing states from [RS22] must be expanded to size $\Theta(N \cdot |C|)$, *before* the execution phase, consistent with our lower bound from Section 6.

Functionality 3: $\mathcal{F}_{\mathsf{prep}}$

Parameters: Finite fields $\mathbb{F}_p$ and $\mathbb{F}_{p^r}$, parties $P_1, \ldots, P_N$, adversary $\mathcal{A}$, set of honest parties Hon, and set of corrupt parties Corr.
Functionality: Generate pairwise authenticated random values and pairwise partially-authenticated multiplication triples.

1. $\mathcal{F}_{\mathsf{prep}}$ receives from $\mathcal{A}$ the global MAC key shares $\{\Delta^i\}_{i\in\mathsf{Corr}}$ for the corrupt parties. It then samples randomly $\Delta^j \leftarrow_\$ \mathbb{F}_{p^r}$ for each honest party $P_j \in \mathsf{Hon}$.
2. For $2 \cdot \texttt{tot\_trip} + \texttt{tot\_rand}$ number of times, for every $P_i \neq P_j$:
   (a) If both $P_i, P_j \in \mathsf{Hon}$, $\mathcal{F}_{\mathsf{prep}}$ samples random values $r^i, r^j \leftarrow_\$ \mathbb{F}_p$ and MAC keys $K^{i,j}, K^{j,i} \leftarrow_\$ \mathbb{F}_{p^r}$. It then computes $M^{i,j} = K^{j,i} + \Delta^j \cdot r^i \in \mathbb{F}_{p^r}$ and $M^{j,i} = K^{i,j} + \Delta^i \cdot r^j \in \mathbb{F}_{p^r}$.
   (b) If one of P_i or P_j, say P_i w.l.o.g., is in Corr, $\mathcal{F}_{\mathsf{prep}}$ receives from $\mathcal{A}$ share r^i along with MAC $M^{i,j}$ and MAC key $K^{i,j}$. It also samples

random share $r^j \leftarrow_\$ \mathbb{F}_p$, then computes $M^{j,i} = K^{i,j} + \Delta^i \cdot r^j$ and $K^{j,i} = M^{i,j} - \Delta^j \cdot r^i$.

(c) If both P_i and P_j are in Corr, $\mathcal{F}_{\mathsf{prep}}$ receives from $\mathcal{A}$ all corresponding values.

3. For `tot_trip` number of times, for every $P_i \neq P_j$, $\mathcal{F}_{\mathsf{prep}}$ let $\langle a_{m_T} \rangle^{i,j}, \langle b_{m_T+1} \rangle^{i,j}$ be the outputs from the m_T and (m_T+1)-st iterations of Step 2 above. Then:
   (a) If both $P_i, P_j \in \mathsf{Hon}$, $\mathcal{F}_{\mathsf{prep}}$ samples $c^{i,j}, c^{j,i} \in \mathbb{F}_p$ randomly such that $c^{i,j} + c^{j,i} = a^i_{m_T} \cdot b^j_{m_T+1} + a^j_{m_T} \cdot b^i_{m_T+1}$.
   (b) If one of P_i or P_j, say P_i w.l.o.g., is in Corr, $\mathcal{F}_{\mathsf{prep}}$ receives from $\mathcal{A}$ share $c^{i,j}$ and additive errors δ_a, δ_b. It then computes $c^{j,i} = \big((a^i_{m_T} + \delta_a) \cdot b^j_{m_T+1} + a^j_{m_T} \cdot (b^i_{m_T+1} + \delta_b)\big) - c^{i,j}$.
   (c) If both P_i and P_j are in Corr, $\mathcal{F}_{\mathsf{prep}}$ receives from $\mathcal{A}$ all corresponding values.
4. For `tot_shared_rand` number of times, for every $P_i \neq P_j$:
   (a) If both $P_i, P_j \in \mathsf{Hon}$, $\mathcal{F}_{\mathsf{prep}}$ samples random $s_{i,j} \leftarrow_\$ \mathbb{F}_p$.
   (b) If at least one of P_i or P_j, say P_i w.l.o.g., is in Corr, $\mathcal{F}_{\mathsf{prep}}$ receives $s_{i,j}$ from $\mathcal{A}$.
5. Finally, $\mathcal{F}_{\mathsf{prep}}$ outputs all shares of $\langle r \rangle^{i,j}$, $(\langle a \rangle^{i,j}, \langle b \rangle^{i,j}, [c]^{i,j})$, and $s_{i,j}$ computed above to parties in Hon.

We now let $\langle x \rangle^{\mathcal{P},\mathcal{Q}} := \Big(\big(x^i, \{M^{i,j}\}_{j \in \mathcal{Q}}\big)_{i \in \mathcal{P}}, \big(\Delta^j, \{K^{j,i}\}_{j \in \mathcal{Q}}\big)_{i \in \mathcal{P}} \Big)$ represent a pairwise BeDOZa [BDOZ11] sharing of x where each party P_i of $\mathcal{P}$ holds an additive share x^i of the secret x and MACs $M^{i,j}$ of this share under the local keys and shares of the global key $K^{j,i}, \Delta^j$ of each party P_j of $\mathcal{Q}$. Each P_j of $\mathcal{Q}$ indeed holds their share Δ^j of the global MAC key and each local key $K^{j,i}$. These MACs are computed as above. Functionality $\mathcal{F}_{\mathsf{prep}}$ is useful as it can be used to generate partially authenticated multiplication triples and shares of authenticated uniformly random values within any committee, once the identity of this committee is known. This is described in detail in Procedure $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$.

Procedure 1: $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$

Usage: Generate BeDOZa random sharings and partially-authenticated BeDOZa random multiplication triples using the pairwise random sharings and partially-authenticated random multiplication triples from $\mathcal{F}_{\mathsf{prep}}$.

Init: m_T is initialized to 0 and m_R is initialized to $2 \cdot$ `tot_trip`.

1. On input $(\mathsf{rand}, \mathcal{P}, \mathcal{Q})$, each party $P_i \in \mathcal{P}$ outputs $(r^i_{m_R}, \{M^{i,j}_{m_R}\}_{j \in \mathcal{Q}})$ and each $P_j \in \mathcal{Q}$ outputs $(\Delta^j, \{K^{j,i}_{m_R}\}_{i \in \mathcal{P}})$, using the fresh m_R-th rand value from $\mathcal{F}_{\mathsf{prep}}$. Then m_R is incremented: $m_R = m_R + 1$.
2. On input $(\mathsf{trip}, \mathcal{P}, \mathcal{Q})$, each party $P_i \in \mathcal{P}$ outputs $((a^i_{m_T}, \{M^{i,j}_{m_T}\}_{j \in \mathcal{Q}}), (b^i_{m_T+1}, \{M^{i,j}_{m_T+1}\}_{j \in \mathcal{Q}}), a^i_{m_T} \cdot b^i_{m_T+1} + \sum_{l \in \mathcal{P} \setminus \{i\}} c^{i,l}_{m_T})$ and each $P_j \in \mathcal{Q}$ outputs $(\Delta^j, \{K^{j,i}_{m_T}\}_{i \in \mathcal{P}}, \{K^{j,i}_{m_T+1}\}_{i \in \mathcal{P}})$, using the fresh m_T-th partially authenticated triples from $\mathcal{F}_{\mathsf{prep}}$. Then m_T is incremented: $m_T = m_T + 2$.

4.2 Efficient Resharing for Dishonest Majority

For efficient resharing we make use of the techniques in [RS22], which consist of the parties sending additive shares of their shares towards the next committee, but using some preprocessing in the form of pairwise shared randomness in order to precompute most of the shares.[11] Details are given in Procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ below.

Procedure 2: $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$

Usage: $\mathcal{C}_i$ reshares $[r]^{\mathcal{C}_i}$ to $\mathcal{C}_{i+1}$.

1. Each party P_j in $\mathcal{C}_i$ will use the next fresh pairwise shared random values from $\mathcal{F}_{\mathsf{prep}}$, $\{r_{j,l}\}_{l \in \mathcal{C}_{i+1}}$.
2. Each P_j will then take their share r^j of $[r]^{\mathcal{C}_i}$ and locally compute $r^{j,1} = r^j - \sum_{l=2}^{n_{i+1}} r_{j,l}$.
3. Next, each P_j will send their $r^{j,1}$ to P_1 in $\mathcal{C}_{i+1}$.
4. Finally, parties P_l in $\mathcal{C}_{i+1}$ will locally compute their share r^l of $[r]^{\mathcal{C}_{i+1}}$ as $r^l = \sum_{j \in \mathcal{C}_i} r^{l,j}$ (where if $j \neq 1$, each $r^{l,j} = r_{l,j}$, obtained from $\mathcal{F}_{\mathsf{prep}}$).

Lemma 1. *Procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$'s transcript is simulatable by random values.*

Proof. Assume without loss of generality that only party $P_{n_i} \in \mathcal{C}_i$ is honest (the case where other parties are honest follows easily). Now $P_1 \in \mathcal{C}_{i+1}$ is the only party that receives communication in this procedure, so if P_1 is not corrupted, it is easy to simulate. If P_1 is corrupted, assume without loss of generality that $P_{n_{i+1}}$ is honest (again, the case where others are honest follows easily). So, we must argue that simulating $r^{n_i,1}$, that $P_1 \in \mathcal{C}_{i+1}$ receives from $P_{n_i} \in \mathcal{C}_i$ with a random value is valid. From $\mathcal{F}_{\mathsf{prep}}$, we know that only P_{n_i} and $P_{n_{i+1}}$ have knowledge of $r_{n_i,n_{i+1}}$; for everyone else, it is distributed uniformly at random. Thus, since $r_{n_i,n_{i+1}}$ is added to $r^{n_i,1}$, $r^{n_i,1}$ appears uniformly random to $\mathcal{A}$, and we are done. $\square$

As in Le Mans, we also rely on the fact that BeDOZa authenticated sharings can be converted locally into SPDZ sharings. This is carefully discussed in Procedure π_{convert}. First, recall that an authenticated SPDZ sharing of value x amongst Committee $\mathcal{C}$ has the form $[\![x]\!]_{\Delta_{\mathcal{C}}}^{\mathcal{C}} := ([x]^{\mathcal{C}}, [\Delta_{\mathcal{C}} \cdot x]^{\mathcal{C}}, [\Delta_{\mathcal{C}}]^{\mathcal{C}})$. Since each committee $\mathcal{C}$ has its own shared MAC key $\Delta_{\mathcal{C}} = \sum_{j \in \mathcal{C}_i} \Delta^j$, we specify under which committee's MAC key the sharing is authenticated in the subscript.

Procedure 3: π_{convert}

Usage Case 1: For input pairwise-authenticated BeDOZa sharing $\langle x \rangle^{\mathcal{C}_i,\mathcal{C}_i}$, convert to SPDZ sharing $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$.

[11] The parties can instead use pairwise shared PRG seeds, and expand them each time they need the values $r_{j,l}$ used in $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$.

1. Note that

$$\sum_{j\in\mathcal{C}_i}\left(\Delta^j\cdot x^j+\sum_{l\in\mathcal{C}_i}M_l^j-K_l^j\right)=$$

$$\sum_{j\in\mathcal{C}_i}\left(\Delta^j\cdot x^j+\sum_{l\in\mathcal{C}_i}K_j^l+\Delta^l\cdot x^j-K_l^j\right)=\Delta_{\mathcal{C}_i}\cdot x.$$

2. So, each P_j in $\mathcal{C}_i$ outputs as their SPDZ share of x: $(x^j, \Delta^j\cdot x^j+\sum_{l\in\mathcal{C}_i}M_l^j-K_l^j, \Delta^j)$.

Usage Case 2: For input pairwise-authenticated BeDOZa sharing $\langle x\rangle^{\mathcal{C}_i,\mathcal{C}_i\cup\mathcal{C}_{i+1}}$, convert to SPDZ sharings $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$ *and* $[\![x]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$.

1. We first note that $\mathcal{C}_i$ can obtain SPDZ shares of x in the same way as above (by ignoring its pairwise MAC and MAC keys with those parties $P_l\in\mathcal{C}_{i+1}$, as the sum above is written).
2. Now, each $P_j\in\mathcal{C}_i$ additionally computes $M^j=\sum_{l\in\mathcal{C}_{i+1}}M_l^j$ to obtain sharing $[M]^{\mathcal{C}_i}$ and invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on it, along with $[x]^{\mathcal{C}_i}$.
3. Let M^l be the share of $[M]^{\mathcal{C}_{i+1}}$ obtained by P_l. Now note that

$$\sum_{l\in\mathcal{C}_{i+1}}\left(M^l-\sum_{j\in\mathcal{C}_i}K_j^l\right)=$$

$$\sum_{l\in\mathcal{C}_{i+1}}\left(M^l-\sum_{j\in\mathcal{C}_i}M_l^j-\Delta^l\cdot x^j\right)=0+\Delta_{\mathcal{C}_{i+1}}\cdot x.$$

4. So, each P_l in $\mathcal{C}_{i+1}$ outputs as their SPDZ share of x: $(x^l, M^l-\sum_{j\in\mathcal{C}_i}K_j^l, \Delta^l)$.

Lemma 2. *Procedure π_{convert}'s transcript is simulatable by random values.*

Proof. Follows from Lemma 1, since the only communication is invoking $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[M]^{\mathcal{C}_i}$, for which the honest shares are uniformly random, by the security of $\mathcal{F}_{\mathsf{prep}}$. □

4.3 Checking and Maintaining MACs

Similar to our honest majority protocol of Section 5 (and Le Mans), in our dishonest majority protocol, we use n-out-of-n sharings. This means that a malicious adversary can easily add errors to *any* value throughout the computation. Thus, as before, we use MACs, which authenticate every intermediate value used throughout the computation, and guarantee those values' integrity. To attest to the integrity of every value that is opened throughout the protocol, we again use an accumulator that is computed similarly as before. As in the honest majority case, to ensure that the adversary does not cheat, we open a challenge β to the parties of $\mathcal{C}_{i+1}$, even though the values are opened to $\mathcal{C}_i$. The parties of $\mathcal{C}_{i+1}$ then use it to compute a random linear combination on the openings. So, to maintain linear communication, P_1 of $\mathcal{C}_i$ forwards the openings to every party in $\mathcal{C}_{i+1}$, and we ensure that P_1 does not cheat by having all of the other parties of $\mathcal{C}_i$ send hashes of the openings to the parties of $\mathcal{C}_{i+1}$. Since the hashes are short, total communication will still be $O(n|C|)$ if the width of C is $\Omega(n)$. Details are given in Procedure $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ below.

Procedure 4: $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$

Usage: Each committee $\mathcal{C}_i$ incrementally updates a MAC check state $[\sigma]^{\mathcal{C}_i}$ based on the values $\{[\![A_m]\!]_{\Delta_j}^{\mathcal{C}_{i-2}}\}_{m\in[T]}$ (for some $j \leq i-2$) opened to them, which the final committees at the end of the computation use to check that all openings throughout the protocol were performed correctly.

Init: Each Party P_i in committee $\mathcal{C}_2$ (the first to have values opened to it) initially defines their share of $[\sigma]^{\mathcal{C}_2}$ as $\sigma^i = 0$.

Update State: On input $(\mathsf{update}, \{(A_m, [A_m \cdot \Delta_j]^{\mathcal{C}_i})\}_{m\in[T]}, [\Delta_j]^{\mathcal{C}_i})$ from committee $\mathcal{C}_i$, where $\{A_m\}_{m\in[T]}$ were the values opened to $\mathcal{C}_i$:

1. Committee $\mathcal{C}_i$ invokes $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$ with $(\mathsf{rand}, \mathcal{C}_i, \mathcal{C}_{i+1})$ so that $P_j \in \mathcal{C}_i$ gets $(\beta^j, \{M^{j,l}\}_{l\in\mathcal{C}_{i+1}})$ and $P_l \in \mathcal{C}_{i+1}$ gets $(\Delta^l, \{K^{l,j}\}_{j\in\mathcal{C}_i})$.
2. Each $P_j \in \mathcal{C}_i$ also interprets the next pairwise shared random values from $\mathcal{F}_{\mathsf{prep}}$, $\{s_{j,l}\}_{l\in\mathcal{C}_{i+1}}$ as keys to a universal hash family $\mathcal{H} = \{\mathcal{H}_s : \mathbb{F}_p^T \to \mathbb{F}_p\}$ and computes $h_{j,l} = \mathcal{H}_{s_{j,l}}(\{A_m\}_{m\in[T]})$ for each $P_l \in \mathcal{C}_{i+1}$.
3. In parallel: (i) each party $P_j \in \mathcal{C}_i$ then sends to each $P_l \in \mathcal{C}_{i+1}$ their share β^j and corresponding MAC for P_l, $M^{j,l}$, along with the computed hash value $h_{j,l}$; (ii) only $P_1 \in \mathcal{C}_i$ sends $\{A_m\}_{m\in[T]}$ to each $P_l \in \mathcal{C}_{i+1}$; and (iii) all of $\mathcal{C}_i$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\sigma]^{\mathcal{C}_i}$, $\left[\Delta_{\mathcal{C}_j}\right]^{\mathcal{C}_i}$, $\{\left[\Delta_{\mathcal{C}_j} \cdot A_m\right]^{\mathcal{C}_i}\}_{m\in[T]}$.
4. Each $P_l \in \mathcal{C}_{i+1}$ then locally checks that $M^{j,l} = \beta^j \cdot \Delta^l + K^{l,j}$, for each $P_j \in \mathcal{C}_i$, and aborts if any check fails. If not, let $\beta = \sum_{j\in\mathcal{C}_i} \beta^j$.
5. Each P_l additionally uses the pairwise shared random values from $\mathcal{F}_{\mathsf{prep}}$, $\{s_{j,l}\}_{j\in\mathcal{C}_i}$, to compute $h'_{j,l} = \mathcal{H}_{s_{j,l}}(\{A_m\}_{m\in[T]})$, checks that each $h'_{j,l} = h_{j,l}$, and aborts if any check fails; else continues.
6. Each $P_l \in \mathcal{C}_{i+1}$ next locally computes $A = \sum_{m=1}^{T} (\beta)^m \cdot A_m$ and $[\gamma]^{\mathcal{C}_{i+1}} = \sum_{m=1}^{T} (\beta)^m \cdot \left[\Delta_{\mathcal{C}_j} \cdot A_m\right]^{\mathcal{C}_{i+1}}$ (here $(\beta)^m$ is the m-th power of β).
7. It finally updates $[\sigma]^{\mathcal{C}_{i+1}} = [\sigma]^{\mathcal{C}_{i+1}} + [\gamma]^{\mathcal{C}_{i+1}} - \left[\Delta_{\mathcal{C}_j}\right]^{\mathcal{C}_{i+1}} \cdot A$ and invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\sigma]^{\mathcal{C}_{i+1}}$.

Check State: On input check from the clients $\mathcal{C}_{\mathsf{clnt}}$:

1. Each client $P_i \in \mathcal{C}_{\mathsf{clnt}}$ invokes $\mathcal{F}_{\mathsf{commit}}$ on their share σ^i of the MAC check state $[\sigma]^{\mathcal{C}_{\mathsf{clnt}}}$.
2. Then they all open their commitments, and if they are consistent, output Accept if $\sum_{i\in\mathcal{C}_{\mathsf{clnt}}} \sigma^i = 0$; else Reject.

