Perfectly Secure Fluid MPC with Abort and Linear Communication Complexity

Alexander Bienstock, Daniel Escudero and Antigoni Polychroniadou

J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, New York, USA

Abstract. The *Fluid* multiparty computation (MPC) model, introduced in (Choudhuri *et al.* CRYPTO 2021), addresses dynamic scenarios where participants can join or leave computations between rounds. Communication complexity initially stood at $\Omega(n^2)$ elements per gate, where n is the number of parties in a committee online at a time. This held for both statistical security (honest majority) and computational security (dishonest majority) in (Choudhuri *et al.* CRYPTO'21) and (Rachuri and Scholl, CRYPTO'22), respectively. The work of (Bienstock *et al.* CRYPTO'23) improved communication to $O(n)$ elements per gate. However, it's important to note that the perfectly secure setting with one-third corruptions per committee has only recently been addressed in the work of (David *et al.* CRYPTO'23). Notably, their contribution marked a significant advancement in the Fluid MPC literature by introducing guaranteed output delivery. However, this achievement comes at the cost of prohibitively expensive communication, which scales to $\Omega(n^9)$ elements per gate.

In this work, we study the realm of perfectly secure Fluid MPC under one-third active corruptions. Our primary focus lies in proposing efficient protocols that embrace the concept of security with abort. Towards this, we design a protocol for perfectly secure Fluid MPC that requires only *linear* communication of $O(n)$ elements per gate, matching the communication of the non-Fluid setting. Our results show that, as in the case of computational and statistical security, perfect security with abort for Fluid MPC comes "for free" (asymptotically linear in n) with respect to traditional non-Fluid MPC, marking a substantial leap forward in large scale dynamic computations, such as Fluid MPC.

1 Introduction

In the setting of secure multiparty computation (MPC), multiple parties collaborate to compute a function on secret inputs, ensuring that only the output is revealed, even when an adversary corrupts an unknown subset of parties of a certain size. Over the past four decades, the field of MPC has undergone remarkable evolution, leading to highly efficient protocols. What's even more compelling is that MPC has been successfully applied to to address privacy and security concerns across various real-world applications. For a view of these practical implementations, see several real-world deployments at https://mpc.cs.berkeley.edu/.

MPC protocols are intricate in design and typically demand a high degree of reliability from the honest parties. These parties are expected to maintain continuous online presence without disruptions or crashes, and the set of parties should remain constant throughout the process. In certain scenarios, the set of parties is more dynamic, with individual parties being more susceptible to connection drops or crashes. A prime illustration of

This article appears in Volume 1 (2024), Issue 4 of the IACR Communications in Cryptology Journal.
E-mail: alex.bienstock@jpmchase.com (Alexander Bienstock), daniel.escudero@protonmail.com (Daniel Escudero), antigonipoly@gmail.com (Antigoni Polychroniadou)

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2024-12-24.

such cases is found in permissionless blockchains, where parties may ephemerally join a given execution, but may leave, crash or drop arbitrarily, either voluntarily or because of adverse conditions. Furthermore, this should *not* be considered adversarial behavior: truly corrupt parties may send incorrect messages and try to cheat in the protocol, while actions like dropping or crashing can happen even to a genuinely honest party. Just as in the cases mentioned, MPC applications can also foresee dynamic participant sets where honest parties might experience disruptions, like drops or crashes. For instance, in computations with a *large* number of participants using commodity hardware, such disruptions can be expected. Moreover, within these highly distributed scenarios with numerous geographically dispersed participants, MPC protocols for even moderately complex tasks, such as federated learning [BIK$^+$17; BBG$^+$20; MWA$^+$23], consume substantial resources. In such cases, parties, even if considered stable, may prefer to participate only during specific computation phases. This underscores the necessity to accommodate dynamic settings, where parties join or depart at will, whether by choice or circumstance, such as in the model of Fluid MPC [CGG$^+$21]. See Section B of [BEP23] for an overview of other dynamic settings in MPC [FHM98; GPS19; BJMS20; GHK$^+$21; DEP21].

In the fluid model, there is a "global pool" of parties willing to participate in the computation, and each party is assigned to one or more committees which take care of executing a given step of the computation. Ideally, each committee should only be in charge of performing a small portion of the computation, in order to avoid committing to resources for a long period of time. This is captured by the concept of *fluidity*, which corresponds to the amount of rounds that each committee runs. In the maximal fluidity setting, which is the one studied by previous works, each committee is only required to execute *one* round: they only need to receive some messages from the previous committee, perform some local computation, and send messages to the next committee in line. This models the most fatalistic case in terms of parties' dynamics: a party can belong to only one committee, joining only to receive messages from the previous committee and dropping after sending messages to the next, and a maximally Fluid MPC protocol still guarantees security in this setting. Let n be the number of parties in each committee (which can be variable across committees, but we restrict to same-size committees for simplicity), and let t be the number of corrupted parties in each committee. The three common settings in traditional MPC are *computational* security (for arbitrary $t < n$), *statistical* security (for honest majority $t < n/2$) and *perfect* security (for $t < n/3$). The same taxonomy exists in the *Fluid* context. For arbitrary $t < n$ and computational security with abort, the work of [RS22] introduces a Fluid protocol with quadratic communication $O(n^2)$ per multiplication, which is improved to linear $O(n)$ by [BEP23]. The work [BEP23] also achieves linear communication for statistical security with abort and $t < n/2$, improving over the quadratic costs from [CGG$^+$21] in this setting. Note that both for computational and statistical security in traditional (*i.e.* non-fluid) MPC, protocols with linear communication are known [GSZ20; DPSZ12], so the work [BEP23] effectively shows that the fluid model in these security settings does not add communication overhead, asymptotically.