Lemma 3. *Procedure* $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ *is correct, i.e., it accepts if all the opened values* A_m *and the corresponding MACs are computed correctly. Moreover, it is* *sound, i.e., it rejects except with probability at most $(2 + \max_i T_i)/p$ in case at least one opened value is not correctly computed. Furthermore, the transcript of* ***Update State*** *is simulatable.*

Proof. For soundness, we consider all of the points in which the adversary can inject error, when a single committee $\mathcal{C}_i$ is updating $[\sigma]^{\mathcal{C}_i}$. First, note that with all-but-negligible probability, an honest party P_j of $\mathcal{C}_i$ will receive the same (potentially incorrect) $A'_m = A_m + \delta^i_m$ for each m as an honest party P_l of $\mathcal{C}_{i+1}$. This is because their shared universal hash key $s_{j,l}$ from $\mathcal{F}_{\mathsf{prep}}$ is uniformly random and unknown to the adversary. So, since $\mathcal{H}$ is a universal hash family, it holds that if P_j gets $\{A'_m\}_{m\in[T]}$ and P_l gets a different $\{A''_m\}_{m\in[T]}$,

$$\Pr[\mathcal{H}_{s_{j,l}}(\{A'_m\}_{m\in[T]}) = h_{j,l} = h'_{j,l} = \mathcal{H}_{s_{j,l}}(\{A''_m\}_{m\in[T]})] \leq 1/p.$$

The adversary can thus only inject the following kind of errors:

1. When opening shares of some A_m to P_{king} of $\mathcal{C}_{i-1}$, or if P_{king} itself is corrupted, then when sending opened A_m to the honest parties P_j of committee $\mathcal{C}_i$, the adversary can add some δ^i_m error. From above, these δ^i_m's must in fact be independent of the opened challenge randomness for $\mathcal{C}_i$, β^i.
2. When the MAC key $\left[\Delta_{\mathcal{C}_j}\right]^{\mathcal{C}_i}$ is reshared, the adversary can add some additive η^i error.
3. When resharing a MAC $\left[\Delta_{\mathcal{C}_j} \cdot A_m\right]^{\mathcal{C}_i}$, the adversary can add some additive ε^i_m.
4. Finally, when $[\sigma]^{\mathcal{C}_{i+1}}$ is reshared, the adversary can add some additive ζ^i.

So, at the end of the computation (after ℓ committees), if the check passes we will have:

$$0 = [\sigma]^{\mathcal{C}_{\mathsf{clnt}}} = \sum_{i=1}^{\ell} \sum_{m=1}^{T_i} (\beta^i)^m \cdot ((\Delta_{\mathcal{C}_{j_i}} \cdot A^i_m + \varepsilon^i_m) - (\Delta_{\mathcal{C}_{j_i}} + \eta^i) \cdot (A^i_m + \delta^i_m)) + \zeta^i$$

$$= \sum_{i=1}^{\ell} \sum_{m=1}^{T_i} (\beta^i)^m \cdot (\varepsilon^i_m - \Delta_{\mathcal{C}_{j_i}} \cdot \delta^i_m + \eta^i \cdot A^i_m + \delta^i_m \cdot \eta^i) + \zeta^i$$

Recall that we want to show that for any i, m $\delta^i_m \neq 0$ can only happen with negligible probability. First, note that the corrupt parties of committee $\mathcal{C}_i$ cannot forge their share β^j of β to any of the honest parties of $\mathcal{C}_{i+1}$ except with probability $1/p$, by the security of the information-theoretic MAC provided by $\mathcal{F}_{\mathsf{prep}}$. Thus, the reconstructed challenge β must indeed be uniformly random and independent of all other values. Now, assume that indeed there is some $\delta^{i^*}_m \neq 0$. First, we argue that by the Schwartz-Zippel Lemma, with probability $\leq T_{i^*}/p$, $\sum_{m=1}^{T_{i^*}} (\beta^{i^*})^m \cdot \delta^{i^*}_m = 0$. This holds since β^{i^*} is chosen uniformly at random and, by assumption, at least one $\delta^{i^*}_m \neq 0$. So now we rewrite:

$$0 = \sum_{i=1}^{\ell} \sum_{m=1}^{T_i} (\beta^i)^m \cdot (\varepsilon^i_m - \Delta_{\mathcal{C}_{j_i}} \cdot \delta^i_m + \eta^i \cdot A^i_m + \delta^i_m \cdot \eta^i) + \zeta^i$$

$$= -\Delta_{\mathcal{C}_{j_{i^*}}} \cdot \sum_{m=1}^{T_{i^*}} (\beta^{i^*})^m \cdot \delta_m^{i^*} + \sum_{m=1}^{T_{i^*}} (\beta^{i^*})^m \cdot (\varepsilon_m^{i^*} + \eta^{i^*} \cdot A_m^{i^*} + \delta_m^{i^*} \cdot \eta^{i^*}) + \zeta^{i^*} +$$

$$\sum_{i \in [\ell] \setminus \{i^*\}} \sum_{m=1}^{T_i} (\beta^i)^m \cdot (\varepsilon_m^i - \Delta_{\mathcal{C}_{j_i}} \cdot \delta_m^i + \eta^i \cdot A_m^i + \delta_m^i \cdot \eta^i) + \zeta^i.$$

Therefore:

$$\Delta_{\mathcal{C}_{j_{i^*}}} = \frac{1}{\sum_{m=1}^{T_{i^*}} (\beta^{i^*})^m \cdot \delta_m^{i^*}} \left(\sum_{m=1}^{T_{i^*}} (\beta^{i^*})^m \cdot (\varepsilon_m^{i^*} + \eta^{i^*} \cdot A_m^{i^*} + \delta_m^{i^*} \cdot \eta^{i^*}) + \zeta^{i^*} + \right.$$

$$\left. \sum_{i \in [\ell] \setminus \{i^*\}} \sum_{m=1}^{T_i} (\beta^i)^m \cdot (\varepsilon_m^i - \Delta_{\mathcal{C}_{j_i}} \cdot \delta_m^i + \eta^i \cdot A_m^i + \delta_m^i \cdot \eta^i) + \zeta^i \right).$$

However, since $\Delta_{\mathcal{C}_{j_{i^*}}}$ is sampled uniformly at random and independently, and is unknown to the adversary, this can only happen with probability $1/p$.

In total, the probability that there is some $\delta_m^i \neq 0$ is bounded by $(2 + \max_i T_i)/p$, which is negligible.

Correctness clearly holds if all errors are 0. Furthermore, we know from Lemma 1 that $\pi_{\textsf{eff-reshare-dm}}$ is simulatable by random values. Also, the simulator knows the universal hash keys that honest parties use to compute the hashes on the opened values (which it also knows), thus it can simulate these hashes itself. Furthermore, each β^j is sampled uniformly at random in $\mathcal{F}_{\textsf{prep}}$, so the simulator can also simulate these, and the corresponding MACs by using the MAC keys obtained from $\mathcal{F}_{\textsf{prep}}$. Finally, the simulator can easily emulate $\mathcal{F}_{\textsf{commit}}$ for the commitments. □

In our protocol, each committee $\mathcal{C}_i$ has their own MAC key $\Delta_{\mathcal{C}_i}$. Thus, when some state $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$ of the computation is reshared from one committee to the next, we also need to somehow "switch" the key of its MAC to that of the new committee. Le Mans also needs to deal with this issue. To do so, they take advantage of $\pi_{\textsf{convert}}$ to obtain random sharings $[\![t]\!]_{\Delta_i}^{\mathcal{C}_i}, [\![t]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$ of the same value, authenticated under *both* committees' MAC keys. This is sufficient for them, as they can then mask $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$ with t, reconstruct $x + t$, and then use their two authenticated sharings of t, $[\![t]\!]_{\Delta_i}^{\mathcal{C}_i}, [\![t]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$ (including some relationship between their MACs), to create unmasked shares of x that are indeed authenticated under Δ_{i+1}. Note, however, that they use reconstruction of $x + t$ from one committee to the next, which has quadratic communication.

We instead use the "king idea" to reconstruct $(x + t)$ across two committees, i.e., from $\mathcal{C}_i$ to $\mathcal{C}_{i+2}$. However, since $\pi_{\textsf{convert}}$ can only obtain authenticated random sharings of the same random value between *consecutive* committees, $\pi_{\textsf{eff-key-switch}}$ below switches $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$, shared and authenticated among $\mathcal{C}_i$, to $[\![x]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+2}}$, shared amongst $\mathcal{C}_{i+2}$, but authenticated using $\mathcal{C}_{i+1}$'s key.

Procedure 5: $\pi_{\textsf{eff-key-switch}}$

Usage: Transform a SPDZ sharing $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$ held by $\mathcal{C}_i$ and MAC'd under $\mathcal{C}_i$'s key to a SPDZ sharing $[\![x]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+2}}$ held by $\mathcal{C}_{i+2}$ and MAC'd under $\mathcal{C}_{i+1}$'s key.

1. Let $[\![x]\!]_{\Delta_i}^{\mathcal{C}_i}$ be the shares of the input authenticated value x. All parties in $\mathcal{C}_i$ agree on a special party $P_{\textsf{king}}$ in $\mathcal{C}_{i+1}$.
2. $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ invoke $\pi_{\textsf{get-combined-prep}}$ with $(\textsf{rand}, \mathcal{C}_i, \mathcal{C}_i \cup \mathcal{C}_{i+1})$ to receive $\langle t \rangle^{\mathcal{C}_i, \mathcal{C}_i \cup \mathcal{C}_{i+1}}$.
3. $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ then invoke $\pi_{\textsf{convert}}$ on $\langle t \rangle^{\mathcal{C}_i, \mathcal{C}_i \cup \mathcal{C}_{i+1}}$ to form $[\![t]\!]_{\Delta_i}^{\mathcal{C}_i}$ and $[\![t]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$.
4. Parties in $\mathcal{C}_i$ in parallel: (i) invoke $\pi_{\textsf{eff-reshare-dm}}$ on input $[x]^{\mathcal{C}_i}$; and (ii) open shares of $[\![(x+t)]\!]_{\Delta_i}^{\mathcal{C}_i}$ to $P_{\textsf{king}}$ in $\mathcal{C}_{i+1}$.
5. While $P_{\textsf{king}}$ of $\mathcal{C}_{i+1}$ distributes opened value $(x+t)$ to all parties in $\mathcal{C}_{i+2}$, all parties in $\mathcal{C}_{i+1}$ invoke $\pi_{\textsf{eff-reshare-dm}}$ on input $[x]^{\mathcal{C}_{i+1}}$, $\left[\Delta_{\mathcal{C}_{i+1}}\right]^{\mathcal{C}_{i+1}}$, and $\left[\Delta_{\mathcal{C}_{i+1}} \cdot t\right]^{\mathcal{C}_{i+1}}$.
6. Finally, each party P_j in $\mathcal{C}_{i+2}$ locally computes its share of the MAC $\left[\Delta_{\mathcal{C}_{i+1}} \cdot x\right]^{\mathcal{C}_{i+2}}$ as $\left[\Delta_{\mathcal{C}_{i+1}}\right]^{\mathcal{C}_{i+2}} \cdot (x+t) - \left[\Delta_{\mathcal{C}_{i+1}} \cdot t\right]^{\mathcal{C}_{i+2}}$. $\mathcal{C}_{i+2}$ outputs $([x]^{\mathcal{C}_{i+2}}, \left[\Delta_{\mathcal{C}_{i+1}} \cdot x\right]^{\mathcal{C}_{i+2}}, \left[\Delta_{\mathcal{C}_{i+1}}\right]^{\mathcal{C}_{i+2}})$.

Lemma 4. *Procedure* $\pi_{\textsf{eff-key-switch}}$*'s transcript is simulatable by random values.*

Proof. First, from Lemmas 1 and 2, we know that all values communicated to $\mathcal{A}$ by $\pi_{\textsf{convert}}$ and $\pi_{\textsf{eff-reshare-dm}}$ are simulatable by random values. Furthermore, from $\mathcal{F}_{\textsf{prep}}$, we know that $[\![t]\!]_{\Delta_i}^{\mathcal{C}_i}$ is a random sharing of a random value. Thus, also the openings $(x+t)$ can be simulated with random values. □

We leverage $\pi_{\textsf{eff-key-switch}}$ in the following procedure, $\pi_{\textsf{get-x-comm-shrs}}$, to obtain SPDZ sharings of the same random value amongst *both* committees $\mathcal{C}_i$ and $\mathcal{C}_{i+3}$. Committee $\mathcal{C}_i$'s sharing will be authenticated under its own key, while committee $\mathcal{C}_{i+3}$'s sharing will be authenticated under $\mathcal{C}_{i+2}$'s key.

Procedure 6: $\pi_{\textsf{get-x-comm-shrs}}$

Usage: On input BeDOZa sharing $\langle r \rangle^{\mathcal{C}_i, \mathcal{C}_i \cup \mathcal{C}_{i+1}}$, output $[\![r]\!]_{\Delta_i}^{\mathcal{C}_i}$ to $\mathcal{C}_i$ and $[\![r]\!]_{\Delta_{i+2}}^{\mathcal{C}_{i+3}}$ to $\mathcal{C}_{i+3}$.

1. $\mathcal{C}_i$ and $\mathcal{C}_{i+1}$ invoke $\pi_{\textsf{convert}}$ on $\langle r \rangle^{\mathcal{C}_i, \mathcal{C}_i \cup \mathcal{C}_{i+1}}$ to obtain $[\![r]\!]_{\Delta_i}^{\mathcal{C}_i}$ and $[\![r]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$.
2. $\mathcal{C}_i$ outputs $[\![r]\!]_{\Delta_i}^{\mathcal{C}_i}$.
3. $\mathcal{C}_{i+1}$ then invokes $\pi_{\textsf{eff-key-switch}}$ on $[\![r]\!]_{\Delta_{i+1}}^{\mathcal{C}_{i+1}}$ with $\mathcal{C}_{i+3}$ (through $\mathcal{C}_{i+2}$), who outputs $[\![r]\!]_{\Delta_{i+2}}^{\mathcal{C}_{i+3}}$.

Lemma 5. *Procedure* $\pi_{\textsf{get-x-comm-shrs}}$*'s transcript is simulatable by random values.*

Proof. Since only $\pi_{\textsf{convert}}$ and $\pi_{\textsf{eff-key-switch}}$ are invoked, we know from Lemmas 2 and 4 that the transcript is simulatable by random values. □

4.4 Secure Multiplication and Verification

Finally, before we present the complete protocol, we present Procedure $\pi_{\textsf{mult-dm}}$ below which enables a given committee to make progress on the computation by securely processing multiplication gates. As in the honest majority protocol of Section 5, this procedure makes use of multiplication triples to reduce the task of securely multiplying two shared values to that of reconstructing two secrets. Once again, we use the "king idea" to achieve reconstruction with linear communication.

As in the honest majority case, we have two issues to deal with. First multiplication triples require different committees to have access to the same triple. This is *in part* achieved by making use of the efficient resharing procedure $\pi_{\textsf{eff-reshare-dm}}$. Also, a given committee can once again only obtain a *partially* authenticated multiplication triple $(\llbracket a \rrbracket_{\Delta_i}^{\mathcal{C}_i}, \llbracket b \rrbracket_{\Delta_i}^{\mathcal{C}_i}, [c]^{\mathcal{C}_i})$ from $\mathcal{F}_{\textsf{prep}}$, where the c part is not authenticated (and in fact might have some error). Using the same idea from [RS22] and Section 5, we authenticate the c part "on the fly" using a random, authenticated sharing $\llbracket v \rrbracket_{\Delta_i}^{\mathcal{C}_i}$.

However, since our $\pi_{\textsf{eff-key-switch}}$ procedure when resharing a value from $\mathcal{C}_i$ to $\mathcal{C}_{i+2}$ only switches the MAC key to that of $\mathcal{C}_{i+1}$, we need to take some extra care to ensure that the MAC does not "fall behind". In particular, $\pi_{\textsf{mult-dm}}$ below maintains the invariant that the inputs to multiplication gates at some circuit level which $\mathcal{C}_i$ computes are MAC'd under $\mathcal{C}_{i-2}$'s key. At a high-level, this is accomplished by combining the above "on the fly" authentication of triples obtained from $\mathcal{F}_{\textsf{prep}}$ by $\mathcal{C}_{i-2}$, so that the triple is MAC'd under the same key as the inputs, with the use of $\pi_{\textsf{get-x-comm-shrs}}$ to transfer this triple to $\mathcal{C}_{i+1}$ (under $\mathcal{C}_i$'s key). $\mathcal{C}_{i+1}$ can then reshare this triple to $\mathcal{C}_{i+2}$, who can then use the usual multiplication triple technique to compute the output of the multiplication gate, MAC'd under $\mathcal{C}_i$'s key, thus maintaining the invariant.

Unfortunately, we are not done yet. Committee $\mathcal{C}_i$ can only obtain outputs that are MAC'd under $\mathcal{C}_{i-1}$'s key when $\mathcal{C}_{i-2}$ tries to use $\pi_{\textsf{eff-key-switch}}$ on its sharings. Thus, in order to maintain the invariant above, we need to wait until $\mathcal{C}_{i+1}$ to authenticate the c part of multiplication triples. However, this would allow the adversary to add errors dependent on x and y to c, since $(x+a)$ and $(y+b)$ are opened from $\mathcal{C}_i$ to the king of $\mathcal{C}_{i+1}$, leading to a selective failure attack. To solve this, the value $(c+v)$ needed to authenticate c is first opened to the parties of $\mathcal{C}_i$ using the efficient "king idea", then *only* P_1 of $\mathcal{C}_i$ sends the opened $(c+v)$ to the parties of $\mathcal{C}_{i+1}$, for efficiency. To ensure that P_1 does not cheat, the rest of the parties of $\mathcal{C}_i$ send hashes of the opened $(c+v)$'s for *all* of the multiplication gates at this level to the parties of $\mathcal{C}_{i+1}$. Although this introduces $\Omega(n^2)$ overhead, as long as if the width of the circuit is $\Omega(n)$, overall $O(n \cdot |C|)$ communication is still achieved. Details are as follows.

Procedure 7: $\pi_{\textsf{mult-dm}}$

Usage: Multiply $[\![x]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ and $[\![y]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ held by $\mathcal{C}_i$ and MAC'd under $\mathcal{C}_{i-2}$'s key so that $\mathcal{C}_{i+2}$ outputs $[\![x \cdot y]\!]^{\mathcal{C}_{i+2}}_{\Delta_i}$ MAC'd under $\mathcal{C}_i$'s key.

1. All parties in $\mathcal{C}_{i-2}$ first agree on a special party $P_{\textsf{king}}$ in $\mathcal{C}_{i-1}$.
2. Then $\mathcal{C}_{i-2}$ and $\mathcal{C}_{i-1}$ invoke $\pi_{\textsf{get-combined-prep}}$ on input $(\textsf{trip}, \mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1})$ to get partially authenticated BeDOZa triple $(\langle a \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}, \langle b \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}, [c]^{\mathcal{C}_{i-2}})$ and $(\textsf{rand}, \mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1})$ to get $\langle v \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}$. They also locally compute $[c+v]^{\mathcal{C}_{i-2}}$.
3. Then $\mathcal{C}_{i-2}$ invokes $\pi_{\textsf{get-x-comm-shrs}}$ on $\langle a \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}, \langle b \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}, \langle v \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}$ (with $\mathcal{C}_{i+1}$) to get SPDZ sharings $[\![a]\!]^{\mathcal{C}_{i-2}}_{\Delta_{i-2}}, [\![b]\!]^{\mathcal{C}_{i-2}}_{\Delta_{i-2}}$ ($[\![v]\!]^{\mathcal{C}_{i-2}}_{\Delta_{i-2}}$ is not needed).
4. Next, parties in $\mathcal{C}_{i-2}$ in parallel invoke $\pi_{\textsf{eff-reshare-dm}}$ on input $[\![a]\!]^{\mathcal{C}_{i-2}}_{\Delta_{i-2}}, [\![b]\!]^{\mathcal{C}_{i-2}}_{\Delta_{i-2}}$ and open shares of $[c+v]^{\mathcal{C}_{i-2}}$ to $P_{\textsf{king}}$ in $\mathcal{C}_{i-1}$.
5. While $P_{\textsf{king}}$ distributes opened $(c+v)$ to the parties of $\mathcal{C}_i$, all parties in $\mathcal{C}_{i-1}$ then invoke $\pi_{\textsf{eff-reshare-dm}}$ on input $[\![a]\!]^{\mathcal{C}_{i-1}}_{\Delta_{i-2}}, [\![b]\!]^{\mathcal{C}_{i-1}}_{\Delta_{i-2}}$.
6. Now, parties in $\mathcal{C}_i$ agree on a special party $P'_{\textsf{king}}$ in $\mathcal{C}_{i+1}$ and then compute $[\![x+a]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = [\![x]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} + [\![a]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$, and $[\![y+b]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = [\![y]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} + [\![b]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$.
7. Each $P_j \in \mathcal{C}_i$ also interprets the next pairwise shared random values from $\mathcal{F}_{\textsf{prep}}$, $\{s_{j,l}\}_{l \in \mathcal{C}_{i+1}}$ as keys to a universal hash family $\mathcal{H} = \{\mathcal{H}_s : \mathbb{F}_p^M \to \mathbb{F}_p\}$ and computes $h_{j,l} = \mathcal{H}_{s_{j,l}}(\{(c+v)_m\}_{m \in [T_i]})$ for each $P_l \in \mathcal{C}_{i+1}$, on input the T_i values $(c+v)_m$ used for the T_i multiplications computed by this committee.
8. In parallel: (i) each P_j in $\mathcal{C}_i$ invokes $\pi_{\textsf{eff-reshare-dm}}$ on input $[\Delta_i]^{\mathcal{C}_i}$, opens their share of $[\![x+a]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}, [\![y+b]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ to $P'_{\textsf{king}}$ in $\mathcal{C}_{i+1}$, and sends to each $P_l \in \mathcal{C}_{i+1}$ the computed hash value $h_{j,l}$; while (ii) only $P_1 \in \mathcal{C}_i$ sends $(c+v)$ to each $P_l \in \mathcal{C}_{i+1}$.
9. Each P_l in $\mathcal{C}_{i+1}$ first uses the pairwise shared random values from $\mathcal{F}_{\textsf{prep}}$, $\{s_{j,l}\}_{j \in \mathcal{C}_i}$ to compute $h'_{j,l} = \mathcal{H}_{s_{j,l}}(\{(c+v)_m\}_{m \in [T_i]})$, checks that each $h'_{j,l} = h_{j,l}$, and aborts if any check fails; else continues.
10. Then use $[\![v]\!]^{\mathcal{C}_{i+1}}_{\Delta_i}$ (obtained from $\pi_{\textsf{get-x-comm-shrs}}$) and opened $(c+v)$ to compute authenticated $[\![c]\!]^{\mathcal{C}_{i+1}}_{\Delta_i} = ((c+v) - [v]^{\mathcal{C}_{i+1}}, [\Delta_i]^{\mathcal{C}_{i+1}} \cdot (c+v) - [\Delta_i \cdot v]^{\mathcal{C}_{i+1}}, [\Delta_i]^{\mathcal{C}_{i+1}})$.
11. Then while $P'_{\textsf{king}}$ distributes opened $d = x + a$ and $e = y + b$ to the parties of $\mathcal{C}_{i+2}$, all parties in $\mathcal{C}_{i+1}$ invoke $\pi_{\textsf{eff-reshare-dm}}$ on input $[\![a]\!]^{\mathcal{C}_{i+1}}_{\Delta_i}, [\![b]\!]^{\mathcal{C}_{i+1}}_{\Delta_i}$ (also obtained from $\pi_{\textsf{get-x-comm-shrs}}$), and $[\![c]\!]^{\mathcal{C}_{i+1}}_{\Delta_i}$.
12. $\mathcal{C}_{i+2}$ finally locally compute $[\![x \cdot y]\!]^{\mathcal{C}_{i+2}}_{\Delta_i} = de - d\,[\![b]\!]^{\mathcal{C}_{i+2}}_{\Delta_i} - e\,[\![a]\!]^{\mathcal{C}_{i+2}}_{\Delta_i} + [\![c]\!]^{\mathcal{C}_{i+2}}_{\Delta_i}$.
13. Parties in $\mathcal{C}_{i+2}$ will also invoke $\pi_{\textsf{MAC-check-hm}}$ on input $(\textsf{update}, \{((x_m + a_m, [\Delta_{i-2} \cdot (x_m + a_m)]^{\mathcal{C}_{i+2}}), (y_m + b_m, [\Delta_{i-2} \cdot (y_m + b_m)]^{\mathcal{C}_{i+2}}))\}_{m \in [T]})$ corresponding to all of the openings of the above form they receive for the multiplication gates at this layer of the circuit.

Lemma 6. *Procedure* $\pi_{\textsf{mult-dm}}$*'s transcript is simulatable.*

Proof. First, we know that $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$ and $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ are simulatable by random values from Lemmas 5 and 1. Also, since a, b, v are uniformly random and unknown to the adversary by security of $\mathcal{F}_{\mathsf{prep}}$, openings $(c+v), (x+a), (y+b)$ are simulatable by random values. Also, the simulator knows the universal hash keys that honest parties use to compute the hashes on the opened values (which it also knows), thus it can simulate these hashes itself. $\square$

As with the honest majority protocol of Section 5 and Le Mans, we also need to account for the errors that can be introduced in the originally unauthenticated c parts of multiplication triples, that $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ will not catch. Indeed, as with Le Mans, $\mathcal{F}_{\mathsf{prep}}$ also allows the adversary to add errors to c of the form $\{a^j \cdot \delta_b^{j,l} + b^j \cdot \delta_a^{j,l}\}_{j \in \mathcal{H}_{\mathcal{C}_{i-2}}, l \in \mathcal{T}_{\mathcal{C}_{i-2}}}$, where a^j, b^j are the honest parties' shares of the a and b parts of the triple, and $\delta_b^{j,l}, \delta_a^{j,l}$ are chosen by the adversary. These errors could cause multiplications to be computed incorrectly. We thus use similar ideas to [RS22, CGG$^+$21], with ideas rooted in [CGH$^+$18], to compute a randomized version of the circuit that will be used to verify multiplications. The details are in Procedure $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ below. Note that in order to "keep up" with the invariant that we used in $\pi_{\mathsf{mult\text{-}dm}}$, we need to use similar techniques to ensure that the accumulators used in $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ are MAC'd under the same keys as the multiplication gate outputs.

Procedure 8: $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$

Usage: Each committee $\mathcal{C}_i$ that gets the output wires of the multiplication gates of some layer ℓ of the circuit incrementally updates a multiplication verification state $(\llbracket u' \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i}, \llbracket w' \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i})$, which the final committees at the end of the computation use to check that all multiplications throughout the protocol were performed correctly.

Init: Each Party P_i in committee $\mathcal{C}_5$ (the first to get output from $\pi_{\mathsf{mult\text{-}dm}}$) initially defines their shares of $\llbracket u' \rrbracket_{\Delta_5}^{\mathcal{C}_7}, \llbracket w' \rrbracket_{\Delta_5}^{\mathcal{C}_7}$ as $(u')^i = (w')^i = 0$ (same for the SPDZ MAC shares).

Update State: On input $(\mathsf{update}, \{(\llbracket z_m \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i}, \llbracket rz_m \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i}\}_{m \in [T]})$ from committee $\mathcal{C}_i$, where $\{(\llbracket z_m \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i}, \llbracket rz_m \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_i}\}_{m \in [T]}$ were the output wires of multiplication gates computed by $\mathcal{C}_i$:

1. $\mathcal{C}_{i-4}$ and $\mathcal{C}_{i-3}$ invoke $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$ with $(\mathsf{rand}, \mathcal{C}_{i-4}, \mathcal{C}_{i-4} \cup \mathcal{C}_{i-3})$ twice to get $\langle s \rangle^{\mathcal{C}_{i-4}, \mathcal{C}_{i-4} \cup \mathcal{C}_{i-3}}, \langle s' \rangle^{\mathcal{C}_{i-4}, \mathcal{C}_{i-4} \cup \mathcal{C}_{i-3}}$ and then $\mathcal{C}_{i-4}$ invokes $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$ on them (with $\mathcal{C}_{i-1}$) so that $\mathcal{C}_{i-4}$ gets $\llbracket s \rrbracket_{\Delta_{i-4}}^{\mathcal{C}_{i-4}}, \llbracket s' \rrbracket_{\Delta_{i-4}}^{\mathcal{C}_{i-4}}$.
2. Then $\mathcal{C}_{i-4}$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $\llbracket s \rrbracket_{\Delta_{i-4}}^{\mathcal{C}_{i-4}}, \llbracket s' \rrbracket_{\Delta_{i-4}}^{\mathcal{C}_{i-4}}$ and then $\mathcal{C}_{i-3}$ does the same.
3. Now, $\mathcal{C}_{i-2}$ first agrees on P_{king} in $\mathcal{C}_{i-1}$ then in parallel: (i) invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\Delta_{i-2}]^{\mathcal{C}_{i-2}}$; and (ii) opens shares of $\llbracket u+s \rrbracket_{\Delta_{i-4}}^{\mathcal{C}_{i-2}}, \llbracket w+s' \rrbracket_{\Delta_{i-4}}^{\mathcal{C}-2i}$ to P_{king}.
4. Committee $\mathcal{C}_{i-1}$ invokes $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$ with $(\mathsf{rand}, \mathcal{C}_{i-1}, \mathcal{C}_i)$ so that $P_j \in \mathcal{C}_{i-1}$ gets $(\alpha^j, \{M^{j,l}\}_{l \in \mathcal{C}_i})$ and $P_l \in \mathcal{C}_i$ gets $(\Delta^l, \{K^{l,j}\}_{j \in \mathcal{C}_{i-1}})$.
5. While P_{king} distributes opened $(u+s), (w+s')$ to the parties of $\mathcal{C}_i$, all parties in $\mathcal{C}_{i-1}$ invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $\llbracket s \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_{i-1}}, \llbracket s' \rrbracket_{\Delta_{i-2}}^{\mathcal{C}_{i-1}}$ (obtained from

$\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$) and send to each $P_l \in \mathcal{C}_i$ their share α^j and corresponding MAC for P_l, $M^{j,l}$.

6. Parties P_l in $\mathcal{C}_i$ then locally compute $[\![u]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = (u+s) - [\![s]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ and $[\![w]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = (w+s') - [\![s']\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$.
7. Then each $P_l \in \mathcal{C}_i$ locally checks that $M^{j,l} = \alpha^j \cdot \Delta^l + K^{l,j}$, for each $P_j \in \mathcal{C}_{i-1}$, and aborts if any fail. If not, let $\alpha = \sum_{j \in \mathcal{C}_{i-1}} \alpha^j$.
8. Finally, each $P_l \in \mathcal{C}_i$ locally computes $[\![u]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = [\![u]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} + \sum_{m=1}^{T} (\alpha)^m \cdot [\![rz_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ and $[\![w]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} = [\![w]\!]^{\mathcal{C}_i}_{\Delta_{i-2}} + \sum_{m=1}^{T} (\alpha)^m \cdot [\![z_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ (here $(\alpha)^m$ is the m-th power of α).

Check State: On input check from the clients $\mathcal{C}_{\mathsf{clnt}}$:

1. The clients $\mathcal{C}_\ell$ first open $[\![r]\!]^{\mathcal{C}_\ell}_{\Delta_1}$ and check its MAC by running both phases of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$.
2. Then they all open $([\![u]\!]^{\mathcal{C}_\ell}_{\Delta_{\ell-2}} - r \cdot [\![w]\!]^{\mathcal{C}_\ell}_{\Delta_{\ell-2}})$ and check its MAC by running both phases of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$. If the opened value is 0, output Accept; else Reject.

Lemma 7. *Procedure $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ is correct, i.e., it accepts if all multiplications are computed correctly. Moreover, it is sound, i.e., it rejects except with probability at most $(2+\max_i T_i)/p$ in case at least one multiplication is not correctly computed. Furthermore, the transcript of **Update State** is simulatable.*

Proof. For soundness, we consider all of the points in which the adversary can inject error, either when multiplying $[\![r]\!]^{\mathcal{C}_3}_{\Delta_1}$ with each input $[\![v_j]\!]^{\mathcal{C}_3}_{\Delta_1}$ or when a given committee $\mathcal{C}_i$ is updating $[\![u]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ and $[\![w]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ based on multiplication gate outputs it has received. First, note that with all-but-negligible probability, the additive error for some $[\![c]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ in a multiplication triple is independent of the opened $(x+a), (y+b)$ for that multiplication. This is because the (potentially incorrect) $(c+v)$ that is opened to some honest party $P_j \in \mathcal{C}_{i-2}$ before $(x+a), (y+b)$ are opened is the same as that received by an honest party $P_l \in \mathcal{C}_{i-1}$ with all-but-negligible probability. We know this because their shared universal hash key $s_{j,l}$ from $\mathcal{F}_{\mathsf{prep}}$ is uniformly random and unknown to the adversary. So, since $\mathcal{H}$ is a universal hash family, it holds that if P_j gets $\{(c+v)_m\}_{m\in[T_i]}$ and P_l gets a different $\{(c+v)'_m\}_{m\in[T_i]}$,

$$\Pr[\mathcal{H}_{s_{j,l}}(\{(c+v)_m\}_{m\in[T_i]}) = h_{j,l} = h'_{j,l} = \mathcal{H}_{s_{j,l}}(\{(c+v)'_m\}_{m\in[T_i]})] \leq 1/p.$$

So, since $(c+v)$ is opened before $(x+a)$ and $(y+b)$ and v is uniformly random and independent of them, the additive error for $[\![c]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ must be independent of them (along with the challenge α which is opened even later).

The adversary can thus only inject the following kind of errors:

1. The $[\![c_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ part of the m-th multiplication triple $([\![a_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}, [\![b_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}, [\![c_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}})$ used to compute $[\![x_m y_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ may have errors $\{a^j_m \cdot \delta^{j,l}_{b_m} + b^j_m \cdot \delta^{j,l}_{a_m}\}_{j \in \mathcal{H}_{\mathcal{C}_{i-4}}, l \in \mathcal{T}_{\mathcal{C}_{i-4}}} +$

δ_{c_m}. The errors in the brackets come from adversarial action in $\mathcal{F}_{\mathsf{prep}}$ while the δ_c comes from the fact that c is not authenticated, so the adversary can insert more error when resharing/authenticating them. Call this error $\varepsilon_{i,m}$. (Note this is the only source of error when computing some $[\![rv_j]\!]^{\mathcal{C}_5}_{\Delta_3}$.)

2. Additionally, the $[\![c'_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ part of the multiplication triple ($[\![a'_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$, $[\![b'_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$, $[\![c'_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$) used to compute $[\![rx_m y_m]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ may have the same kind of errors $\{(a'_m)^j \cdot (\delta')^{j,l}_{b_m} + (b'_m)^j \cdot (\delta')^{j,l}_{a_m}\}_{j \in \mathcal{H}_{\mathcal{C}_{i-4}}, l \in \mathcal{T}_{\mathcal{C}_{i-4}}} + \delta_{c_m}$. Call this error $\varepsilon'_{i,m}$.
3. Also, we must consider the accumulated error $\eta_{i,m}$ on the randomized value rx_m from previous gates.

So, at the end of the computation (after d multiplications), if the check passes we will have:

$$0 = [\![u]\!]^{\mathcal{C}_\ell}_{\Delta_{\ell-2}} - r \cdot [\![w]\!]^{\mathcal{C}_\ell}_{\Delta_{\ell-2}}$$

$$= \sum_{j=1}^{T_0} \alpha_0^j \cdot (rv_j + \varepsilon_{0,j}) + \sum_{i=1}^{d} \sum_{m=1}^{T_i} \alpha_i^m \cdot ((rx_m + \eta_{i,m}) \cdot y_m + \varepsilon'_{i,m}) -$$

$$r \cdot \left(\sum_{i=1}^{d} \sum_{m=1}^{T_i} \alpha_i^m \cdot (x_m \cdot y_m + \varepsilon_{i,m}) + \sum_{j=1}^{M} \alpha_0^j \cdot v_j \right)$$

$$= \sum_{i=1}^{d} \sum_{m=1}^{T_i} \alpha_i^m (\eta_{i,m} \cdot y_m + \varepsilon'_{i,m} - r \cdot \varepsilon_{i,m}) + \sum_{j=1}^{M} \alpha_0^j \cdot \varepsilon_{0,j}.$$

Now, note that the corrupt parties of committee $\mathcal{C}_{i-1}$ cannot forge their share α^j of α to any of the honest parties of $\mathcal{C}_i$ except with probability $1/p$, by the security of the information-theoretic MAC provided by $\mathcal{F}_{\mathsf{prep}}$. Thus, the reconstructed challenge α must indeed be uniformly random and independent of all other values. We analyze the two following cases:

Case 1: *There is some j such that $\varepsilon_{0,j} \neq 0$.* In this case, it is clear that the above polynomial is non-zero. Therefore, since each α_i is unknown to the adversary and sampled uniformly at random and independently of all other values, the Schwartz-Zippel Lemma tells us that the evaluation of this polynomial on these α_i equals 0 with probability at most $\max_i T_i/p$.

Case 2: *For all j, $\varepsilon_{0,j} = 0$.* Let layer i^* be the first in which the adversary injected error into a multiplication; i.e., $\varepsilon'_{i^*,m} \neq 0$ and/or $\varepsilon_{i^*,m} \neq 0$ for some m. Note that since this is the first such layer, it must be that $\eta_{i,m} = 0$ for all i^*, m. So,

$$0 = \sum_{i=1}^{d} \sum_{m=1}^{T_i} \alpha_i^m (\eta_i \cdot y_m + \varepsilon'_i - r \cdot \varepsilon_i)$$

$$= \sum_{m=1}^{T_{i^*}} \alpha_{i^*}^m (\varepsilon'_{i^*,m} - r \cdot \varepsilon_{i^*,m}) + \sum_{i \in [d] \setminus \{i^*\}} \sum_{m=1}^{T_i} \alpha_i^m (\eta_i \cdot y_m + \varepsilon'_i - r \cdot \varepsilon_i).$$

First, for the given m where the adversary injected error, $(\varepsilon'_{i^*,m} - r \cdot \varepsilon_{i^*,m}) = 0$ can only happen with probability $1/p$ since r is unknown to the adversary and sampled uniformly and independently of all other values. Now, if $(\varepsilon'_{i^*,m} - r \cdot \varepsilon_{i^*,m}) \neq 0$, then the above polynomial is non-zero. Since each α_i is unknown to the adversary and sampled uniformly at random and independently of all other values, the Schwartz-Zippel Lemma tells us that the evaluation of this polynomial on these α_i equals 0 with probability at most $\max_i T_i/p$.

Thus, the total probability that the adversary can inject some error is upper bounded by $(2 + \max_i T_i)/p$.

Correctness clearly holds if all errors are 0. Finally, From Lemma 5 we know that $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$ is simulatable, and from Lemma 1, we know that $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ is simulatable. Also, since s, s' are uniformly random and unknown to the adversary by the security of $\mathcal{F}_{\mathsf{prep}}$, openings $(u + s), (w + s')$ are simulatable by random values. Additionally, each α^j is sampled uniformly at random in $\mathcal{F}_{\mathsf{prep}}$, so the simulator can simulate these, and their corresponding MACs by using the MAC keys obtained from $\mathcal{F}_{\mathsf{prep}}$. Finally, from Lemma 3, we know that $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$'s transcript is indeed simulatable. □

4.5 Dishonest Majority Protocol

With all of the previous tools in place, we can finally present our full-fledged actively secure, dishonest majority MPC protocol in the fluid setting, achieving linear communication complexity and maximal fluidity. The clients first use $\pi_{\mathsf{eff\text{-}key\text{-}switch}}$ to securely transfer authenticated versions of their inputs to $\mathcal{C}_2$. The committees then proceed to compute both the regular and randomized version of the circuit on the authenticated inputs, using $\pi_{\mathsf{mult\text{-}dm}}$ as well as addition and identity gate procedures that work similarly using the same "mask, open to king, and unmask" paradigm along with some local computation. Addition and identity gates also need to preserve the invariant on MACs discussed in Section 4.4. The committees also make sure to update the accumulators of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ and $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ along the way with each opening and multiplication, respectively. We note that, unlike the honest majority protocol, the outputs of the final circuit layer will be shared by the clients themselves, i.e., $\mathcal{C}_{\mathsf{clnt}} = \mathcal{C}_\ell$. Once all circuit layers have been computed, the clients invoke the **Check State** phases of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ and $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$, then reconstruct the outputs. We note that, as is remarked in the protocols of [RS22, CGG+21], if the clients indeed have access to a broadcast channel in the last round of the protocol, or implement a broadcast over their point-to-point channels, then security with unanimous abort is achieved by having the clients broadcast "abort", if their check on their output fails.