For perfect security and $t < n/3$, the work by [DDG$^+$23] introduces a perfectly secure Fluid MPC protocol based on the BGW paradigm with verifiable secret sharing. Their focus is on *guaranteed output delivery* (G.O.D.),[1] where the adversary cannot prevent honest parties from getting the correct output, and in that setting they obtained an impractically high communication cost of $\Omega(n^9)$ per multiplication gate. In contrast, in the non-fluid regime, MPC with such guarantees is achievable with *linear* communication [BH08], which shows that for perfect security the current gap in terms of efficiency between the Fluid and non-fluid worlds is too high. Hence, the natural question that remains open and needs to be addressed is the following:

[1]GOD is also possible for statistical security in non-fluid MPC [GSZ20], although it remains unclear how to adapt existing protocols to the Fluid setting. All previous statistically secure Fluid protocols satisfy security with abort. More on this difficulty in Remark 1.

Table 1: Existing works in the Fluid setting and their communication complexity.

Protocol	Comm. complexity	Threshold	Security	Output guarantees
[RS22] [BEP23]	$\Theta(n^2\|C\|)$ $\Theta(n\|C\| + \mathsf{depth}(C)n^2)$	$t < n$	comp.	w. abort
[CGG+21] [BEP23]	$\Theta(n^2\|C\|)$ $\Theta(n\|C\| + \mathsf{depth}(C)n^2)$	$t < n/2$	stat.	w. abort
[DDG+23] **Ours**	$\Theta(n^9\|C\|)$ $\Theta(n\|C\| + \mathsf{depth}(C)n^2)$	$t < n/3$	perfect	G.O.D. w. abort

Is it possible to obtain perfectly secure maximally-fluid MPC protocols for $t < n/3$, with linear communication complexity per gate?

1.1 Our Contribution

In our work we address precisely this question, providing an answer in the *affirmative*. We show that, by adopting the notion of *security with abort* instead of GOD (which is the property achieved by all previous fluid MPC protocols, except [DDG+23]), it is possible indeed to design perfectly secure maximally-fluid protocols for $t < n/3$ with *linear* communication, hence matching the asymptotic communication of the non-fluid setting. For a comprehensive view of our contributions in relation to existing works in the Fluid MPC domain, please refer to Table 1, where we provide a comparative overview of the literature landscape. Our work shows that, even under the stringent conditions of maximal fluidity, it is possible to achieve perfectly secure MPC without giving up on linear communication.

In the following, we let C be a *layered* arithmetic cicuit 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 $|C|/\mathsf{depth}(C) = \Omega(n)$, so that the term $\mathsf{depth}(C)n^2$ is absorbed by $n|C|$. One way this is satisfied is if the circuit has uniform width $\Omega(n)$. We summarize our result below.

Theorem 1. *(Informal) Let n be a positive integer. For a layered arithmetic circuit C that computes an n-ary functionality $\mathcal{F}$, there exists an n-party Fluid protocol with maximal fluidity that securely computes C with perfect security against an active adversary who can control up to $t < n/3$ corrupted parties. The protocol has total communication $O(n|C| + \mathsf{depth}(C)n^2)$ elements.*

Similar to the work achieving linear communication efficiency in the fluid setting with statistical and computational security [BEP23], our protocol operates by harnessing multiplication triples (without preprocessing) to manage multiplication gates. Additionally, we adopt the "king idea" from [DN07] to facilitate the reconstructions within Beaver-based multiplication. These are the methods typically used in the non-fluid setting to achieve linear communication complexity, and they serve as a good starting point for the fluid case. However, in the fluid protocol from [BEP23] cheating is handled by transfering, from committee to committee, some form of *accumulator* that attests to the correctness of the computation. This "accumulator" is succinct (which is crucial for the efficiency claims), and if the adversary misbehaves in at least one multiplication gate, then this will be reflected in the accumulator with *overwhelming probability.* Unfortunately, for the perfectly secure case we cannot afford non-zero error probability, so the ideas in [BEP23] are of no use to us. Instead, we design new methods to ensure that either multiplications are carried out correctly, or parties abort. To achieve this we employ ideas from error-correcting codes, particularly parity-check matrices, in a novel way in the context of Fluid MPC. Interestingly, we rely on packed secret-sharing to transfer the state of the computation from one committee to the next, which is a technique that was not used in any prior Fluid MPC work and turns out to be pivotal for our complexity claim. This resharing protocol can be of independent interest. We provide a detailed technical overview in Section 1.2, which outlines the core ideas behind our construction, and further explains the inadequacy of the techniques of [BEP23] for the perfect setting.