Protocol 9: $\Pi_{\mathsf{main\text{-}dm}}$

Preprocessing Phase: All parties $P_i \in \mathcal{U}$ invoke $\mathcal{F}_{\mathsf{prep}}$ to receive their share of the global MAC key Δ^i, along with enough pairwise random sharings, multiplication triples, and sharings of 0.

Input Phase: To form a SPDZ sharing of an input x_i possessed by $P_i \in \mathcal{C}_{\mathsf{clnt}}$:

1. P_i invokes $\pi_{\mathsf{eff\text{-}key\text{-}switch}}$ on $(x_i, \Delta^i \cdot x_i, \Delta^i)$ with $\mathcal{C}_2$ (through $\mathcal{C}_1$).
2. $\mathcal{C}_1$ invokes $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$ with $(\mathsf{rand}, \mathcal{C}_1, \mathcal{C}_1)$ to get $\langle r \rangle^{\mathcal{C}_1, \mathcal{C}_1}$ then invokes π_{convert} on it to get SPDZ sharing $[\![r]\!]_{\Delta_1}^{\mathcal{C}_1}$, and finally $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on this.
3. $\mathcal{C}_2$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\![x]\!]_{\Delta_1}^{\mathcal{C}_2}$ (from $\pi_{\mathsf{eff\text{-}key\text{-}switch}}$ above) and $[\![r]\!]_{\Delta_1}^{\mathcal{C}_2}$.
4. Finally, $\mathcal{C}_3$ invokes $\pi_{\mathsf{mult\text{-}dm}}$ on $[\![x]\!]_{\Delta_1}^{\mathcal{C}_3}$ and $[\![r]\!]_{\Delta_1}^{\mathcal{C}_3}$, as well as the identity gate procedure (below) on $[\![x]\!]_{\Delta_1}^{\mathcal{C}_3}$ so that $\mathcal{C}_5$ gets $[\![x]\!]_{\Delta_3}^{\mathcal{C}_5}$ and $[\![x \cdot r]\!]_{\Delta_3}^{\mathcal{C}_5}$.
5. Finally, $\mathcal{C}_5$ invokes $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ on input $(\mathsf{update}, \{([\![x_i]\!]_{\Delta_3}^{\mathcal{C}_5}, [\![rx_i]\!]_{\Delta_3}^{\mathcal{C}_5})\}_{i \in [|\mathcal{C}_{\mathsf{clnt}}|]})$, corresponding to each input.

Execution Phase:

1. Each committee $\mathcal{C}_i$ of the execution phase first invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\![r]\!]_{\Delta_1}^{\mathcal{C}_i}$.
2. In parallel, every other committee (with the help of the others) will compute the gates at each layer of the circuit as below:

Addition: To perform addition on $[\![x]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ and $[\![y]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ (and identically for $[\![rx]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ and $[\![ry]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$):

1. $\mathcal{C}_{i-2}$ and $\mathcal{C}_{i-1}$ invoke $\pi_{\mathsf{get\text{-}combined\text{-}prep}}$ with $(\mathsf{rand}, \mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1})$ to get $\langle s \rangle^{\mathcal{C}_{i-2}, \mathcal{C}_{i-2} \cup \mathcal{C}_{i-1}}$ and then invokes $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$ on it (with $\mathcal{C}_{i+1}$) to get SDPZ sharing $[\![s]\!]_{\Delta_{i-2}}^{\mathcal{C}_{i-2}}$.
2. Then $\mathcal{C}_{i-2}$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\![s]\!]_{\Delta_{i-2}}^{\mathcal{C}_{i-2}}$ and $\mathcal{C}_{i-1}$ does the same.
3. Now, $\mathcal{C}_i$ first agrees on P_{king} in $\mathcal{C}_{i+1}$ then locally computes $[\![x+y]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$.
4. Then $\mathcal{C}_i$ in parallel: (i) invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\Delta_i]^{\mathcal{C}_i}$; and (ii) opens shares of $[\![x+y+s]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ to P_{king}.
5. While P_{king} distributes opened $(x+y+s)$ to the parties of $\mathcal{C}_{i+2}$, all parties of $\mathcal{C}_{i+1}$ invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ on $[\![s]\!]_{\Delta_i}^{\mathcal{C}_{i+1}}$ (obtained through $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$).
6. Parties in $\mathcal{C}_{i+2}$ finally locally compute $[\![x+y]\!]_{\Delta_i}^{\mathcal{C}_{i+2}} = (x+y+s) - [\![s]\!]_{\Delta_i}^{\mathcal{C}_{i+2}}$.
7. Parties in $\mathcal{C}_{i+2}$ will also invoke $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ on input $(\mathsf{update}, \{(x_m + y_m + s_m, [\Delta_{i-2} \cdot (x_m + y_m + s_m)]^{\mathcal{C}_{i+2}})\}_{m \in [T]})$ corresponding to all of the openings of the above form they receive for the addition gates at this layer of the circuit.[a]

Identity Gates: $\mathcal{C}_i$ forwards $[\![x]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}, [\![rx]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ to $\mathcal{C}_{i+2}$ (so that they are MAC'd under Δ_i) in a similar fashion as addition above.

Multiplication: To multiply $[\![x]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ and $[\![y]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$, invoke $\pi_{\mathsf{mult\text{-}dm}}$ on them (and identically for $[\![rx]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$ and $[\![y]\!]_{\Delta_{i-2}}^{\mathcal{C}_i}$). Then invoke $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ on input $(\mathsf{update}, \{([\![x_m y_m]\!]_{\Delta_i}^{\mathcal{C}_{i+2}}, [\![(rx)_m y_m]\!]_{\Delta_i}^{\mathcal{C}_{i+2}})\}_{m \in [T_i]})$, corresponding to each multiplication performed at this layer of the circuit.

Output Phase:

1. Clients in $\mathcal{C}_{\mathsf{clnt}}$ first invoke $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ on check. If it outputs Reject, then abort; else, continue. This takes 2 rounds.

2. Clients in $\mathcal{C}_{\mathsf{clnt}}$ then invoke $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ on check. If it outputs Reject, then abort; else, continue. This takes 6 more rounds.
3. Clients finally open each output wire $[\![z]\!]^{\mathcal{C}_\ell}_{\Delta_{\ell-2}}$ and check their MACs by running both phases of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$. If it outputs Reject, then abort; else, output each z. This takes 3 more rounds.

[a] This invocation can be combined with that of the multiplication gates for this circuit layer.

Theorem 4. *Let $\mathcal{A}$ be an* R*-adaptive adversary in $\Pi_{\mathsf{main\text{-}dm}}$. Then the protocol UC-securely computes $\mathcal{F}_{\mathsf{DABB}}$ in the presence of $\mathcal{A}$ in the $(\mathcal{F}_{\mathsf{prep}}, \mathcal{F}_{\mathsf{commit}})$-hybrid model.*

Proof. We construct a Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}_{\mathsf{DABB}}$. It internally emulates the functionalities $\mathcal{F}_{\mathsf{prep}}$ and $\mathcal{F}_{\mathsf{commit}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment $\mathcal{Z}$. It keeps track of the current committee via inputs $(\mathsf{Init}, \mathcal{C})$ and $(\mathsf{Next\text{-}Committee}, \mathcal{C})$ from $\mathcal{F}_{\mathsf{DABB}}$ (and therefore in which committees to simulate corresponding communication for circuit gates). The simulator uses bad, initially set to 0, to detect any bad behavior from $\mathcal{A}$. If so, it sets $\mathsf{bad} = 1$. The simulation proceeds as follows:

Init: On input $(\mathsf{Init}, \mathcal{C})$, keep track of $\mathcal{A}$'s inputs to $\mathcal{F}_{\mathsf{prep}}$, including any additive errors $\delta_a \neq 0$ or $\delta_b \neq 0$ for any pairwise multiplication triples (this might not be an issue yet, as long as if the eventual additive error on any c of any triple ends up as 0; see below).

Input: On input $(\mathsf{Input}, \mathsf{id}_x)$ (for both honest and adversarial inputs), simulate $\pi_{\mathsf{eff\text{-}key\text{-}switch}}$ as in Lemma 4, π_{convert} as in Lemma 2, $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ as in Lemma 1, and $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ as in Lemma 3. If $\mathcal{A}$ cheats when sending some universal hash value during $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$, abort. If $\mathcal{A}$ cheats either when resharing or opening a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$. Also, simulate the multiplication as below.

Addition (and similarly for identity gates): On input $(\mathsf{Add}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$, simulate $\pi_{\mathsf{get\text{-}x\text{-}comm\text{-}shrs}}$ as in Lemma 5, $\pi_{\mathsf{eff\text{-}reshare\text{-}dm}}$ as in Lemma 1, and $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ as in Lemma 3. Additionally, simulate the opening of $[\![x + y + s]\!]^{\mathcal{C}_i}_{\Delta_{i-2}}$ with random values. Since s is uniformly random and unknown to $\mathcal{A}$, this is a perfect simulation. If $\mathcal{A}$ cheats when sending some universal hash value during $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$, abort. If $\mathcal{A}$ cheats either when resharing or opening a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$.

Multiplication: On input $(\mathsf{Mult}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$, simulate $\pi_{\mathsf{mult\text{-}dm}}$ as in Lemma 6 and $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$ as in Lemma 7. If $\mathcal{A}$ cheats when sending some universal hash value during $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ or $\pi_{\mathsf{mult\text{-}dm}}$, abort. If for any c part of a multiplication triple, the additive error $\delta_c + \sum_{j \in \mathcal{H}_{\mathcal{C}_{i-2}}, l \in \mathcal{T}_{\mathcal{C}_{i-2}}} a^j \cdot \delta_b^{j,l} + b^j \cdot \delta_a^{j,l} \neq 0$, set $\mathsf{bad} = 1$.

Additionally, if $\mathcal{A}$ cheats either when resharing or opening a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$.

Output: If $\mathcal{S}$ ever set $\mathsf{bad} = 1$ because $\mathcal{A}$ cheated when opening or resharing a value, $\mathcal{S}$ sends random values for σ on behalf of the honest parties, then aborts. Otherwise, $\mathcal{S}$ records $\{\sigma^i\}_{i \in \mathcal{T}_{\mathcal{C}_\ell}}$ sent to $\mathcal{F}_{\mathsf{commit}}$ by $\mathcal{A}$ in the check state phase of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$, samples random shares for the honest parties such that $\sum_{i \in \mathcal{C}_\ell} \sigma^i = 0$ and sends them to $\mathcal{A}$. If $\mathcal{A}$ cheats during the *check state* phase of $\pi_{\mathsf{MAC\text{-}check\text{-}dm}}$ (e.g., by committing to the wrong MAC check value σ^i), $\mathcal{S}$ aborts. In the *check state* phase of $\pi_{\mathsf{mult\text{-}verify\text{-}dm}}$, $\mathcal{S}$ sends random shares on behalf of the honest parties for the opening of r. If $\mathcal{A}$ cheats by opening the wrong values for r, $\mathcal{S}$ aborts after the MAC check (as above). If $\mathcal{S}$ ever set $\mathsf{bad} = 1$ because $\mathcal{A}$ added non-zero error to the c part of a multiplication triple, $\mathcal{S}$ sends random values on behalf of the honest parties for $(u - r \cdot w)$, then aborts. Otherwise, $\mathcal{S}$ records the values sent by $\mathcal{A}$ for $(u - r \cdot w)$, then samples shares such that $(u - r \cdot w) = 0$, and sends them to $\mathcal{A}$. If $\mathcal{A}$ cheats when opening $(u - r \cdot w)$, $\mathcal{S}$ aborts after the MAC check (as above).

Finally, $\mathcal{S}$ gets the outputs from $\mathcal{F}_{\mathsf{DABB}}$ and forwards it to $\mathcal{A}$. $\mathcal{S}$ then forwards whatever it receives from $\mathcal{A}$ back to $\mathcal{F}_{\mathsf{DABB}}$.

From all of the Lemmas, we have that the simulation is perfect up until the output phase. By Lemmas 3 and 7, $\mathcal{A}$ is only able to cheat in the real world with probability negligible in p. Thus, the distance between the real-world and the simulation is negligible in p. □

5 Honest Majority

We now turn to presenting our protocol for fluid MPC with linear communication complexity and maximal fluidity in the honest majority setting, where each committee contains at most a minority of corrupt parties. The outline of this section is the following. First, in Section 5.1, we present a major building block, Procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$, which enables a given committee holding a sharing of a random value to efficiently reshare this secret to the next committee. As in the two previous fluid protocols [RS22, CGG+21], we make use of a randomized version of the circuit that aims at detecting cheating in multiplication gates, and we also draw inspiration from [RS22] and make use of a MAC check that accounts for the correctness of the openings throughout the computation, which is crucial in our case to achieve linear communication complexity. This is discussed in Section 5.2. Then, in Section 5.3 we show how the parties make progress through the computation by processing multiplication gates. Finally, these pieces are put together in Section 5.4 to obtain our final protocol, $\mathit{\Pi}_{\mathsf{main\text{-}hm}}$, for honest majority MPC in the fluid model with linear communication complexity and maximal fluidity.

Notation and initial building blocks. We let $[x]_{t_i}^{\mathcal{C}_i}$ denote a Shamir secret-sharing of value x with degree-t_i among the parties of committee $\mathcal{C}_i$. A SPDZ sharing [DPSZ12] of a value x among the parties of committee $\mathcal{C}$, $[\![x]\!]^{\mathcal{C}}$ contains a vector of degree-$2t$ Shamir shares $[\![x]\!]^{\mathcal{C}} := ([x]_{2t}, [\Delta]_{2t}, [\Delta \cdot x]_{2t})$.

We now present some of the building blocks we will require for our final protocol. For our main honest majority protocol, we will require the following functionalities. These are fairly standard in the literature and implementing them in the fluid setting represents little challenge, using the randomness extraction ideas through Vandermonde matrices in [DN07]. Thus we omit their instantiations for brevity.

Functionality 4: $\mathcal{F}_{\mathsf{rand}}$

Functionality: Distribute degree-t_i sharings of random value r to $\mathcal{C}_i$.

1. $\mathcal{F}_{\mathsf{rand}}$ receives from the adversary shares $\{r_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$. $\mathcal{F}_{\mathsf{rand}}$ views these as the shares of the corrupted parties.
2. $\mathcal{F}_{\mathsf{rand}}$ randomly samples r, then based on r and the t_i shares $\{r_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$ of corrupted parties, $\mathcal{F}_{\mathsf{rand}}$ reconstructs the whole sharing $[r]^{\mathcal{C}_i}_{t_i}$.
3. Finally, $\mathcal{F}_{\mathsf{rand}}$ distributes the shares of $[r]^{\mathcal{C}_i}_{t_i}$ to the honest parties of $\mathcal{C}_i$.

Functionality 5: $\mathcal{F}_{\mathsf{coin}}$

Functionality: Sample a random coin $r \in \mathbb{F}_p$ to $\mathcal{C}_i$.

1. $\mathcal{F}_{\mathsf{coin}}$ samples a random field element r.
2. $\mathcal{F}_{\mathsf{coin}}$ sends r to the adversary and:
   - If the adversary replies continue, $\mathcal{F}_{\mathsf{coin}}$ sends r to the honest parties of $\mathcal{C}_i$.
   - If the adversary replies abort, $\mathcal{F}_{\mathsf{coin}}$ sends abort to the honest parties of $\mathcal{C}_i$.

Functionality 6: $\mathcal{F}_{\mathsf{double\text{-}rand}}$

Functionality: Distribute degree-t_i and degree-$2t_i$ sharings of the same random value r to $\mathcal{C}_i$.

1. $\mathcal{F}_{\mathsf{double\text{-}rand}}$ receives from the adversary two sets of shares $\{r_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$ and $\{r'_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$. $\mathcal{F}_{\mathsf{double\text{-}rand}}$ views the first set as the shares of the corrupted parties for the degree t_i-sharing, and the second set as the shares for the degree $2t_i$-sharing.
2. $\mathcal{F}_{\mathsf{double\text{-}rand}}$ randomly samples r and prepares the double sharings as follows.
   - For the degree-t_i sharing, based on the secret r and the t_i shares $\{r_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$ of corrupted parties, $\mathcal{F}_{\mathsf{double\text{-}rand}}$ reconstructs the whole sharing $[r]^{\mathcal{C}_i}_{t_i}$.
   - For the degree-$2t_i$ sharing, $\mathcal{F}_{\mathsf{double\text{-}rand}}$ randomly samples t_i elements as the shares of the first t_i honest parties. Based on the secret r, the t_i shares of the first t_i honest parties, and the t_i shares $\{r'_i\}_{i \in \mathcal{T}_{\mathcal{C}}}$ of the corrupted parties, $\mathcal{F}_{\mathsf{double\text{-}rand}}$ reconstructs the whole sharing $[r]^{\mathcal{C}_i}_{2t_i}$.

3. Finally, $\mathcal{F}_{\mathsf{double\text{-}rand}}$ distributes the shares of $([r]^{\mathcal{C}_i}_{t_i}, [r]^{\mathcal{C}_i}_{2t_i})$ to the honest parties of $\mathcal{C}_i$.

Functionality 7: $\mathcal{F}_{\mathsf{zero}}$

Functionality: Distribute degree $2t_i$ shares of $o = 0$ to $\mathcal{C}_i$.

1. $\mathcal{F}_{\mathsf{zero}}$ receives from the adversary the set of shares $\{r_i\}_{i\in\mathcal{T}_C}$.
2. $\mathcal{F}_{\mathsf{zero}}$ randomly samples t_i elements as the shares of the first t_i honest parties. Based on the secret $o = 0$, the t_i shares of the first t_i honest parties, and the t_i shares $\{r_i\}_{i\in\mathcal{T}_C}$ of the corrupted parties, $\mathcal{F}_{\mathsf{zero}}$ reconstructs the whole sharing $[o]^{\mathcal{C}_i}_{2t_i}$.
3. Finally, $\mathcal{F}_{\mathsf{zero}}$ distributes the shares of $[o]^{\mathcal{C}_i}_{2t_i}$ to the honest parties of $\mathcal{C}_i$.

As we will later accomplish in $\Pi_{\mathsf{main\text{-}hm}}$, each committee will have a degree-t_i and degree-$2t_i$ double sharing of the global MAC key Δ. Therefore we will assume that all procedures presented below that are invoked by $\mathcal{C}_i$ will implicitly take these sharings as input.

We rely on the following procedure that enables the parties in a given committee $\mathcal{C}_i$ to obtain authenticated sharings of a uniformly random value $[\![r]\!]^{\mathcal{C}_i}$, assuming Shamir sharings of the key $([\Delta]^{\mathcal{C}_i}_{t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i})$. This is described below. Observe that in the protocol the MAC sharings produced $[r\cdot\Delta]^{\mathcal{C}}_{t_i}$ are not uniformly random, but instead, they are a product $[r]^{\mathcal{C}_i}_{t_i}\cdot[\Delta]^{\mathcal{C}_i}_{t_i}$. These sharings will be randomized in the places we use them.

Procedure 10: $\pi_{\mathsf{get\text{-}rand\text{-}sharing}}$

Usage: Using double sharing $([\Delta]^{\mathcal{C}_i}_{t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i})$ of the global MAC key, $\mathcal{C}_i$ outputs a random SPDZ sharing $[\![r]\!]^{\mathcal{C}_i}$.

1. All parties in $\mathcal{C}_i$ invoke $\mathcal{F}_{\mathsf{double\text{-}rand}}$ to get random double sharing $([r]^{\mathcal{C}_i}_{t_i}, [r]^{\mathcal{C}_i}_{2t_i})$.
2. Parties in $\mathcal{C}_i$ then locally obtain and output authenticated sharing $[\![r]\!]^{\mathcal{C}_i} = ([r]^{\mathcal{C}_i}_{2t_i}, [r]^{\mathcal{C}_i}_{t_i}\cdot[\Delta]^{\mathcal{C}_i}_{t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i})$.

5.1 Efficient Resharing for Honest Majority

As we highlighted in Section 2, a fundamental reason why the protocol from [CGG$^+$21] does not achieve linear communication complexity stems from the fact that the hand-off procedure from one committee to the next one consists of every party resharing their share towards the next committee, which requires quadratic communication. In our work, we address this limitation by making use of Procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ below, which shows how to reshare a degree-$2t_i$ Shamir sharing from committee $\mathcal{C}_i$ to the next committee $\mathcal{C}_{i+1}$, while using only linear communication. The idea is in fact simple: assuming each committee has the same amount of parties n (the procedure below is more general), each party with index j in committee $\mathcal{C}_i$ will send (a re-randomized version of) their share to the party with index j in $\mathcal{C}_{i+1}$ directly. This is secure since the adversary learns in total at most $2t$ shares across the two committees, which is the degree of the polynomial used. As briefly mentioned above, the parties first re-randomize their shares using $\mathcal{F}_{\mathsf{zero}}$, which is done to prevent a new sharing from leaking the underlying secret when transmittted to the next committee.

Procedure 11: $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$

Usage: $\mathcal{C}_i$ reshares re-randomized $[r]^{\mathcal{C}_i}_{2t_i}$ to $\mathcal{C}_{i+1}$. Assume that the parties in $\mathcal{C}_i$ are indexed from 1 to n_i and those in $\mathcal{C}_{i+1}$ are indexed from n_i+1 to n_i+n_{i+1}.

1. Let $[r]^{\mathcal{C}_i}_{2t_i}$ be the input shares.
2. $\mathcal{C}_i$ invokes $\mathcal{F}_{\mathsf{zero}}$ and receives a sharing of $o=0$, $[o]^{\mathcal{C}_i}_{2t_i}$.
3. All parties locally compute $[r']^{\mathcal{C}_i}_{2t_i} = [r]^{i}_{2t_i} + [o]^{i}_{2t_i}$, for $r' = r + 0 = r$.
4. Finally:
   - If $n_i < n_{i+1}$: Let $d = n_{i+1}/n_i$ (assuming $n_i | n_{i+1}$ for simplicity). Each $P_j \in \mathcal{C}_i$ samples $d-1$ random values r_l, sets $r_{j\cdot d} = (r')^j - \sum_{l=1}^{d-1} r_l$, where $(r')^j$ is their share of $[r']^{\mathcal{C}_i}_{2t_i}$, and sends each r_l for $l \in [d]$ to $P_{n_i+(j-1)\cdot d+l} \in \mathcal{C}_{i+1}$, who outputs this as their share of $[r']^{\mathcal{C}_{i+1}}_{2t_{i+1}}$.
   - Else: Let $d = n_i/n_{i+1}$ (assuming $n_{i+1} | n_i$ for simplicity). For l such that $(l-1)\cdot d < j \le l \cdot d$, each $P_j \in \mathcal{C}_i$ sends their share r'^j to $P_{n_i+l} \in \mathcal{C}_{i+1}$, who outputs as their share of $[r']^{\mathcal{C}_{i+1}}_{2t_{i+1}}$, $\sum_j r'^j$ for each P_j it received from.

Lemma 8. *Assume that at most $2t_i$ shares of $[r]^{\mathcal{C}_i}_{2t_i}$ can be computed by $\mathcal{A}$ (and the rest are uniformly random to $\mathcal{A}$). Then procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$'s transcript is simulatable with random values and preserves the invariant that at most $2t_{i+1}$ shares of $[r]^{\mathcal{C}_{i+1}}_{2t_{i+1}}$ can be computed by $\mathcal{A}$, while the rest are uniformly random to $\mathcal{A}$.*

Proof. For this proof, we assume for simplicity that $t_i = t_{i+1}$. Now, assume w.l.o.g. that the shares of $[r]^{\mathcal{C}_i}_{2t_i}$ known by $\mathcal{A}$ are $r^1, \ldots, r^{2t_i}$, which must also mean that $P_{n_i} \in \mathcal{C}_i$ is honest. Thus, we can also assume w.l.o.g. that the corrupted parties in $\mathcal{C}_i$ are $P_1, \ldots, P_{t_i}$. This means that the shares of $[o]^{\mathcal{C}_i}_{2t_i}$ held by $P_{t_i+1}, \ldots, P_{n_i}$, are unknown and uniformly random (subject to them reconstructing to 0) to $\mathcal{A}$, by the security of $\mathcal{F}_{\mathsf{zero}}$.

Now, consider what the adversarial parties in $\mathcal{T}_{\mathcal{C}_{i+1}}$ are sent from $P_{t_i+1} \ldots P_{n_i}$: $r^j + o^j$ (where each $P_j \notin \mathcal{T}_{\mathcal{C}_i}$ in the worst case). Since $\mathcal{A}$ does not know at least two shares of $[o]^{\mathcal{C}_i}_{2t_i}$, the o^j's in the communication above that it receives are each individually uniformly random. Therefore, all of these messages can be simulated with random values.

In particular, this means that even if the adversary sees $r^{n_i} + o^{n_i}$, r^{n_i} remains uniformly random and unknown to $\mathcal{A}$. Furthermore, consider some $P_l \in \mathcal{C}_{i+1}$

such that P_l itself is uncorrupted and $P_j \in \mathcal{C}_i$ who sent to P_l was not corrupted (there must exist at least one such pair). Since the o^j in P_l's share $(r')^l = r^j + o^j$ is uniformly random and unknown to $\mathcal{A}$, then $(r')^l$ itself is uniformly random and unknown to $\mathcal{A}$, even if r^j was known by $\mathcal{A}$. In fact, all of the shares in this case are uniform and unknown to $\mathcal{A}$. All other shares (corresponding to the case in which at least one of $P_l \in \mathcal{C}_{i+1}$, or the P_j who sent to P_l is corrupted) are known to $\mathcal{A}$. Thus, the invariant is preserved. □

Inefficient resharing. We will also need to reshare degree-t_i Shamir sharings across committees using Procedure $\pi_{\mathsf{ineff\text{-}reshare\text{-}hm}}$, below. This can only be done with $\Omega(n^2)$ communication, however, since it is only done once per committee, we can still achieve $O(n|C|)$ total communication if the width of circuit C is $\Omega(n)$.

Procedure 12: $\pi_{\mathsf{ineff\text{-}reshare\text{-}hm}}$

Usage $\mathcal{C}_i$ reshares $[r]_{t_i}^{\mathcal{C}_i}$ to $\mathcal{C}_{i+1}$.

1. Let r^j be P_j's share of $[r]_{t_i}^{\mathcal{C}_i}$. Each $P_j \in \mathcal{C}_i$ will create a random degree t_{i+1} Shamir secret sharing $\left[r^j\right]_{t_{i+1}}^{\mathcal{C}_i}$ of their share and distribute the corresponding shares to each $P_l \in \mathcal{C}_{i+1}$.
2. Finally, each $P_l \in \mathcal{C}_{i+1}$ will then compute $[r]_{t_{i+1}}^{\mathcal{C}_i} = \sum_{j \in \mathcal{C}_i} c_j \left[r^j\right]_{t_{i+1}}^{\mathcal{C}_i}$, where c_j is the Lagrange reconstruction coefficient for a degree-t_{i+1} polynomial.

Lemma 9. *Procedure* $\pi_{\mathsf{ineff\text{-}reshare\text{-}hm}}$*'s transcript is simulatable with random values.*

Proof. This follows easily from the fact that the shares of the honest parties of $\mathcal{C}_i$ are unknown to the adversary. So, by the security of Shamir secret sharing, the t_{i+1} shares of each honest party's share in $\mathcal{C}_i$ that the corrupt parties of $\mathcal{C}_{i+1}$ receive are uniformly random.

5.2 Incremental Checks

As in [CGG+21], we achieve active security by maintaining a few "accumulators" that somehow aggregate the potential errors that are introduced by each committee. These accumulators are updated by every other committee, and the current (possibly updated) version of the accumulator is transferred from one committee to the next. Finally, the final committees will use these accumulators to verify the integrity of the computation.

In our protocol, we make use of the "straightline" resharing procedure $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ that achieves linear communication complexity, but requires a larger threshold of $2t_i$ to achieve security. This means that the underlying secrets are not determined by the honest parties alone, and as a result a malicious adversary can in fact add errors to *any* value throughout the computation. A similar issue happens in the dishonest majority fluid protocol of [RS22], and we draw inspiration from such approach to address this attack in our protocol. The solution consists of using MACs, which are used to authenticate every intermediate value used throughout the computation and serve as additional redundancy on secret values that guarantees integrity. This is done by maintaining an accumulator that attests for the integrity of all of the reconstructions, which is built using the shared MACs and the claimed openings.

Succinctly maintaining this accumulator involves opening random challenges β to committees, who then use such β to compute random linear combinations that *compress* the verification of many MACs into one field element that should be 0. However, these challenges β should not be opened at the same time that the values whose MACs it checks are opened, for otherwise the adversary could cheat in the above linear combination. Thus, when the parties of $\mathcal{C}_i$ receive some openings and want to verify their MACs, they each hash together all of these openings and then send these hashes to each of the parties of $\mathcal{C}_{i+1}$. The challenge β is then opened to $\mathcal{C}_{i+1}$, and only P_1 of $\mathcal{C}_i$ forwards all of the openings to all of the parties of $\mathcal{C}_{i+1}$, in order to maintain linear communication. Since the hashes prevent P_1 from changing the openings, they cannot be dependent on β. Since the hashes are short, total communication will still be $O(n|C|)$ if the width of C is $\Omega(n)$.

Also note that it takes two committees to update the accumulator based on values opened to the first committee. However, since values are only opened to every other committee, there is no entanglement of updates. Details are given in Procedure $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ below.

Procedure 13: $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$

Usage: Each committee $\mathcal{C}_i$ incrementally updates a MAC check state $[\sigma]^{\mathcal{C}_i}_{2t_i}$ based on the values opened to them, which the final committees at the end of the computation use to check that all openings throughout the protocol were performed correctly.

Init: Each Party P_i in committee $\mathcal{C}_4$ (the first to have values opened to it, since the first invocation of $\pi_{\mathsf{mult\text{-}hm}}$ is by $\mathcal{C}_2$ to create randomized versions of the circuit inputs) initially defines their share of $[\sigma]^{\mathcal{C}_4}_{2t_4}$ as $\sigma^i = 0$.

Update State: On input $(\mathsf{update}, \{(A_m, [\Delta \cdot A_m]^{\mathcal{C}_i}_{2t_i})\}_{m\in[T]}, [\Delta]^{\mathcal{C}_i}_{2t_i})$ from committee $\mathcal{C}_i$, where $\{A_m\}_{m\in[T]}$ were the values opened to $\mathcal{C}_i$:

1. First each party $P_j \in \mathcal{C}_i$ samples keys $s_{j,l}$ to the universal hash family $\mathcal{H} = \{\mathcal{H}_s : \mathbb{F}_p^T \to \mathbb{F}_p\}$ for each $P_l \in \mathcal{C}_{i+1}$ and computes $h_{j,l} = \mathcal{H}_{s_{j,l}}(\{A_m\}_{m\in[T]})$.
2. In parallel: (i) each party $P_j \in \mathcal{C}_i$ then sends to each $P_l \in \mathcal{C}_{i+1}$ the universal hash key and value $s_{j,l}, h_{j,l}$; (ii) only P_1 sends $\{A_m\}_{m\in[T]}$ to each $P_l \in \mathcal{C}_{i+1}$; and (iii) all of $\mathcal{C}_i$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on $[\sigma]^{\mathcal{C}_i}_{2t_i}, \{[\Delta \cdot A_m]^{\mathcal{C}_i}_{2t_i}\}_{m\in[T]}$.
3. Each P_l in Committee $\mathcal{C}_{i+1}$ first for each $P_j \in \mathcal{C}_i$ computes $h'_{j,l} = \mathcal{H}_{s_{j,l}}(\{A_m\}_{m\in[T]})$ and checks if $h'_{j,l} = h_{j,l}$. If not, it aborts; else continues.
4. $\mathcal{C}_{i+1}$ then invokes $\mathcal{F}_{\mathsf{coin}}$ to get a random challenge β.

5. Each $P_l \in \mathcal{C}_{i+1}$ next locally computes $A = \sum_{m=1}^{T} \beta^m \cdot A_m$ and $[\gamma]_{2t_{i+1}}^{\mathcal{C}_{i+1}} = \sum_{m=1}^{T} \beta^m \cdot [\Delta \cdot A_m]_{2t_{i+1}}^{\mathcal{C}_i}$.
6. It finally updates $[\sigma]_{2t_{i+1}}^{\mathcal{C}_{i+1}} = [\sigma]_{2t_{i+1}}^{\mathcal{C}_{i+1}} + [\gamma]_{2t_{i+1}}^{\mathcal{C}_{i+1}} - [\Delta]_{2t_{i+1}}^{\mathcal{C}_{i+1}} \cdot A$ and invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on $[\sigma]_{2t_{i+1}}^{\mathcal{C}_{i+1}}$.

Check State: On input check from committee $\mathcal{C}_i$:

1. Let σ^j be the share of the MAC check state $[\sigma]_{2t_i}^{\mathcal{C}_i}$ held by each $P_j \in \mathcal{C}_i$. Each P_j creates a random degree-t_{i+1} Shamir secret share $\left[\sigma^j\right]_{t_{i+1}}^{\mathcal{C}_{i+1}}$ of their share σ^j and distributes the corresponding shares to the parties of $\mathcal{C}_{i+1}$.
2. Then each party $P_l \in \mathcal{C}_{i+1}$ computes $[\sigma]_{t_{i+1}}^{\mathcal{C}_{i+1}} = \sum_{j \in \mathcal{C}_i} c_j \cdot \left[\sigma^j\right]_{t_{i+1}}^{\mathcal{C}_{i+1}}$, where c_j is the Lagrange reconstruction coefficient for a degree-$2t_i$ polynomial.
3. Finally, parties open the shares of $[\sigma]_{t_{i+1}}^{\mathcal{C}_{i+1}}$ to each party of $\mathcal{C}_{i+2}$, who reconstruct σ, and if successful, output Accept if $\sigma = 0$; else Reject.

Lemma 10. *Procedure $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ is correct, i.e., it accepts if all the opened values A_m and the corresponding MACs are computed correctly. Moreover, it is sound, i.e., it rejects except with probability at most $(2 + \max_i T_i)/p$ in case at least one opened value is not correctly computed. Furthermore, the transcript of **Update State** is simulatable.*

Proof. The proof follows along the lines of the proof for Lemma 3 in the dishonest majority case. One difference is that the parties in $\mathcal{C}_i$ send unique universal hash keys to each party in $\mathcal{C}_{i+1}$, rather than getting them from the preprocessing of the dishonest majority protocol. But since there will be at least one honest party in each committee, at least one key will remain random and unknown to the adversary, and thus it serves the same purpose. Also, $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ gets β from $\mathcal{F}_{\mathsf{coin}}$, but from the security of $\mathcal{F}_{\mathsf{coin}}$, this is also still random and independent of all other values.

So, the adversary can thus only inject additive error δ_m^i for each m-th value A_m^i opened to $\mathcal{C}_i$, η^i for when the MAC key Δ is reshared to $\mathcal{C}_i$, ε_m^i for when the MAC of the m-th opened value is reshared to $\mathcal{C}_i$, and ζ^i for when the current accumulator value σ is reshared to $\mathcal{C}_i$. Additionally, in the **Check State** phase, since at least one share of $[\sigma]_{2t_i}^{\mathcal{C}_i}$ is unknown to $\mathcal{A}$ and remains unknown after the degree reduction step, $\mathcal{A}$ can only inject another additive error ε independent of σ. Thus, ensuring that $[\sigma]_{2t_\ell}^{\mathcal{C}_\ell} = 0$ follows the analysis of the proof of Lemma 3 for the dishonest majority case.

Finally, we know from Lemma 8 that $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ is simulatable by random values. Also, the universal hash keys are sampled randomly by the clients, and the opened values are known to the simulator, so the hash keys and their resulting hash outputs can easily be simulated.

□

Unfortunately, this is not the only kind of error we need to account for. As in [RS22], the c parts of multiplication triples that are used in $\pi_{\mathsf{mult\text{-}hm}}$

are only authenticated "on the fly". This means that the adversary can inject additive errors into these c parts that $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ will not catch (since the corresponding errors will be incorporated into the MACs, too). As a result, multiplications may not be computed correctly. To address this attack vector, we use similar ideas to [RS22, CGG$^+$21], which have their roots in the techniques of [CGH$^+$18], and consists of maintaining a randomized version of the circuit which can be used to verify multiplications. The associated accumulator is presented in Procedure $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ below.[12]

Procedure 14: $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$

Usage: Each committee $\mathcal{C}_i$ that gets the output wires of the multiplication gates of some layer ℓ of the circuit incrementally updates a multiplication verification state $(\llbracket u' \rrbracket^{\mathcal{C}_{2t_i}}, \llbracket w' \rrbracket^{\mathcal{C}_{2t_i}})$, which the final committees at the end of the computation use to check that all multiplications throughout the protocol were performed correctly.

Init: Each Party P_i in committee $\mathcal{C}_4$ (the first to get output from $\pi_{\mathsf{mult\text{-}hm}}$, as a result of $\mathcal{C}_2$ creating randomized versions of the circuit inputs) initially defines their shares of $\llbracket u' \rrbracket^{\mathcal{C}_4}, \llbracket w' \rrbracket^{\mathcal{C}_4}$ as $(u')^i = (w')^i = 0$ (same for the MAC shares).

Update State: On input $(\mathsf{update}, \{(\llbracket z_m \rrbracket^{\mathcal{C}_i}, \llbracket rz_m \rrbracket^{\mathcal{C}_i}\}_{m\in[T]})$ from committee $\mathcal{C}_i$, where $\{(\llbracket z_m \rrbracket^{\mathcal{C}_i}, \llbracket rz_m \rrbracket^{\mathcal{C}_i}\}_{m\in[T]}$ were the output wires of multiplication gates computed by $\mathcal{C}_i$:

1. Each P_j in $\mathcal{C}_i$ invokes $\mathcal{F}_{\mathsf{coin}}$ to get random challenge α.
2. Parties P_j in $\mathcal{C}_i$ locally compute $\llbracket u \rrbracket^{\mathcal{C}_i} = \llbracket u \rrbracket^{\mathcal{C}_i} + \sum_{m=1}^{T} \alpha^m \cdot \llbracket rz_m \rrbracket^{\mathcal{C}_i}$ and $\llbracket w \rrbracket^{\mathcal{C}_i} = \llbracket w \rrbracket^{\mathcal{C}_i} + \sum_{m=1}^{T} \alpha^m \cdot \llbracket z_m \rrbracket^{\mathcal{C}_i}$.
3. Finally $\mathcal{C}_i$ invokes $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on $\llbracket u \rrbracket^{\mathcal{C}_i}, \llbracket w \rrbracket^{\mathcal{C}_i}$.

Check State: On input check from the clients $\mathcal{C}_i$:

1. The parties of $\mathcal{C}_i$ first open $\llbracket r \rrbracket^{\mathcal{C}_i}$ to the parties of $\mathcal{C}_{i+1}$, who then check its MAC by running both phases of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ on it.
2. Then the parties of $\mathcal{C}_{i+4}$ ($\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ takes 4 rounds) all open $(\llbracket u \rrbracket^{\mathcal{C}_{\mathsf{clnt}}} - r \cdot \llbracket w \rrbracket^{\mathcal{C}_{\mathsf{clnt}}})$ to the parties of $\mathcal{C}_{i+5}$, who then check its MAC by running both phases of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ on it. If the opened value is 0, the parties of $\mathcal{C}_{i+8}$ ($\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ takes 4 rounds) output Accept; else Reject.