Remark 1 (On GOD and identifiable abort in the maximally-Fluid setting.). In the non-fluid context, GOD is possible for $t < n/2$, hence particularly for $t < n/3$. However, the most common design pattern to obtain *efficient* protocols (cf. [BH08; GSZ20]) involves somehow identifying cheating parties, and falling back to a previous protocol state that is "safe". This recovery-based approach is not suitable for (maximally) Fluid MPC, where parties are possibly gone after performing one single computation step. Instead, a different approach that does not require "looking back" in the protocol execution is needed. One such approach is the one initiated in BGW [WOG88] for the non-fluid setting, which achieves high $O(n^6)$ communication, but avoids this "recovery" paradigm. This is precisely what the work of [DDG$^+$23] exploits: by basing their approach on BGW, they manage to obtain GOD for Fluid MPC with $t < n/3$, at the expense of very high communication overhead $\Omega(n^9)$ that is even orders of magnitude higher than BGW in the non-fluid setting.

Only recently, after a long and fruitful series of works [AAY21; AAPP23], a highly non-trivial protocol in the non-fluid regime with linear communication was designed, which satisfies GOD while avoiding "recoveries". Adapting these ideas to the fluid setting in order to achieve GOD with better communication than [DDG$^+$23] is interesting and challenging future work. Indeed, simply reducing the high communication overhead with respect to the non-fluid setting introduced by the fluid work of [DDG$^+$23] has many remaining barriers.

For similar reasons, namely that parties are possibly gone after a single round, so conflicts are hard to resolve, we believe that even achieving security with identifiable abort efficiently in the maximally-fluid setting is challenging.

1.2 Technical Overview

Let $\mathcal{C}_1, \mathcal{C}_2, \ldots$ be the committees involved, each having n parties and t corruptions, with $n = 3t+1$, and let $[x]_d^{\mathcal{C}_i}$ denote a Shamir secret sharing of value x over a finite field $\mathbb{F}$, using degree d, among the parties of committee $\mathcal{C}_i$. In our work, we will mostly work with degree $d = 2t$. As previous works in Fluid MPC, we assume that the circuit is *layered*, meaning that the multiplication gates in a given layer can only depend on the outputs of the layer inmediately before it. The resulting circuit has addition, multiplication, and *identity* gates for relaying values from one layer to the next one. First each client, who provides input $x \in \mathbb{F}$ to the computation, secret-shares their input towards the first committee $\mathcal{C}_1$ as $[x]_{2t}^{\mathcal{C}_1}$. At this point, the parties in $\mathcal{C}_1$ have sharings of the first layer of the computation, and the goal now is to let them evaluate the first layer of the circuit so that parties in a future committee get sharings of the outputs of this layer. This is then continued for each layer, until a certain committee obtains shares of the output layer, which can be reconstructed to the clients. From this general template, the core question becomes the following: design a protocol so that, starting from a committee $\mathcal{C}_i$ holding shares of two values $[x]_{2t}^{\mathcal{C}_i}$ and $[y]_{2t}^{\mathcal{C}_i}$, a future committee $\mathcal{C}_j$ with $j > i$ can learn the product $[xy]_{2t}^{\mathcal{C}_{i+1}}$. Such protocol enables processing multiplication gates, and our overview focuses mostly on illustrating how this is done in our work. Handling identity gates (and in fact also addition gates) boils down to one committee $\mathcal{C}_i$ holding $[x]_{2t}^{\mathcal{C}_i}$, transfering this to $\mathcal{C}_j$ so that they obtain $[x]_{2t}^{\mathcal{C}_j}$. This is a strictly easier task than multiplication and it is approached via a simplification of our multiplication protocol.