Lemma 11. *Procedure $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ is correct, i.e., it accepts if all multiplications are computed correctly. Moreover, it is sound, i.e., it rejects except with probability at most $(1+\max_i T_i)/p$ in case at least one multiplication is not correctly computed. Furthermore, the transcript of **Update State** is simulatable.*

Proof. This proof follows along the lines of Lemma 7 for the dishonest majority case. One difference is that in $\pi_{\mathsf{mult\text{-}hm}}$, we do not need to use a universal hash function. This is because we can authenticate $\llbracket c \rrbracket^{\mathcal{C}_{i-2}}$ before $(x+a), (y+b)$ are

[12] Note that the invocations of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ in the **Check State** phase of $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ can be condensed to 3 rounds, since only one value at a time is opened.

opened and thus the error in $[\![c]\!]^{\mathcal{C}_{i-2}}$ must be independent of them (and also the later opened challenge α). So, the probability of $\pi_{\textsf{mult-verify-hm}}$ failing when the adversary cheats is even lower (i.e., as in the lemma statement). Also, since parties locally compute $[c]^{\mathcal{C}_i}_{2t_i} = [a]^{\mathcal{C}_i}_{t_i} \cdot [b]^{\mathcal{C}_i}_{t_i}$, there is only additive error on c, δ_c, i.e., independent of the shares a^j, b^j of the honest parties (and same for $[\![c']\!]^{\mathcal{C}_{i+2}}$ of the randomized computation). Finally, $\pi_{\textsf{mult-verify-hm}}$ gets α from $\mathcal{F}_{\textsf{coin}}$, but from the security of $\mathcal{F}_{\textsf{coin}}$, α is in this case also random and independent of all other values.

So (assuming that $\pi_{\textsf{MAC-check-hm}}$ does not fail), the adversary can only inject additive error $\epsilon_{i,m} (= \delta_{c_{i,m}})$ for each m-th multiplication gate that $\mathcal{C}_i$ receives output for, along with $\epsilon'_{i,m}$ for that of the randomized version of the multiplication gate, and finally, any accumulated error $\eta_{i,m}$ on the randomized value from the previous gates in the circuit. Thus, ensuring that $[\![u]\!]^{\mathcal{C}_{\textsf{clnts}}} - r \cdot [\![w]\!]^{\mathcal{C}_{\textsf{clnts}}} = 0$ follows the exact same case analysis of that in the proof of Lemma 7.

Finally, we know from Lemma 8 that $\pi_{\textsf{eff-reshare-hm}}$ is simulatable by random values. □

5.3 Secure Multiplication

Finally, before we discuss our ultimate protocol, we present Procedure $\pi_{\textsf{mult-hm}}$ below which enables a given committee to make progress on the computation by securely processing multiplication gates. At a high level, this procedure makes use of multiplication triples [Bea92] to reduce the task of securely multiplying two shared values, to that of reconstructing two secrets. Reconstruction is done by using the "king idea", originating from [DN07], which achieves linear communication complexity by first reconstructing to a single party who then sends the reconstruction to the other parties.

However, there are a couple of issues we need to deal with. First, using multiplication triples requires different committees to have access to the same multiplication triple. We indeed achieve this by making use of our resharing procedure $\pi_{\textsf{eff-reshare-hm}}$ from Section 5.1. Second, a given committee can only obtain a *partially* authenticated multiplication triple $([\![a]\!]^{\mathcal{C}}, [\![b]\!]^{\mathcal{C}}, [c]^{\mathcal{C}}_{2t})$, where the c part is not authenticated. Using an idea from [RS22], we authenticate the c part of each triple "on the fly". Intuitively, this is done by masking $[c]^{\mathcal{C}}_{2t}$ with a random, authenticated sharing $[\![v]\!]^{\mathcal{C}}$, reconstructing $(c + v)$, then creating unmasked, authenticated shares of c using $[\![v]\!]^{\mathcal{C}}$ (including its MAC). Reconstructing $(c + v)$ here is also done by using the "king idea". The details are presented below.

Procedure 15: $\pi_{\textsf{mult-hm}}$

Usage: Using double sharing $([\Delta]^{\mathcal{C}_i}_{t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i})$ of the global MAC key, multiply $[\![x]\!]^{\mathcal{C}_i}$ and $[\![y]\!]^{\mathcal{C}_i}$ held by $\mathcal{C}_i$ so that $\mathcal{C}_{i+2}$ outputs $[\![x \cdot y]\!]^{\mathcal{C}_{i+2}}$.

1. All parties in $\mathcal{C}_{i-2}$ agree on a special party $P_{\textsf{king}}$ in $\mathcal{C}_{i-1}$.

2. All parties in $\mathcal{C}_{i-2}$ invoke $\pi_{\mathsf{get\text{-}rand\text{-}sharing}}$ three times to get $[\![a]\!]^{\mathcal{C}_{i-2}}, [\![b]\!]^{\mathcal{C}_{i-2}}, [\![v]\!]^{\mathcal{C}_{i-2}}$ (they also save the sharings $[a]^{\mathcal{C}_{i-2}}_{t_{i-2}}, [b]^{\mathcal{C}_{i-2}}_{t_{i-2}}$ generated during this invocation).
3. $\mathcal{C}_{i-2}$ then locally obtains (unauthenticated) $[c]^{\mathcal{C}_{i-2}}_{2t_{i-2}} = [a]^{\mathcal{C}_{i-2}}_{t_{i-2}} \cdot [b]^{\mathcal{C}_{i-2}}_{t_{i-2}}$.
4. Finally, parties in $\mathcal{C}_{i-2}$ in parallel invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on input $[\![a]\!]^{\mathcal{C}_{i-2}}, [\![b]\!]^{\mathcal{C}_{i-2}}, [\![v]\!]^{\mathcal{C}_{i-2}}$ and open $[c+v]^{\mathcal{C}_{i-2}}_{2t_{i-2}}$ to P_{king} in $\mathcal{C}_{i-1}$.
5. Then, while P_{king} distributes opened $(c+v)$ to the parties of $\mathcal{C}_i$, the rest of the parties in $\mathcal{C}_{i-1}$ in parallel invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on input $[\![a]\!]^{\mathcal{C}_{i-1}}, [\![b]\!]^{\mathcal{C}_{i-1}}, [\![v]\!]^{\mathcal{C}_{i-1}}$.
6. Parties in $\mathcal{C}_i$ then use opened $(c+v)$ to compute authenticated $[\![c]\!]^{\mathcal{C}_i} = ((c+v) - [v]^{\mathcal{C}_i}_{2t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i} \cdot (c+v) - [\Delta \cdot v]^{\mathcal{C}_{i+2}}_{2t_i}, [\Delta]^{\mathcal{C}_i}_{2t_i})$.
7. Parties in $\mathcal{C}_i$ agree on a special party P'_{king} in $\mathcal{C}_{i+1}$ and then compute $[\![x+a]\!]^{\mathcal{C}_i} = [\![x]\!]^{\mathcal{C}_i} + [\![a]\!]^{\mathcal{C}_i}$, and $[\![y]\!]^{\mathcal{C}_i} = [\![y]\!]^{\mathcal{C}_i} + [\![b]\!]^{\mathcal{C}_i}$.
8. Parties in $\mathcal{C}_i$ then in parallel open $[\![x+a]\!]^{\mathcal{C}_i}$, $[\![y+b]\!]^{\mathcal{C}_i}$ to P'_{king} in $\mathcal{C}_{i+1}$ and invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on $[\![a]\!]^{\mathcal{C}_i}, [\![b]\!]^{\mathcal{C}_i}, [\![c]\!]^{\mathcal{C}_i}, [\Delta \cdot (x+a)]^{\mathcal{C}_i}_{2t_i}$, $[\Delta \cdot (y+b)]^{\mathcal{C}_i}_{2t_i}$.
9. Then while P'_{king} distributes opened $d = x + a$ and $e = y + b$ to the parties of $\mathcal{C}_{i+2}$, all parties in $\mathcal{C}_{i+1}$ invoke $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ on input $[\![a]\!]^{\mathcal{C}_{i+1}}, [\![b]\!]^{\mathcal{C}_{i+1}}, [\![c]\!]^{\mathcal{C}_{i+1}}, [\Delta \cdot (x+a)]^{\mathcal{C}_{i+1}}_{2t_{i+1}}$, $[\Delta \cdot (y+b)]^{\mathcal{C}_{i+1}}_{2t_{i+1}}$.
10. $\mathcal{C}_{i+2}$ finally locally computes $[\![x \cdot y]\!]^{\mathcal{C}_{i+2}} = de - d\,[\![b]\!]^{\mathcal{C}_{i+2}} - e\,[\![a]\!]^{\mathcal{C}_{i+2}} + [\![c]\!]^{\mathcal{C}_{i+2}}$.
11. Parties in $\mathcal{C}_{i+2}$ will also invoke $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ on input $(\mathsf{update}, \{((x_m + a_m, [\Delta \cdot (x_m + a_m)]^{\mathcal{C}_{i+2}}_{2t_{i+2}}), (y_m + b_m, [\Delta \cdot (y_m + b_m)]^{\mathcal{C}_{i+2}}_{2t_{i+2}}))\}_{m \in [T]})$ corresponding to all of the openings of the above form they receive for the multiplication gates at this layer of the circuit.

Lemma 12. *Procedure $\pi_{\mathsf{mult\text{-}hm}}$'s transcript is simulatable.*

Proof. First, we know that $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ is simulatable by random values from Lemma 8. Also, since a, b, v are uniformly random and unknown to the adversary by the security of $\mathcal{F}_{\mathsf{double\text{-}rand}}$, openings $(c+v), (x+a), (y+b)$ are simulatable by random values. Finally, from Lemma 10, we know that $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$'s transcript is indeed simulatable. □

5.4 Honest Majority Protocol

With all the previous tools into place, we are finally ready to present our full-fledged actively secure, honest majority MPC protocol in the fluid setting, achieving linear communication complexity and maximal fluidity. The clients first distribute double sharings $([x_i]^{\mathcal{C}_1}_{t_1}, [x_i]^{\mathcal{C}_1}_{2t_1})$ of their inputs to $\mathcal{C}_1$. Then $\mathcal{C}_1$ obtains a double sharing $([\Delta]^{\mathcal{C}_1}_{t_1}, [\Delta]^{\mathcal{C}_1}_{2t_1})$ of the global MAC key using $\mathcal{F}_{\mathsf{double\text{-}rand}}$, and forms authenticated SPDZ sharings of the inputs using these sharings. The committees then proceed to compute both the regular and randomized version of the circuit on the authenticated inputs, using $\pi_{\mathsf{mult\text{-}hm}}$ as well as addition and identity gate procedures that work similarly using the same "mask, open to king, and unmask" paradigm along with some local computation. The committees also make sure to update the accumulators of $\pi_{\textsf{MAC-check-hm}}$ and $\pi_{\textsf{mult-verify-hm}}$ along the way with each opening and multiplication, respectively. Finally, once all circuit layers have been computed, the final committees invoke the **Check State** phases of $\pi_{\textsf{MAC-check-hm}}$ and $\pi_{\textsf{mult-verify-hm}}$, then reconstruct the outputs to the clients. We note that, as it is remarked in the protocols of [RS22, CGG^{+}21], if the clients indeed have access to a broadcast channel in the last round of the protocol, or implement a broadcast over their point-to-point channels, then security with unanimous abort can be achieved by having the clients broadcast "abort", if their check on their output fails.

Protocol 16: $\Pi_{\textsf{main-hm}}$

Input Phase: To form a SPDZ sharing of an input x_i possessed by $P_i \in \mathcal{C}_{\textsf{clnt}}$:

1. P_i samples random degree-t_1 and degree-$2t_1$ Shamir sharings of x_i and distributes the corresponding shares to all parties in $\mathcal{C}_1$.
2. $\mathcal{C}_1$ then invokes $\mathcal{F}_{\textsf{double-rand}}$ to get random double sharings of the global MAC key $([\Delta]_{t_1}^{\mathcal{C}_1}, [\Delta]_{2t_1}^{\mathcal{C}_1})$.
3. Next, parties in $\mathcal{C}_1$ locally obtain authenticated sharing $[\![x_i]\!]^{\mathcal{C}_1} = ([x_i]_{2t_1}^{\mathcal{C}_1}, [x_i]_{t_1}^{\mathcal{C}_1} \cdot [\Delta]_{t_1}^{\mathcal{C}_1}, [\Delta]_{2t_1}^{\mathcal{C}_1})$ and invoke $\pi_{\textsf{ineff-reshare-hm}}$ on input $[\Delta]_{t_1}^{\mathcal{C}_1}$ and $\pi_{\textsf{eff-reshare-hm}}$ on input $[\![x_i]\!]^{\mathcal{C}_1}$.[a]
4. Parties in $\mathcal{C}_2$ then invoke $\pi_{\textsf{get-rand-sharing}}$ to get $[\![r]\!]^{\mathcal{C}_2}$ and invoke $\pi_{\textsf{mult-hm}}$ on input $[\![x_i]\!]^{\mathcal{C}_2}$ and $[\![r]\!]^{\mathcal{C}_2}$, as well as the identity gate procedure (below) on $[\![x_i]\!]^{\mathcal{C}_2}$ so that $\mathcal{C}_4$ gets $[\![x_i]\!]^{\mathcal{C}_4}, [\![rx_i]\!]^{\mathcal{C}_4}$.
5. Finally, $\mathcal{C}_4$ invokes $\pi_{\textsf{mult-verify-hm}}$ on input $(\textsf{update}, \{([\![x_i]\!]^{\mathcal{C}_4}, [\![rx_i]\!]^{\mathcal{C}_4})\}_{i\in[|\mathcal{C}_{\textsf{clnt}}|]})$, corresponding to each input.

Execution Phase:

1. Each Committee $\mathcal{C}_i$ of the execution phase will first of all invoke $\pi_{\textsf{ineff-reshare-hm}}$ on input $[\Delta]_{t_i}^{\mathcal{C}_i}$ and $\pi_{\textsf{eff-reshare-hm}}$ on input $[\![r]\!]^{\mathcal{C}_i}$ and $[\Delta]_{2t_i}^{\mathcal{C}_i}$.
2. In parallel, every other committee (with the help of the others) will compute the gates at each layer of the circuit as below:

Addition: To perform addition on $[\![x]\!]^{\mathcal{C}_i}$ and $[\![y]\!]^{\mathcal{C}_i}$ (and identically for $[\![rx]\!]^{\mathcal{C}_i}$ and $[\![ry]\!]^{\mathcal{C}_i}$):

1. All parties in $\mathcal{C}_i$ agree on a special party $P_{\textsf{king}}$ in $\mathcal{C}_{i+1}$ then invoke $\pi_{\textsf{get-rand-sharing}}$ to get $[\![s]\!]^{\mathcal{C}_i}$.
2. Then, parties in $\mathcal{C}_i$ locally obtain $[\![x+y+s]\!]^{\mathcal{C}_i}$ and open it to $P_{\textsf{king}}$ while invoking $\pi_{\textsf{eff-reshare-hm}}$ on input $[\![s]\!]^{\mathcal{C}_i}$.
3. While $P_{\textsf{king}}$ distributes opened $x+y+s$ to the parties of $\mathcal{C}_{i+2}$, all parties in $\mathcal{C}_{i+1}$ invoke $\pi_{\textsf{eff-reshare-hm}}$ on $[\![s]\!]^{\mathcal{C}_{i+1}}$.
4. Parties in $\mathcal{C}_{i+2}$ finally locally compute $[\![x+y]\!]^{\mathcal{C}_{i+2}} = (x+y+s) - [\![s]\!]^{\mathcal{C}_{i+2}}$.
5. Parties in $\mathcal{C}_{i+2}$ will also invoke $\pi_{\textsf{MAC-check-hm}}$ on input $(\textsf{update}, \{(x_m+y_m+s_m, [\Delta\cdot(x_m+y_m+s_m)]_{2t_{i+2}}^{\mathcal{C}_{i+2}})\}_{m\in[T]})$ corresponding to all of the openings of the above form they receive for the addition gates at this layer of the circuit.[b]

Identity Gates: $\mathcal{C}_i$ forwards $[\![x]\!]^{\mathcal{C}_i}, [\![rx]\!]^{\mathcal{C}_i}$ to $\mathcal{C}_{i+2}$ in a similar fashion as addition above.

Multiplication: To multiply $[\![x]\!]^{\mathcal{C}_i}$ and $[\![y]\!]^{\mathcal{C}_i}$, invoke $\pi_{\mathsf{mult\text{-}hm}}$ on them (and identically for $[\![rx]\!]^{\mathcal{C}_i}$ and $[\![y]\!]^{\mathcal{C}_i}$). Then invoke $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ on input $(\mathsf{update}, \{([\![x_m y_m]\!]^{\mathcal{C}_{i+2}}, [\![(rx)_m y_m]\!]^{\mathcal{C}_{i+2}})\}_{m\in[T_i]})$, corresponding to each multiplication performed at this layer of the circuit.

Output Phase:

1. Parties in the last committee $\mathcal{C}_\ell$ who compute the shares of the output gates then invoke $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ on check. If it outputs Reject, then abort; else, continue.
2. Parties in $\mathcal{C}_{\ell+2}$ (check of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ takes 3 rounds) then invoke $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ on check. If it outputs Reject, then abort; else, continue.
3. Next, for Party P_j in $\mathcal{C}_{\ell+9}$ (check of $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ takes 8 rounds), let $z^j, (\Delta\cdot z)^j$ be their respective shares of output wire $[z]_{2t_{\ell+9}}^{\mathcal{C}_{\ell+9}}$ and MAC $[\Delta\cdot z]_{2t_{\ell+9}}^{\mathcal{C}_{\ell+9}}$. Each P_j creates random degree-$t_{\ell+10}$ Shamir secret sharings $\left[z^j\right]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}, \left[(\Delta\cdot z)^j\right]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$ and distributes the corresponding shares to the parties of $\mathcal{C}_{\ell+10}$.
4. Then each party $P_l \in \mathcal{C}_{\ell+10}$ computes $[z]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}} = \sum_{j\in\mathcal{C}_{\ell+9}} c_j \cdot \left[z^j\right]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$, and similarly for $[\Delta\cdot z]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$, where c_j is the Lagrange reconstruction coefficient for a degree-$2t_{\ell+9}$ polynomial.
5. Finally, the parties of $\mathcal{C}_{\ell+10}$ open the shares of each $[z]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$, $[\Delta]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$, and $[\Delta\cdot z]_{t_{\ell+10}}^{\mathcal{C}_{\ell+10}}$ to the clients, who attempt to reconstruct them and check that indeed the product of the former two values equal the last value. If so, they output each z; else, they abort.

[a] Note that the computed MAC for $[\![x_i]\!]^{\mathcal{C}_1}$ is re-randomized with a fresh 0-sharing in $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$.

[b] This invocation can be combined with that of the multiplication gates for this circuit layer.

Theorem 5. *Let $\mathcal{A}$ be an* R*-adaptive adversary in $\Pi_{\mathsf{main\text{-}hm}}$. Then the protocol UC-securely computes $\mathcal{F}_{\mathsf{DABB}}$ in the presence of $\mathcal{A}$ in the $(\mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{double\text{-}rand}}, \mathcal{F}_{\mathsf{zero}})$-hybrid model.*

Proof. The proof of this Theorem follows very similarly to that of Theorem 4. We construct a Simulator ($\mathcal{S}$) that runs the adversary ($\mathcal{A}$) as a subroutine, and is given access to $\mathcal{F}_{\mathsf{DABB}}$. It internally emulates the functionalities $\mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{coin}}, \mathcal{F}_{\mathsf{double\text{-}rand}}, \mathcal{F}_{\mathsf{zero}}$ and we implicitly assume that it passes all communication between $\mathcal{A}$ and the environment $\mathcal{Z}$. It keeps track of the current committee via inputs $(\mathsf{Init}, \mathcal{C})$ and $(\mathsf{Next\text{-}Committee}, \mathcal{C})$ from $\mathcal{F}_{\mathsf{DABB}}$ (and therefore in which committees to simulate corresponding communication for circuit gates). The simulator uses bad, initially set to 0, to detect any bad behavior from $\mathcal{A}$. If so, it sets $\mathsf{bad} = 1$. The simulation proceeds as follows:

Input: On input $(\mathsf{Input}, \mathsf{id}_x)$, if it is from an honest party, simulate the double sharings using random values. From the security of Shamir secret sharing, this is a perfect simulation. For inputs from all (even adversarial) parties, simulate $\pi_{\mathsf{ineff\text{-}reshare\text{-}hm}}$ as in Lemma 9 and $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ as in Lemma 8. If $\mathcal{A}$ cheats when resharing a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$. Also, simulate the multiplication as below.

Addition (and similarly for identity gates): On input $(\mathsf{Add}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$, simulate $\pi_{\mathsf{eff\text{-}reshare\text{-}hm}}$ as in Lemma 8, and $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ as in Lemma 10. Additionally, simulate the opening of $[\![x + y + s]\!]^{\mathcal{C}_i}$ with random values. Since s is uniformly random and unknown to $\mathcal{A}$, this is a perfect simulation. If $\mathcal{A}$ cheats when sending some universal hash value during $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$, abort. If $\mathcal{A}$ cheats either when resharing or opening a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$.

Multiplication: On input $(\mathsf{Mult}, \mathsf{id}_z, \mathsf{id}_x, \mathsf{id}_y)$, simulate $\pi_{\mathsf{mult\text{-}hm}}$ as in Lemma 12 and $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$ as in Lemma 11. If $\mathcal{A}$ cheats when sending some universal hash value during $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$, abort. If for any c part of a multiplication triple, the additive error $\delta_c \neq 0$, set $\mathsf{bad} = 1$. Additionally, if $\mathcal{A}$ cheats either when resharing or opening a value (i.e., by sending a wrong share), set $\mathsf{bad} = 1$.

Output: In the **Check State** phase of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$, for the distributed degree t_{i+1} Shamir sharings we know from Lemma 8, that at least one share of $[\sigma]_{2t_i}^{\mathcal{C}_i}$ is uniformly random and unknown to $\mathcal{A}$, while all others can be computed by $\mathcal{A}$. For those that are uniformly random, by the security of Shamir secret sharings, the distributed degree t_{i+1} shares can be simulated with random values. For those that are known, the simulator can simply sample the distributed degree t_{i+1} shares on its own. Now, if $\mathcal{S}$ ever set $\mathsf{bad} = 1$ because $\mathcal{A}$ cheated when opening or resharing a value, $\mathcal{S}$ sends random values for σ on behalf of the honest parties, then aborts. Otherwise, $\mathcal{S}$, samples random shares for the honest parties such that they reconstruct to 0 and are consistent with the degree t_{i+1} sampled by the simulator above, and sends them to $\mathcal{A}$. If $\mathcal{A}$ cheats during the *check state* phase of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$ (e.g., by distributing incorrect degree-t_i shares of some share σ^i), $\mathcal{S}$ aborts. In the *check state* phase of $\pi_{\mathsf{mult\text{-}verify\text{-}hm}}$, $\mathcal{S}$ sends random shares on behalf of the honest parties for the opening of r. If $\mathcal{A}$ cheats by opening the wrong values for r, $\mathcal{S}$ aborts after the MAC check (as above). If $\mathcal{S}$ ever set $\mathsf{bad} = 1$ because $\mathcal{A}$ added non-zero error to the c part of a multiplication triple, $\mathcal{S}$ sends random values on behalf of the honest parties for $(u - r \cdot w)$, then aborts. Otherwise, $\mathcal{S}$ records the values sent by $\mathcal{A}$ for $(u - r \cdot w)$, then samples shares such that $(u - r \cdot w) = 0$, and sends them to $\mathcal{A}$. If $\mathcal{A}$ cheats when opening $(u - r \cdot w)$, $\mathcal{S}$ aborts after the MAC check (as above).

Finally, $\mathcal{S}$ gets the outputs from $\mathcal{F}_{\mathsf{DABB}}$ and forwards it to $\mathcal{A}$. $\mathcal{S}$ then forwards whatever it receives from $\mathcal{A}$ back to $\mathcal{F}_{\mathsf{DABB}}$. We can simulate the opening of the output wires (and MACs) in a similar fashion as the *check state* phase of $\pi_{\mathsf{MAC\text{-}check\text{-}hm}}$.

From all of the Lemmas, we have that the simulation is perfect up until the output phase. By Lemmas 10 (a similar argument holds for checking the MACs of the output wires) and 11, $\mathcal{A}$ is only able to cheat in the real world with probability negligible in p. Thus, the distance between the real-world and the simulation is negligible in p. $\square$

6 Dishonest Majority Preprocessing Size is Tight

In this section, we show that the per-party size of the preprocessing produced in our dishonest majority protocol $\Pi_{\mathsf{main\text{-}dm}}$ is tight in the following sense. *Any* protocol that uses more than one committee to compute some function must have per-party size of preprocessing proportional to N, i.e. the size of the entire server universe, $\mathcal{U}$.

To show this, we intuitively reduce the problem of MPC in the Fluid Model with more than two committees to the problem of simply resharing state securely. Indeed, for any such MPC protocol, after one committee finishes their step of the computation, they must securely reshare some sort of state to the next committee, since the next committee has no information about the current state of the computation (e.g., including the original inputs).

More formally, we show a lower bound on the per-party preprocessing size for Secure Message Transmission (SMT) with two committees. In such an SMT setting, there is a sender A who wishes to send some (possibly uniformly random) message x to a receiver B, but first must send some private representation of x through two committees, $\mathcal{C}_1$ and $\mathcal{C}_2$ that are not known ahead of time. Informally, this corresponds to the "resharing" argument in the Fluid MPC model above, since the transmitted x corresponds to the state that is being reshared by the first committee to the next.

6.1 Lower Bound Preliminaries

Here we present some additional notation and definitions from probability and information theory. For a random variable X we use $\mathrm{H}(X)$ to represent its *Shannon entropy*. For two random variables X and Y, we define their *mutual information* as:

$$\mathrm{I}(X;Y) = \mathrm{H}(X) - \mathrm{H}(X|Y).$$

Also, for two random variables X and Y over the same space $\mathcal{Z}$, we define the *statistical distance* between their probability distributions as:

$$\mathsf{SD}(X,Y) = \max_{Z \subseteq \mathcal{Z}} |\Pr[X \in Z] - \Pr[Y \in Z]|.$$

Finally, we define the *Kullback-Leibler divergence* of the probability distribution of X from that of Y as:

$$D_{\mathrm{KL}}(X||Y) = \sum_{z \in \mathcal{Z}} \Pr[X = z] \cdot \log\left(\frac{\Pr[X = z]}{\Pr[Y = z]}\right).$$

6.2 Secure Message Transmission with Two Committees

Now we formally define SMT with two committees. In this definition, we will demand that a uniformly random message x of length λ, i.e., $x \leftarrow_\$ \{0,1\}^\lambda$, will be transmitted from A to B, after passing through committees $\mathcal{C}_1$ and $\mathcal{C}_2$. The two committees $\mathcal{C}_1$ and $\mathcal{C}_2$ can be arbitrarily chosen from a larger universe $\mathcal{U} = \{P_1, \ldots, P_N\}$ of size N. We will allow for a *preprocessing phase* to be performed *before* the input x and committees $\mathcal{C}_1$ and $\mathcal{C}_2$ are chosen. First we present the syntax:

- Algorithm $\{r_i\}_{P_i \in \mathcal{U}} \leftarrow_\$ \mathsf{SMT\text{-}Prep}(\mathcal{U})$ takes as input the set of parties in $\mathcal{U}$ and outputs a preprocessing state, r_i, for each $P_i \in \mathcal{U}$.
- The sender will use algorithm $\{A_i\}_{P_i \in \mathcal{C}_1} \leftarrow_\$ \mathsf{SMT\text{-}A\text{-}Send}(x, \mathcal{C}_1)$ to send messages A_i for each P_i in $\mathcal{C}_1$, based on chosen $x \in \{0,1\}^\lambda$.
- Each $P_i \in \mathcal{C}_1$ will then use $\{c_{i,j}\}_{P_j \in \mathcal{C}_2} \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_1\mathsf{\text{-}Send}(r_i, A_i, \mathcal{C}_2)$ to send message $c_{i,j}$ to each P_j in $\mathcal{C}_2$.
- Next, each $P_j \in \mathcal{C}_2$ will use algorithm $B_j \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_2\mathsf{\text{-}Send}(r_j, \{c_{i,j}\}_{P_i \in \mathcal{C}_1}, \mathcal{C}_1)$ to send message B_j to the receiver.
- Finally, the receiver will use algorithm $x \leftarrow \mathsf{SMT\text{-}B\text{-}Rcv}(\{B_j\}_{P_j \in \mathcal{C}_2})$ to output the message x.

Since we are in the dishonest majority setting, we will consider any *unbounded* adversary $\mathcal{A}$ that corrupts all-but-one party in each committee, *only* during the online phase. That is, using the same notation as earlier in the paper, the sizes of corruption sets $\mathcal{T}_{\mathcal{C}_1}$ and $\mathcal{T}_{\mathcal{C}_2}$ satisfy $t_1 < n_1$ and $t_2 < n_2$, respectively. Now, we are ready for the definition:

Definition 1 (Secure Message Transmission with Two Committees.). *A* Secure Message Transmission with Two Committees *protocol Π_{SMT} is* perfectly-correct *if for any choice of committees $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{U}$,*

$$\Pr\big[x \leftarrow \mathsf{SMT\text{-}B\text{-}Rcv}(\{B_j\}_{P_j \in \mathcal{C}_2}) : x \leftarrow_\$ \{0,1\}^\lambda, \{r_l\}_{P_l \in \mathcal{U}} \leftarrow_\$ \mathsf{SMT\text{-}Prep}(\mathcal{U}),$$
$$\{A_i\}_{P_i \in \mathcal{C}_1} \leftarrow_\$ \mathsf{SMT\text{-}A\text{-}Send}(x, \mathcal{C}_1), \forall P_i \in \mathcal{C}_1, \{c_{i,j}\}_{P_j \in \mathcal{C}_2} \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_1\mathsf{\text{-}Send}(r_i, A_i, \mathcal{C}_2),$$
$$\forall P_j \in \mathcal{C}_2, B_j \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_2\mathsf{\text{-}Send}(r_j, \{c_{i,j}\}_{P_i \in \mathcal{C}_1}, \mathcal{C}_1)] = 1.$$

Moreover, Π_{SMT} is statistically-secure *if for any choice of committees $\mathcal{C}_1, \mathcal{C}_2 \subseteq \mathcal{U}$, and any choice of corruptions $\mathcal{T}_{\mathcal{C}_1} \subseteq \mathcal{C}_1, \mathcal{T}_{\mathcal{C}_2} \subseteq \mathcal{C}_2$ satisfying $t_1 < n_1$ and $t_2 < n_2$, respectively,*

$$\Pr\big[x \leftarrow \mathcal{A}(\{(r_i, A_i)\}_{P_i \in \mathcal{T}_{\mathcal{C}_1}}, \{(r_j, \{c_{l,j}\}_{P_l \in \mathcal{C}_1})\}_{P_j \in \mathcal{T}_{\mathcal{C}_2}}) : x \leftarrow_\$ \{0,1\}^\lambda,$$
$$\{r_l\}_{P_l \in \mathcal{U}} \leftarrow_\$ \mathsf{SMT\text{-}Prep}(\mathcal{U}), \{A_i\}_{P_i \in \mathcal{C}_1} \leftarrow_\$ \mathsf{SMT\text{-}A\text{-}Send}(x, \mathcal{C}_1),$$
$$\forall P_i \in \mathcal{C}_1, \{c_{i,j}\}_{P_j \in \mathcal{C}_2} \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_1\mathsf{\text{-}Send}(r_i, A_i, \mathcal{C}_2),$$
$$\forall P_j \in \mathcal{C}_2, B_j \leftarrow_\$ \mathsf{SMT\text{-}}\mathcal{C}_2\mathsf{\text{-}Send}(r_j, \{c_{i,j}\}_{P_i \in \mathcal{C}_1}, \mathcal{C}_1)] \leq 2^{-\lambda}.$$

In the rest of the section, we will assume for simplicity that the two committees, $\mathcal{C}_1$ and $\mathcal{C}_2$, will each be of the same size $n_1 = n_2 = n$. Without loss of generality, we may refer to the two committees as $\mathcal{C}_1 = \{P_1, \ldots, P_n\}$ and $\mathcal{C}_2 = \{P_{n+1}, \ldots, P_{2n}\}$.

Communication Complexity. We will in part be concerned by the size of communication needed for a correct and secure SMT protocol. Towards this end, let $\mathsf{Comm} = \sum_{i \in \mathcal{C}_1, j \in \mathcal{C}_2} |c_{i,j}|$ be the total communication from some particular execution of an SMT protocol Π_{SMT}.

6.3 Lower Bound on Per-Party Preprocessing for Linear SMT

We will now prove the following lower bound which informally states that in order for an SMT protocol Π_{SMT} to have, in expectation over the choice of committees $\mathcal{C}_1, \mathcal{C}_2$ and any randomness of the algorithms, $o(n^2 \cdot \lambda)$ total communication Comm, the expected size of each preprocessing state must be $\Omega(N \cdot \lambda)$:

Theorem 6. *For any perfectly-secure SMT protocol Π_{SMT} for two committees $\mathcal{C}_1, \mathcal{C}_2$ of size n, if $\mathcal{C}_1, \mathcal{C}_2$ are sampled uniformly at random from the universe $\mathcal{U}$ of size N, such that $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$, and $\mathbb{E}_{P_i \in \mathcal{U}}[|r_i|] \leq (N - 2n + 1) \cdot \lambda/8$, then $\mathbb{E}_{\mathcal{C}_1, \mathcal{C}_2}[\mathsf{Comm}] \geq n^2 \cdot \lambda/4$.*

First, we provide the following lemma that will help us in proving the above theorem. Assume w.l.o.g. that the smallest ciphertext in a given execution is $c_{1,n+1}$. Also, assume that the adversary corrupts (at least) every party in the two committees except for P_1 and P_{n+1}. In particular, this gives the adversary preprocessing states $R = r_2, \ldots, r_n, r_{n+2}, \ldots, r_{2n}$, ciphertexts $\{c_{i,n+1}\}_{i \in [2,n]}$ sent by the corrupted parties of $\mathcal{C}_1$ to P_{n+1}, and ciphertexts $B_{n+2}, \ldots, B_{2n}$ sent to the receiver by the corrupted parties of $\mathcal{C}_2$. So, the adversary is just missing the message B_{n+1} used by the receiver B in the protocol to reconstruct x. To produce this message B_{n+1}, the adversary in addition to ciphertexts $\{c_{i,n+1}\}_{i \in [2,n]}$ it has, only needs to learn $c_{1,n+1}$ and r_{n+1}. What this Lemma intuitively shows is that if $|c_{1,n+1}| < \lambda/2$ (so that the adversary can guess it with high enough probability), then the preprocessing r_1 *must* provide some additional, non-trivial correlation with r_{n+1} that the corrupted preprocessing states, R, do not provide on their own. If this were not the case, then the adversary could simply sample r_{n+1} conditioned on R. This preprocessing would then be "close enough" to what the correlations in the entire protocol execution indicate it should be so that, by correctness, it should also work, together with the ciphertexts $\{c_{i,n+1}\}_{i \in [n]}$, to produce the missing B_{n+1}.

In the following, we use notation R_J to represent the set of preprocessing states $\{r_j\}_{j \in J}$, where $J \subseteq [N]$.

Lemma 13. *Assume that the two committees $\mathcal{C}_1, \mathcal{C}_2$, each of size n, are chosen uniformly at random from the universe $\mathcal{U}$ of size N, such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. Also, let J be some random (fixed-size) subset of $[N]$ such that $|J| \geq 2n - 2$. If $\Pr_{\mathcal{C}_1, \mathcal{C}_2, i, j}[|c_{i,j}| < \lambda/2] > 1/2$, then for random $i' \neq j' \in \mathcal{U} \setminus J$, we have: $\mathbb{E}_{J, i', j'}[\mathrm{I}(r_{j'} | R_J; r_{i'})] \geq \lambda/4$.*

Proof. We start with the following inequality, where the randomness is over the choice of $J \cup \{i', j'\} \subseteq [N]$, as well as the actual generated preprocessing state $r_{i'}$:

$$\mathbb{E}_{J, i', j', r_{i'}}[\mathsf{SD}((r_{j'} | R_J \cup \{r_{i'}\}), (r_{j'} | R_J))] \leq$$

$$\mathbb{E}_{J,i',j'r_{i'}}\left[1-\frac{1}{2}\exp(-D_{\mathrm{KL}}((r_{j'}|R_J\cup\{r_{i'}\})||(r_{j'}|R_J)))\right]\leq$$

$$1-\frac{1}{2}\exp(-\mathbb{E}_{J,i',j',r_{i'}}[D_{\mathrm{KL}}((r_{j'}|R_J\cup\{r_{i'}\})||(r_{j'}|R_J))])=$$

$$1-\frac{1}{2}\exp(-\mathbb{E}_{J,i',j'}[\mathrm{I}(r_{j'}|R_J;r_{i'})]).$$

The first inequality follows from the Bretagnolle-Huber inequality [BH78], and the second inequality from Jensen's inequality, since $f(x)=e^{-x}$ is convex, while the last equality is a well-known identity.

Thus, if we assume towards contradiction that $\mathbb{E}_{J,i',j'}[\mathrm{I}(r_{j'}|R_J;r_{i'})]<\lambda/4$, this means that

$$\mathbb{E}_{J,i',j',r_{i'}}[\mathsf{SD}((r_{j'}|R_J),(r_{j'}|R_J\cup\{r_{i'}\}))]<1-\frac{1}{2}\exp(-\lambda/4).$$

Now, it could be the case that some $2n-2$ randomly sampled indices in J correspond exactly to the first $n-1$ parties of $\mathcal{C}_1$ and $\mathcal{C}_2$ in a given protocol execution, and further that $P_{i'}$ is the last party chosen for $\mathcal{C}_1$, and $P_{j'}$ is the last party chosen for $\mathcal{C}_2$. Also, from correctness, we know that for any $r_{j'}$ that is produced by the preprocessing phase for $P_{j'}$ with non-zero probability, the SMT protocol must successfully transmit the secret x. Based on this and the above inequality, we describe the following attack: The adversary $\mathcal{A}$ corrupts the set of preprocessing states R_J and then samples guess $r'_{j'}$ for $r_{j'}$ conditioned on the states in R_j, and samples (uniformly) guess $c'_{i',j'}$ for $c_{i',j'}$. Using the guessed $r'_{j'}$ and $c'_{i,j'}$ along with the learned $\{c_{i,j'}\}_{i\in[n]\setminus\{i'\}}$ via corruptions, invoke $\mathsf{SMT}\text{-}\mathcal{C}_2\text{-}\mathsf{Send}$ to produce $B_{j'}$. Finally, using $\{B_j\}_{j\in[n+1,2n]}$, invoke $\mathsf{SMT}\text{-}\mathsf{B}\text{-}\mathsf{Rcv}$ to produce x.