1.2.1 Challenges of our multiplication protocol.

For notational convenience let us relabel the committee who holds the initial sharings from $\mathcal{C}_i$ to $\mathcal{C}_{i+1}$, that is, committee $\mathcal{C}_{i+1}$ has two sharings $[x]_{2t}^{\mathcal{C}_{i+1}}$ and $[y]_{2t}^{\mathcal{C}_{i+1}}$, and the goal is to multiply each x with y. The general structure of our protocol is the following. At a high level, the idea is to let committee $\mathcal{C}_{i+1}$ use a multiplication triple $([a]_{2t}^{\mathcal{C}_{i+1}}, [b]_{2t}^{\mathcal{C}_{i+1}}, [c]_{2t}^{\mathcal{C}_{i+1}})$ where $c = ab$ to compute the product between x and y, as standard in Beaver-based multiplication. This consists of first computing locally the sharings $[d]_{2t}^{\mathcal{C}_{i+1}} = [x]_{2t}^{\mathcal{C}_{i+1}} + [a]_{2t}^{\mathcal{C}_{i+1}}$ and $[e]_{2t}^{\mathcal{C}_{i+1}} = [y]_{2t}^{\mathcal{C}_{i+1}} + [b]_{2t}^{\mathcal{C}_{i+1}}$, reconstructing d and e, and taking the linear combination $[xy]_{2t}^{*} = de - e[a]_{2t}^{*} - d[b]_{2t}^{*} + [c]_{2t}^{*}$. However, there are two complications with this approach. The first is that, because we are in the maximal fluidity setting, reconstruction cannot be done among the members of $\mathcal{C}_{i+1}$ themselves, but rather, towards the next committee. To make matters worse, reconstructing in one round (*i.e.* towards the immediate next committee $\mathcal{C}_{i+2}$) would involve *quadratic* communication (since this requires every party in $\mathcal{C}_{i+1}$ to send a share to every party in $\mathcal{C}_{i+2}$), which we cannot afford. To obtain linear communication we make use of the "multiple king idea" from [DN07] (explained in more detail later in the section), but this results in committee $\mathcal{C}_{i+3}$ learning the reconstructions of d and e, not $\mathcal{C}_{i+2}$. The second complication arises from the fact that the linear combination leading to xy described above must be computed by the committee that knows d and e, $\mathcal{C}_{i+3}$, but this committee must also have the triple $([a]_{2t}^{\mathcal{C}_{i+3}}, [b]_{2t}^{\mathcal{C}_{i+3}}, [c]_{2t}^{\mathcal{C}_{i+3}})$, which is currently held by committee $\mathcal{C}_{i+1}$ (this is why we put asterisks $*$ in the sharings when describing the linear combination). Solving this issue requires designing a method for committee $\mathcal{C}_{i+1}$ to transfer sharings to committee $\mathcal{C}_{i+3}$.

Finally, another aspect we have overlooked is the generation of the multiplication triple: committee $\mathcal{C}_{i+1}$ needs to obtain $([a]_{2t}^{\mathcal{C}_{i+1}}, [b]_{2t}^{\mathcal{C}_{i+1}}, [c]_{2t}^{\mathcal{C}_{i+1}})$ in a first place. For this, we will let committee $\mathcal{C}_{i+1}$ first get $[a]_{t}^{\mathcal{C}_{i+1}}$ and $[b]_{t}^{\mathcal{C}_{i+1}}$ (notice the degree t instead of $2t$), from which the parties in $\mathcal{C}_{i+1}$ can derive *locally* $[ab]_{2t}^{\mathcal{C}_{i+1}} = [a]_{t}^{\mathcal{C}_{i+1}} \cdot [b]_{t}^{\mathcal{C}_{i+1}}$. Since any degree-$t$ sharing is also a degree-$2t$ sharing, the parties in $\mathcal{C}_{i+1}$ interpret $[a]_{t}^{\mathcal{C}_{i+1}}$ as $[a]_{2t}^{\mathcal{C}_{i+1}}$ (and similarly for b). This way, the parties have obtained the required triple. An avid reader may note that the computed $[ab]_{2t}^{\mathcal{C}_{i+1}}$ is *not* a random degree-$2t$ sharing of $c = a \cdot b$, since the underlying polynomial is not random but rather the product of two degree-t polynomials. Also, $[a]_{2t}^{\mathcal{C}_{i+1}}$ (and same for b) is not a random degree-$2t$ sharing, since the underlying polynomial has degree t. However, as we will see, this turns out to not be a problem, and the randomization involved when re-sharing helps us prevent leakage from this underlying structure.

One final question left is how committee $\mathcal{C}_{i+1}$ gets $[a]_{t}^{\mathcal{C}_{i+1}}$ (and $[b]_{t}^{\mathcal{C}_{i+1}}$) in a first place. For this, we use committee $\mathcal{C}_i$: the parties in $\mathcal{C}_i$ execute the random sharing generation protocol from [BH08], except with the receivers in $\mathcal{C}_{i+1}$. We provide more details below.

1.2.2 On the inefficacy of the techniques of [BEP23] in the perfect setting.

The *statistically-secure* Fluid MPC protocol of [BEP23] faces a similar challenge as above in that a committee $\mathcal{C}_{i+1}$ holding sharings of a beaver triple $([a]_{2t}^{\mathcal{C}_{i+1}}, [b]_{2t}^{\mathcal{C}_{i+1}}, [c]_{2t}^{\mathcal{C}_{i+1}})$ must somehow transfer them to committee $\mathcal{C}_{i+3}$. To do so, they develop a technique which allows committee $\mathcal{C}_{i+1}$ to efficiently reshare any sharing $[x]_{2t}^{\mathcal{C}_{i+1}}$ to committee $\mathcal{C}_{i+3}$. For this, each party P_j in committee $\mathcal{C}_{i+1}$ sends (a rerandomized version of) their share to the corresponding j-th party in committee $\mathcal{C}_{i+2}$, who then sends (a rerandomized version of) their share to the corresponding j-th party in committee $\mathcal{C}_{i+3}$ (which achieves linear communication). As will be the case for the perfect setting, the adversary can introduce errors in their protocol. However, as discussed above, their protocol uses a kind of *accumulator* that attests to the correctness of the resharing to catch cheating. This accumulator is succinct (which is crucial for the efficiency claims), and if the adversary misbehaves, then they will be caught with *overwhelming probability.*

One would hope that we could borrow the above techniques for the perfect security setting, but it turns out that we cannot for a few reasons. First, we cannot use such a compressed accumulator to catch the adversary with all-but-negligible probability, since we need perfect security. Second, we cannot use any kind of error detection guaranteed by Shamir secret sharing to catch the adversary cheating, since the adversary gets to control parties in multiple committees, and thus it gets to control as many as $2t$ shares at a time. Also note that if the degree were lower than $2t$, then privacy would be violated, since the adversary can learn $2t$ shares. This is why we design our own resharing mechanism with linear communication, that we will present below.