Now, let us analyze the success probability of this attack. First, we have from $\Pr_{\mathcal{C}_1,\mathcal{C}_2,i,j}[|c_{i,j}|<\lambda/2]>1/2$, that $\Pr[c'_{j',j}=c_{j',j}]>2^{-\lambda/2-1}$. Next, consider the event in which $\mathcal{A}$ samples some $r'_{j'}$ conditioned on R_J that has weight-0 in the distribution of $r_{j'}$ conditioned on $R_J\cup\{r_{i'}\}$. We call such an $r'_{j'}$ a *bad* sample. From the above inequality, such an $r'_{j'}$ is sampled with probability less than $1-\frac{1}{2}\exp(-\lambda/4)$, in expectation. In particular, this means that in expectation, $\mathcal{A}$ samples a *good* $r'_{j'}$ with probability at least $\frac{1}{2}\exp(-\lambda/4)$. Such an $r'_{j'}$ is *good* because by correctness, the protocol must successfully transmit the secret x if in fact $r'_{j'}$ were the actual preprocessing of $P_{j'}$.

Therefore, in expectation over the choice of committee members and the preprocessing $r_{i'}$, the attack by $\mathcal{A}$ succeeds with probability greater than $1/2^{\lambda}$. □

Now we can prove Theorem 6 using Lemma 13. The intuition stems from the fact that the protocol does not *a priori* know which parties will be in the committees. So, we can use Lemma 13 to show that in fact *many* parties outside of the two committees must in expectation provide some unique correlation with r_{n+1} (the receiver of small message $c_{1,n+1}$), in case they were actually the first party of $\mathcal{C}_1$ sending this small ciphertext. As a result, if the state r_{n+1} is small enough, we can completely recover it, guess $c_{1,n+1}$, then recover x with high enough probability.

Proof (of Theorem 6). Assume towards contradiction that $\Pr_{\mathcal{C}_1,\mathcal{C}_2,i,j}[|c_{i,j}| < \lambda/2] > 1/2$. Also assume that some adversary $\mathcal{A}$ in a given execution of some Π_{SMT} first corrupts every party in $\mathcal{C}_1$ and $\mathcal{C}_2$ except randomly chosen P_i of $\mathcal{C}_1$ and randomly chosen P_j of $\mathcal{C}_2$. In particular, this means that the set of indices J of corrupt parties is some random subset of $[N]$ of size $2n-2$, and index j is some random other index outside of J. Now, the adversary will one by one sample $M = (N-2n+1)/2$ indices $i_1, \ldots, i_M$ from $\mathcal{U}$ that are not already part of J, and add them to J. Let J' be the final such set. From Lemma 13, we know that under the above assumption on message size, for any random, fixed-size subset $J \subseteq [N]$ of size $|J| \geq 2n-2$, and random index $j' \notin J$, if we pick another random index $i_l \notin J$, $\mathbb{E}_{J,i_l,j'}[\mathrm{I}(r_{j'}|R_J; r_{i_l})] \geq \lambda/4$. Thus, recalling that $\mathrm{I}(r_{j'}|R_J; r_{i_l}) = \mathrm{H}(r_{j'}|R_j) - \mathrm{H}(r_{j'}|R, r_{i_l})$, we can write

$$\mathbb{E}_{J',j}[\mathrm{I}(r_j; R_{J'})] = \mathbb{E}_{J',j}[\mathrm{H}(r_j) - \mathrm{H}(r_j|R_{J'})] =$$
$$\mathbb{E}_j[\mathrm{H}(r_j)] + \mathbb{E}_{J',j,i_M}[-\mathrm{H}(r_j|R_{J'}) + \mathrm{H}(r_j|R_{J'} \setminus \{r_{i_M}\})]$$
$$+\mathbb{E}_{J',j,i_M,i_{M-1}}[-\mathrm{H}(r_j|R_{J'} \setminus \{r_{i_M}\}) + \mathrm{H}(r_j|R_{J'} \setminus \{r_{i_M}, r_{i_{M-1}}\})]$$
$$\cdots$$
$$+\mathbb{E}_{J',j,i_M,\ldots,i_1}[-\mathrm{H}(r_j|R_{J'} \setminus \{r_{i_l}\}_{l\in[2,M]}) + \mathrm{H}(r_j|R_{J'} \setminus \{r_{i_l}\}_{l\in[M]})]$$
$$-\mathbb{E}_{J',j,i_M,\ldots,i_1}[\mathrm{H}(r_j|R_{J'} \setminus \{r_{i_l}\}_{l\in[M]]})]$$
$$\geq M \cdot \lambda/4 + \mathbb{E}_{J',j,i_M,\ldots,i_1}[\mathrm{H}(r_j) - \mathrm{H}(r_j|R_{J'} \setminus \{r_{i_l}\}_{l\in[M]}) \geq M \cdot \lambda/4.$$

Now, these indices J' will correspond to the parties that $\mathcal{A}$ will corrupt in the execution. However, if some randomly chosen index i_l is indeed the index i corresponding to the only honest party P_i of $\mathcal{C}_1$, the attack will fail, as $\mathcal{A}$ cannot corrupt P_i. Yet, the probability that this happens corresponds to the probability that if we pick M items at random from $N-2n+1$ total items, i is not one of them, which is equal to: $\frac{\binom{N-2n}{M}}{\binom{N-2n+1}{M}} = 1 - \frac{M}{N-2n+1} = 1/2$, since we choose $M = (N-2n+1)/2$.

So, if $\mathbb{E}[|r_j|] \leq (N-2n+1)\cdot\lambda/8$ as in the Theorem statement, in expectation, $\mathcal{A}$ can sample r_j conditioned on $R_{J'}$ correctly (i.e., with probability 1) and guess $c_{i,j}$ with probability greater than $2^{-\lambda/2-1}$. Using the guessed r_j and $c_{i,j}$ along with the learned $\{c_{i',j}\}_{i'\in[n]\setminus\{i\}}$ via corruptions, we can reconstruct B_j. Finally, using $\{B_j\}_{j\in[n+1,2n]}$, we can successfully reconstruct x.

Thus, it cannot be true that $\Pr_{\mathcal{C}_1,\mathcal{C}_2,i,j}[|c_{i,j}| < \lambda/2] > 1/2$. By the law of total probability, it must therefore be that:

$$\mathbb{E}_{\mathcal{C}_1,\mathcal{C}_2}[\mathsf{Comm}] = \sum_{i,j} \mathbb{E}_{\mathcal{C}_1,\mathcal{C}_2}[|c_{i,j}|]$$

$$\geq \sum_{i,j} \lambda/2 \cdot \Pr_{\mathcal{C}_1,\mathcal{C}_2}[|c_{i,j}| \geq \lambda/2] = \lambda/2 \cdot \sum_{i,j} \Pr_{\mathcal{C}_1,\mathcal{C}_2}[|c_{i,j}| \geq \lambda/2] \geq \frac{n^2\cdot\lambda}{4}.$$

□

7 Conclusions and Future Work

The fluid model represents an alternative MPC setting that aims at tolerating certain relevant networking settings, and we believe that an interesting line of research is the study of what is possible in the fluid setting with respect to the non-fluid scenario. Our protocols from Sections 5 and 4 show that fluidity does not come with a price in terms of communication complexity for the honest and dishonest majority cases, as they achieve the same communication as non-fluid works, i.e. $O(n)$ communication per gate where n is the number of parties online at a time. There are other properties whose feasibility in the fluid setting are worth exploring. We mention some of them below.

It would be interesting to explore the cost of tolerating *mixed adversaries* in the fluid setting. A mixed adversary, defined and studied in [FHM98b], can simultaneously corrupt some parties passively and some other parties actively, and it is an attractive question to investigate how such results for non-fluid MPC translate to the fluid case. Other future directions could include the development of advanced primitives such as secure truncation or comparison to support complex applications (such as machine learning), in the fluid setting.

Another direction is to close the gap between fluid and non-fluid protocols by achieving stronger security notions such as identifiable abort for the dishonest majority case, or guaranteed output delivery (GOD) for honest majority which are not explored by prior fluid works [RS22, CGG$^+$21]. We believe this is an interesting direction since the *dispute control* technique of existing honest majority protocols with GOD is not very compatible with the fluid setting (at least with maximal or small fluidity). To elaborate further on this, GOD protocols (*cf.* [GSZ20] and the references therein) split the computation into segments and require correctness checks at the end of each segment. In case of failure, multiple "accuse" rounds lead to a pair of "disputed parties" being identified, and then the segment is repeated once again in such a way that an existing dispute cannot occur again. Given that in the fluid setting a given committee can only send messages to the next committee, which is announced on the fly, disputes of accusing each other are not well defined. Even if a dispute is found by a future committee, the accused parties may not be online anymore, which voids the argument that no new disputes will occur, and the protocol may stall forever. It seems that techniques which do not necessarily resemble those used in the information theoretic non-fluid model are required to achieve G.O.D. in the fluid model when there are no assumptions on the selection of committees.

Acknowledgments

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained

References

BDOZ11. Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

Bea92. Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

BENO19. Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online spdz! improving spdz using function dependent preprocessing. In *International Conference on Applied Cryptography and Network Security*, pages 530–549. Springer, 2019.

BGIN20. Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In *Advances in Cryptology – ASIACRYPT 2020*, pages 244–276, Cham, 2020. Springer International Publishing.

BH78. J. Bretagnolle and C. Huber. Estimation des densités : Risque minimax. In C. Dellacherie, P. A. Meyer, and M. Weil, editors, *Séminaire de Probabilités XII*, pages 342–363, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

BJMS20. Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure mpc: laziness leads to god. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 120–150. Springer, 2020.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

CCD87. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, page 462, 1987.

CGG$^+$21. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid mpc: secure multiparty computation with dynamic participants. In *Annual International Cryptology Conference*, pages 94–123. Springer, 2021.

CGH$^+$18. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority mpc for malicious adversaries. In *Annual International Cryptology Conference*, pages 34–64. Springer, 2018.

DEP21. Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. *Cryptology ePrint Archive*, 2021.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 445–465, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*, pages 572–590. Springer, 2007.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

EGPS22. Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Turbopack: Honest majority MPC with constant online communication. ACM Conference on Computer and Communications Security (CCS), 2022.

FHM98a. Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multi-party computation. In *Annual International Cryptology Conference*, pages 121–136. Springer, 1998.

FHM98b. Matthias Fitzi, Martin Hirt, and Ueli Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In *Annual International Cryptology Conference*, 1998.

GHK$^+$21a. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: you only speak once. In *Annual International Cryptology Conference*, pages 64–93. Springer, 2021.

GHK$^+$21b. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO 2021*, 2021.

GIP$^+$14. Daniel Genkin, Yuval Ishai, Manoj M. Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 495–504, New York, NY, USA, 2014. ACM.

GLO$^+$21. Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. Atlas: efficient and scalable mpc in the honest majority setting. In *Annual International Cryptology Conference*, pages 244–274. Springer, 2021.

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

GPS19. Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019,* *Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 499–529. Springer, 2019.

GS20. Vipul Goyal and Yifan Song. Malicious security comes free in honest-majority mpc. Cryptology ePrint Archive, Report 2020/134, 2020. https://eprint.iacr.org/2020/134.

GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, pages 618–646. Springer, 2020.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid mpc for dishonest majority. *CRYPTO*, 2022.

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

Supplementary Material

A Modelling Fluid MPC

We first recall the modelling of Fluid MPC from [RS22, CGG$^+$21]. As in the above works, we consider the *client-server* model, where there is a universe $\mathcal{U}$ of parties, that includes both the clients and servers. The goal of these clients is to *privately* compute a function over their inputs. The clients delegate this computation to a set of servers in $\mathcal{U}$ that can volunteer their computational resources for *part* of the computation and then potentially go offline. That is, the set of servers is not fixed in advance, and can change from time to time.

Computation proceeds in three or four stages: preprocessing (optional), input, execution, and output. Preprocessing is optional and typically only required to have a statistically-secure execution phase for the dishonest majority setting (see below). In the *preprocessing stage*, all clients and servers in $\mathcal{U}$ interact to generate information that will be used in the execution stage, but that is independent of the actual inputs and the function to be computed (it may be required that *enough* information is generated for some particular function). After the preprocessing stage, servers can go offline until the clients wish to perform the computation. In the *input stage*, clients process their inputs and hand these (private) versions to the servers for computation. In the *execution stage*, only the servers participate to compute the function. The execution stage proceeds in epochs, where each epoch i runs among a fixed set of servers, or committee $\mathcal{C}_i$. An epoch contains two parts, the computation phase, where the committee performs some computation local to itself, followed by a hand-off phase, where the current committee securely transfers some current state to the next committee. Finally, in the *output stage*, the last server committee transfers some final state to the clients, who then interact to reconstruct the output of the function. We stress that there is only *one output stage*, i.e., the clients get some final state from the servers once that allows them to reconstruct the entire output all at that time. We assume that all parties have access to only point-to-point channels.

Fluidity. Both the computation phase and hand-off phase of each epoch in the execution stage may require multiple rounds of interaction. *Fluidity* is defined as the minimum number of rounds in any given epoch of the execution stage. We say a protocol achieves *maximal fluidity* if each epoch i only lasts for one total round. I.e., the computation phase only consists of local computation by the parties in committee $\mathcal{C}_i$, and the hand-off phase consists of only some local computation by the parties in $\mathcal{C}_i$, plus communication from $\mathcal{C}_i$ to $\mathcal{C}_{i+1}$. In this paper, we only consider maximal fluidity, as it is the optimal setting to consider and it is the setting considered in the previous works [RS22, CGG$^+$21]. However, we stress that in our modelling for maximal fluidity (as well as that of [RS22, CGG$^+$21]) the clients in the output stage *may interact for a constant number of rounds* (i.e., independent of the circuit depth) to reconstruct the output.

Committee formation. The committees used in each epoch may either be fixed ahead of time, or chosen on-the-fly throughout the execution stage. While fixing committees ahead of time may result in a simpler, more efficient protocol, we focus on the less restrictive, more realistic setting where committees are chosen on-the-fly. This model is more suitable for the goal of making MPC protocols adequate for use over unstable networks since, intuitively, a given committee has better chances of guaranteeing a stable connection if they do not need to commit to a specific online time far in advance. See [CGG$^+$21] for more motivation and details on committee selection.

The model of [CGG$^+$21] specifies the formation process via an ideal functionality that samples and broadcasts committees according to the desired mechanism. However, as in [RS22], we desire to divorce the study of committee selection from the actual MPC and simply require that all parties of the current committee $\mathcal{C}_i$ somehow agree on the next committee $\mathcal{C}_{i+1}$. Specifically, the parties of committee $\mathcal{C}_i$ during the hand-off phase of epoch i (and not before) are informed by the environment $\mathcal{Z}$ of its choice of committee $\mathcal{C}_{i+1}$ (i.e., it is a worst-case choice by $\mathcal{Z}$). We make no assumptions or restrictions on the size of committees nor the overlap between committees. In particular, committees may consist of a large number (possibly constant fraction) of parties in the entire universe, $\mathcal{U}$.

Corruptions. We study two different settings for the number of parties that may be corrupted for our model to still require security:

– For *honest majority*, the adversary $\mathcal{A}$ may only corrupt any minority of servers in the committee of each epoch.[13] This is the setting that [CGG$^+$21] studies.
– For *dishonest majority*, the adversary $\mathcal{A}$ may corrupt all-but-one client and all-but-one server in the committee of each epoch. This is the setting that [RS22] studies.

More formally, we consider a *malicious R-adaptive adversary* from [CGG$^+$21] and used in [RS22]. For the dishonest majority case in this model, the adversary first *statically* chooses some parties to corrupt during the preprocessing phase. At this point, the only restriction on these corruption is that there must be at least one honest party; however, we will below strengthen this restriction. For both the honest and dishonest majority cases, when the clients are chosen, the adversary *statically* corrupts a set $\mathcal{T}_{\mathcal{C}_0} \subseteq \mathcal{C}_0$ of clients (at the start of the protocol). Then, the adversary corrupts the servers of the committees in an *adaptive* manner with *retroactive* effect. More specifically, in each epoch i, after learning which servers are in committee $\mathcal{C}_i$, the adversary can adaptively choose to corrupt a set of servers $\mathcal{T}_{\mathcal{C}_i} \subseteq \mathcal{C}_i$. Upon corrupting a server (resp. client), $\mathcal{A}$ learns its entire past state and can send messages on its behalf in epoch i (resp. the input and output stages). Therefore, when counting the number of corruptions for some epoch i, we must retroactively include those servers in committee $\mathcal{C}_i$ that are corrupted in some later epoch $j > i$. For each committee $\mathcal{C}_i$, we denote its size as $n_i := |\mathcal{C}_i|$

[13] All-but-one client could be corrupted, however.

and the number of corruptions as $t_i := |\mathcal{T}_{\mathcal{C}_i}|$. For the honest majority setting, it must be that $t_i < n_i/2$. For the dishonest majority setting, the only requirement is that $t_i < n_i$. Note that in the dishonest majority setting, we also count a server in committee $\mathcal{C}_i$ as corrupted if they were corrupted during the preprocessing phase. In the following, we will refer to the honest parties of $\mathcal{C}_i$ as $\mathcal{H}_{\mathcal{C}_i}$.

B Related Work

Fail-stop adversaries. A series of works have studied the setting of MPC, where the adversary is allowed to not only corrupt some parties passively/actively, but also cause some parties to fail (e.g. [FHM98a] and subsequent works). This can be seen as similar to the Fluid setting, where parties who participate in one committee may never participate again in another committee. However, one main difference is that unlike in the committee approach of Fluid, the set of parties that fail and thus exit the computation are not known to the rest of the parties. Second, and most crucially, once a party is set to fail by the adversary, it does not return to the computation, whereas parties in Fluid can arbitrarily be placed in several non-consecutive committees.

LazyMPC. The work of [BJMS20] considers an adversary that can set parties to be offline in any round (called "honest but lazy" in that work). This work differs from ours in several places. First, the authors focus only on the case of computational security, making use of rather strong techniques such as multi-key fully homomorphic encryption. Second, the parties that are chosen to be "lazy" are not known to the other parties. Third, once a party becomes offline, or "lazy", in their model it is assumed not to come back.

Synchronous but with partition tolerance. Recently, the work of [GPS19] designed MPC protocol in the so-called "sleepy model", which enables some of the parties to lag behind the protocol execution, while not being marked as corrupt. This could be achieved with an asynchronous protocol, naturally, but the main result of [GPS19] is obtaining such protocols without the strong threshold assumptions required to obtain asynchronous protocols. In particular, the authors obtain computationally secure constant-round protocols, assuming that the set of "fast"-and-honest parties in every round constitutes as majority, an assumption that is shown to be necessary.

Phoenix. The work of [DEP21] proposes a model that is similar to the one in [GPS19] in that parties can go offline for momentaneous periods of time, but unlike [GPS19], the parties are not assumed to receive messages while they are offline ([GPS19] considers unstable parties as "slow", meaning they still receive messages but they might not do so on time; in contrast, [DEP21] considers these parties to be potentially entirely offline). The work proposes solutions in their "Phoenix" model for MPC with perfect, statistical and computational security, and prove exact conditions on the adversary under which these are possible.

YOSO. In the recent work of Gentry et al. [GHK$^+$21a], the "You Only Speak Once" model for MPC is introduced. In this model, the basic assumption is that the adversary is able to corrupt a party as soon as that party sends a message. The YOSO model breaks the computation into small atomic pieces called *roles* where a role can be executed by sending only one message. The responsibility of executing each role is assigned to a physical party in a randomized fashion. The assumption is that this will prevent the adversary from targeting the relevant party until it sends its (single) message. This means that one should think of the entire set of parties as one "community" which as a whole is able to provide secure computation as a service. However, unlike our work, Yoso requires all parties to be online at all times. In a sense, YOSO aims to make progress and keep the computation alive without any guarantees for particular physical parties such as contributing inputs and receiving the output, This makes good sense in the context of a blockchain, for instance. Moreover, the demand that the MPC protocol must be broken down into roles makes protocol design considerably harder, particularly for information theoretically secure protocols. An additional caveat with the YOSO model is that one can only have information theoretically or statistically secure protocols assuming that the role assignment mechanism is given as an ideal functionality, and an implementation of such a mechanism must inherently be only computationally secure. In comparison, our model assumes a somewhat less powerful adversary who must allow a physical party to come back after being offline, if they desire. This allows for much easier protocol design, information theoretic security based only on point-to-point secure channels, and allows termination such that all parties can provide input and get output.