1.2.3 Parties in $\mathcal{C}_i$ generate random sharings towards $\mathcal{C}_{i+1}$.

Our multiplication protocol operates in *batches*: $t+1$ products are handled simultaneously. Committee $\mathcal{C}_{i+1}$ has multiple sharings $[x_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [x_{t+1}]_{2t}^{\mathcal{C}_{i+1}}$ and $[y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{t+1}]_{2t}^{\mathcal{C}_{i+1}}$, and the goal is to multiply each x_α with y_α for $\alpha \in [t+1]$.

For the multiplication, committee $\mathcal{C}_{i+1}$ needs random sharings $\{[a_\alpha]_t^{\mathcal{C}_{i+1}}, [b_\alpha]_t^{\mathcal{C}_{i+1}}\}_{\alpha\in[t+1]}$ which will be used for producing multiplication triples. For this, we use the standard approach from [BH08] in which each party samples and distributes some random sharings, which are then combined with an appropriate matrix to obtain truly random sharings. However, since parties in $\mathcal{C}_{i+1}$ cannot communicate with each other, they must receive these sharings from $\mathcal{C}_i$. In more detail, to generate a set of *t+1* random sharings $[a_\alpha]_t^{\mathcal{C}_{i+1}}$ for $\alpha \in [t+1]$, each party $P_j \in \mathcal{C}_i$ samples a random $s_j \in \mathbb{F}$ and distributes $[s_j]_t^{\mathcal{C}_{i+1}}$ towards $\mathcal{C}_{i+1}$.

The next step is for the parties in $\mathcal{C}_{i+1}$ to *locally* apply a hyper-invertible matrix to $([s_1]_t^{\mathcal{C}_{i+1}}, \dots, [s_n]_t^{\mathcal{C}_{i+1}})$, obtaining $([a_1]_t^{\mathcal{C}_{i+1}}, \dots, [a_n]_t^{\mathcal{C}_{i+1}})$. These matrices, defined in [BH08], have several important properties that simultaneously help with randomness extraction (since $2t+1$ random sharings are input, just as many can be extracted) and cheating verification, which is used to handle the fact that the parties in $\mathcal{C}_i$ may distribute incorrect random sharings (*e.g.* using incorrect degree); for the latter, a check similar to [BH08] is performed (see more below). The sharings that the parties in $\mathcal{C}_{i+1}$ will finally use are the first $t+1$ sharings $([a_1]_t^{\mathcal{C}_{i+1}}, \dots, [a_{t+1}]_t^{\mathcal{C}_{i+1}})$, and $([b_1]_t^{\mathcal{C}_{i+1}}, \dots, [b_{t+1}]_t^{\mathcal{C}_{i+1}})$ produced in a similar way.

1.2.4 Parties in $\mathcal{C}_{i+1}$ use the random sharings.

These sharings are used to obtain Beaver triples: $[a_\alpha]_t^{\mathcal{C}_{i+1}}$ and $[b_\alpha]_t^{\mathcal{C}_{i+1}}$ can be *locally* multiplied to obtain $[c_\alpha]_{2t}^{\mathcal{C}_{i+1}}$. From now on $[a_\alpha]_t^{\mathcal{C}_{i+1}}$ and $[b_\alpha]_t^{\mathcal{C}_{i+1}}$ are interpreted as $[a_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ and $[b_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, respectively. Then the parties in $\mathcal{C}_{i+1}$ execute the first step of Beaver-based multiplication: they add locally $[d_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [x_\alpha]_{2t}^{\mathcal{C}_{i+1}} + [a_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ and $[e_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [y_\alpha]_{2t}^{\mathcal{C}_{i+1}} + [b_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, and then the goal is to reconstruct these values. Of course, since the members in committee $\mathcal{C}_{i+1}$ cannot talk to each other in the maximal fluidity setting, these parties would reconstruct these values towards committee $\mathcal{C}_{i+2}$. This is done via the standard efficient reconstruction procedure from [DN07], which consists of the parties first expanding the sharings $\{[d_\alpha]_{2t}^{\mathcal{C}_{i+1}}\}_{\alpha\in[t+1]}$ with an error correcting code, instantiated by multiplication with a super-invertible matrix (by definition, a linear operation), to obtain $\{[d'_\beta]_{2t}^{\mathcal{C}_{i+1}}\}_{\beta\in[n]}$ (and similarly $\{[e'_\beta]_{2t}^{\mathcal{C}_{i+1}}\}_{\beta\in[n]}$); this is followed by all parties in $\mathcal{C}_{i+1}$ sending their shares of $[d'_k]_{2t}^{\mathcal{C}_{i+1}}$ and $[e'_k]_{2t}^{\mathcal{C}_{i+1}}$ to each $P_k \in \mathcal{C}_{i+2}$, who reconstruct them.

At this point the parties in $\mathcal{C}_{i+2}$ have "shares" $(d'_1, \dots, d'_n)$ and $(e'_1, \dots, e'_n)$, which together make up codewords encoding $(d_1, \dots, d_{t+1})$ and $(e_1, \dots, e_{t+1})$ respectively. These need to be "reconstructed", so these "sharings" are sent to committee $\mathcal{C}_{i+3}$, who can decode them to learn $(d_1, \dots, d_{t+1})$ and $(e_1, \dots, e_{t+1})$. Committee $\mathcal{C}_{i+3}$ will execute Beaver-based multiplication, which works by taking the local linear combination

$$[x_\alpha y_\alpha]_{2t}^{\mathcal{C}_{i+3}} = d_\alpha e_\alpha - d_\alpha [b_\alpha]_{2t}^{\mathcal{C}_{i+3}} - e_\alpha [a_\alpha]_{2t}^{\mathcal{C}_{i+3}} + [c_\alpha]_{2t}^{\mathcal{C}_{i+3}}. \tag{1}$$

However, in order to do this the parties in $\mathcal{C}_{i+3}$ need the sharings of the triple $([a_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [b_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [c_\alpha]_{2t}^{\mathcal{C}_{i+3}})$.

1.2.5 Committee $\mathcal{C}_{i+3}$ gets the triple - Resharing protocol based on packed secret sharing.

Recall that the parties in $\mathcal{C}_{i+1}$ have $([a_\alpha]_{2t}^{\mathcal{C}_{i+1}}, [b_\alpha]_{2t}^{\mathcal{C}_{i+1}}, [c_\alpha]_{2t}^{\mathcal{C}_{i+1}})$. These sharings will be transferred to committee $\mathcal{C}_{i+3}$ via a resharing protocol. This is done in two steps where parties in $\mathcal{C}_{i+1}$ send packed secret sharings of their shares. Next, the parties in $\mathcal{C}_{i+2}$ can combine these packed sharings of shares into one packed sharing and then Shamir sharings of their packed shares are sent to $\mathcal{C}_{i+3}$ to get non-packed shares of the original secrets. See details below.

Committee $\mathcal{C}_{i+2}$ obtains $([\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}, [\![\boldsymbol{b}]\!]_{2t}^{\mathcal{C}_{i+2}}, [\![\boldsymbol{c}]\!]_{2t}^{\mathcal{C}_{i+2}})$. We denote a packed secret sharing value of vector $\boldsymbol{x} = (x_1, \dots, x_\ell) \in \mathbb{F}^\ell$ with degree d among the parties of $\mathcal{C}_i$ as $[\![\boldsymbol{x}]\!]_d^{\mathcal{C}_i}$. For committee $\mathcal{C}_{i+1}$ to send the triples to committee $\mathcal{C}_{i+3}$, these sharings have to pass through $\mathcal{C}_{i+2}$ first. Towards this, parties in $\mathcal{C}_{i+1}$ transfer these sharings to $\mathcal{C}_{i+2}$, but they do so with *packed secret-sharing*, which is crucial for efficiency. We focus on how to transfer $\{[a_\alpha]_{2t}^{\mathcal{C}_{i+1}}\}$ so that committee $\mathcal{C}_{i+2}$ gets $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$, with the other sharings handled in the same way.

Let us denote the j-th share of each $[a_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ by a_α^j, which satisfy $\sum_{j=1}^{2t+1} L_j(0) a_\alpha^j = a_\alpha$, where $L_j(\mathtt{X})$ are the Lagrange polynomials. Party $P_j \in \mathcal{C}_{i+1}$, having $\boldsymbol{a}_{[1,t+1]}^j = (a_1^j, \dots, a_{t+1}^j)$, distributes sharings $[\![\boldsymbol{a}_{[1,t+1]}^j]\!]_{2t}^{\mathcal{C}_{i+2}}$ to committee $\mathcal{C}_{i+2}$. At this point each party in this committee can compute locally

$$\sum_{j=1}^{2t+1} L_j(0) [\![\boldsymbol{a}_{[1,t+1]}^j]\!]_{2t}^{\mathcal{C}_{i+2}} = [\![\sum_{j=1}^{2t+1} L_j(0) \cdot \boldsymbol{a}_{[1,t+1]}^j]\!]_{2t}^{\mathcal{C}_{i+2}} = [\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}},$$

where $\boldsymbol{a} = (a_1, \dots, a_{t+1})$. Notice that a corrupt party $P_j \in \mathcal{C}_{i+1}$ may distribute $[\![\boldsymbol{a}_{[1,t+1]}^j]\!]_{2t}^{\mathcal{C}_{i+2}}$ incorrectly. For example, the underlying secret may not correspond to $\boldsymbol{a}_{[1,t+1]}^j$. We discuss how to address this later in the section but for now, let us hint that this is prevented by leveraging the fact that each row of the "matrix" $[\boldsymbol{a}_{[1,t+1]}^1 \| \cdots \| \boldsymbol{a}_{[1,t+1]}^n]$ has to be *consistent* with a polynomial of degree $\leq 2t$, and this bounds the adversary to use the correct secrets.

Committee $\mathcal{C}_{i+3}$ obtains $([a_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [b_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [c_\alpha]_{2t}^{\mathcal{C}_{i+3}})$. Here, the sharings $([\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}, [\![\boldsymbol{b}]\!]_{2t}^{\mathcal{C}_{i+2}}, [\![\boldsymbol{c}]\!]_{2t}^{\mathcal{C}_{i+2}})$ held by committee $\mathcal{C}_{i+2}$ are "unpacked" towards committee $\mathcal{C}_{i+3}$. As before, we focus on $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$ for the sake of exposition. Let us denote the share of party $P_j \in \mathcal{C}_{i+2}$ of $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$ by $\boldsymbol{a}^j \in \mathbb{F}$, which satisfy $a_\alpha = \sum_{j=1}^{2t+1} L_j(-\alpha) \cdot \boldsymbol{a}^j$. Each of these parties secret-shares their share as $[\boldsymbol{a}^j]_{2t}^{\mathcal{C}_{i+3}}$ towards committee $\mathcal{C}_{i+3}$. Due to the observation above, we have that

$$\sum_{j=1}^{2t+1} L_j(-\alpha) \cdot [\boldsymbol{a}^j]_{2t}^{\mathcal{C}_{i+3}} = [\sum_{j=1}^{2t+1} L_j(-\alpha) \cdot \boldsymbol{a}^j]_{2t}^{\mathcal{C}_{i+3}} = [a_\alpha]_{2t}^{\mathcal{C}_{i+3}},$$

so the parties in $\mathcal{C}_{i+3}$ can obtain the desired shares. A similar approach is followed to obtain $[b_\alpha]_{2t}^{\mathcal{C}_{i+3}}$ and $[c_\alpha]_{2t}^{\mathcal{C}_{i+3}}$.

1.2.6 Achieving active security.

The protocol sketched so far can be attacked by an active adversary at the following places:

1. The generation of the random sharings by committee $\mathcal{C}_i$ may be done inconsistently.
2. The parties in $\mathcal{C}_{i+1}$ may send incorrect shares of $[d'_k]_{2t}^{\mathcal{C}_{i+1}}$ or $[e'_k]_{2t}^{\mathcal{C}_{i+1}}$ to some $P_k \in \mathcal{C}_{i+2}$, or P_k may send an incorrect d'_k to $\mathcal{C}_{i+3}$.
3. A corrupt party $P_j \in \mathcal{C}_{i+1}$ may reshare $[\![\boldsymbol{a}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+2}}$ (or $[\![\boldsymbol{b}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+2}}$, or $[\![\boldsymbol{c}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+2}}$) inconsistently to committee $\mathcal{C}_{i+2}$.
4. A corrupt party $P_j \in \mathcal{C}_{i+2}$ may reshare $[\boldsymbol{a}^j]_{2t}^{\mathcal{C}_{i+3}}$ (or $[\boldsymbol{b}^j]_{2t}^{\mathcal{C}_{i+3}}$, or $[\boldsymbol{c}^j]_{2t}^{\mathcal{C}_{i+3}}$) inconsistently to committee $\mathcal{C}_{i+3}$.

The first item is addressed by using the hyper-invertible matrix verification from [BH08], where the last $2t$ mapped sharings $([a_{t+2}]_{t,2t}^{\mathcal{C}_{i+1}}, \dots, [a_n]_{t,2t}^{\mathcal{C}_{i+1}})$, held by the receiving $\mathcal{C}_{i+1}$, are opened (note that after this, the first $t+1$ mapped sharings are still random, since the adversary only sees at most t out of $2t+1$ random sharings that can be extracted). In our case, these are opened towards the last $2t$ parties in committee $\mathcal{C}_{i+2}$, who perform the check in [BH08]. Intuitively, as argued in [BH08], due to the invertibility properties of hyper-invertible matrices, the t sharings that are opened to honest parties of committee $\mathcal{C}_{i+2}$ have a one-to-one relationship with the t original sharings $[s_j]_{t,2t}^{\mathcal{C}_{i+1}}$ shared by the corrupted parties of committee $\mathcal{C}_i$. Thus, there is an inconsistency in the latter if and only if there is an inconsistency in the former, which can be detected. We present our random sharings protocol in Section 3.1. On the other hand, the second item is easily handled by performing error detection.

Items (3) and (4) require more care, and we provide an overview on these below.

Ensuring consistency of $[\![\boldsymbol{a}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+2}}$. A party $P_j \in \mathcal{C}_{i+1}$ may cheat by sending $[\![\boldsymbol{a}^j_{[1,t+1]} + \boldsymbol{\delta}^j]\!]_{2t}^{\mathcal{C}_{i+2}}$, where $\boldsymbol{\delta}^j \neq \mathbf{0}$.[2] First, let H be the $(n-2t-1) \times n$ parity-check matrix such that $H \cdot \boldsymbol{x} = \mathbf{0}$ if and only if $\boldsymbol{x} \in \mathbb{F}^n$ are valid shares of a polynomial of degree $\leq 2t$. The intuitive idea behind our check is simple. Let A be the $n \times (t+1)$ matrix whose j-th row is $\boldsymbol{a}^j_{[1,t+1]}$. We have that for any index $\alpha \in [t+1]$, the α-th column is a degree-$2t$ sharings. This means the matrix satisfies $H \cdot A = 0 \in \mathbb{F}^{(n-2t-1)\times(t+1)}$. Let D be the $n \times (t+1)$ matrix whose j-th row is $\boldsymbol{\delta}^j$ (we define $\boldsymbol{\delta}^j = \mathbf{0}$ for an honest party $P_j \in \mathcal{C}_{i+1}$). Since the parties in $\mathcal{C}_{i+2}$ *allegedly* have sharings of each row $[\![\boldsymbol{a}^j_{[1,t+1]} + \boldsymbol{\delta}^j]\!]_{2t}^{\mathcal{C}_{i+2}}$ of $A + D$, the parties can *locally* compute shares of each row of $H \cdot (A + D) = H \cdot A + H \cdot D = H \cdot D$. Then the parties in $\mathcal{C}_{i+2}$ reconstruct these sharings towards committee $\mathcal{C}_{i+3}$, and check that the corresponding secrets are all $\mathbf{0}$. It is possible to verify that, given that D only contains at most t non-zero rows, the only way in which $H \cdot D$ can equal zero is if $D = 0$, so this check indeed ensures that no cheating occurred.

A detail we have neglected is that, simply reconstructing all rows of $H \cdot D$ towards all members of $\mathcal{C}_{i+3}$ is too expensive since it requires n^2 communication. Instead, the parties in $\mathcal{C}_{i+2}$ apply an error correcting code simultaneously to each column of the matrix, instantiated with multiplication by a super-invertible matrix, that expands these $n-2t-1$ rows to n rows. Then, the j-th new row is reconstructed only towards $P_j \in \mathcal{C}_{i+3}$. This ensures that if there is at least one row in $H \cdot D$ that is not zero, then at least one *honest* party in $\mathcal{C}_{i+3}$ receives a non-zero row. This party signals this to the parties in $\mathcal{C}_{i+4}$, who learn at that point whether there was an error in the original sharings or not.

[2]The "worst-case degree" for an inconsistent sharing is $2t$, since the shares of the $n - t = 2t + 1$ honest parties uniquely define a polynomial of degree $\leq 2t$. Hence, a corrupt party cannot cheat on the degree, and at worst can cheat by changing the secret.

The downside of this approach is that some honest parties in $\mathcal{C}_{i+3}$ may not know yet that an error took place. This is an important and subtle issue, since some honest parties in $\mathcal{C}_{i+3}$ may not know yet that some $[\![\boldsymbol{a}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+2}}_{2t}$ is inconsistent, and yet they would proceed with the protocol specification. This could be a problem if the steps that follow leak sensitive information when they are executed on inconsistent shares, as noted for example in [GLS19]. However, this is not a problem in our case. At a high level, compared to [GLS19], we use degree-$2t$, while they use degree-t (which are important for G.O.D.). This removes all "extra redundancy" that the honest parties' shares may carry, which is what causes privacy issues in [GLS19].

Ensuring consistency of $[\boldsymbol{a}^j]^{\mathcal{C}_{i+3}}_{2t}$. Recall that each $\boldsymbol{a}^j \in \mathbb{F}$ is a share of the degree-$2t$ packed sharing $[\![\boldsymbol{a}]\!]^{\mathcal{C}_{i+2}}_{2t}$. As it turns out, the consistency of $[\boldsymbol{a}^j]^{\mathcal{C}_{i+3}}_{2t}$ is verified in a similar manner as above. This is thanks to the observation that $(\boldsymbol{a}^1, \dots, \boldsymbol{a}^n)$ is a consistent degree-$2t$ sharing (it equals $[\![\boldsymbol{a}]\!]^{\mathcal{C}_{i+2}}_{2t}$), so it should equal $\mathbf{0}$ when multiplied by the matrix H. Furthermore, since the parties in $\mathcal{C}_{i+3}$ have sharings of each $[\boldsymbol{a}^j]^{\mathcal{C}_{i+3}}_{2t}$, they can compute shares of this matrix product. As before, instead of sending these shares to all parties in $\mathcal{C}_{i+4}$, the parties in $\mathcal{C}_{i+3}$ first apply a super-invertible matrix and reconstruct the j-th entry to the j-th party in $\mathcal{C}_{i+4}$, who checks the underlying secret is 0. If one party finds an error, this is reported to all parties in $\mathcal{C}_{i+5}$.

1.2.7 Final remarks

Communication complexity. Notice that communication of all the steps described above involves each party from one committee sending a constant amount of field elements to the parties in the next committee. This leads to a communication of $O(n^2)$. However, recall that this is to process a batch of $t + 1 = \Omega(n)$ multiplication gates, which means that the amortized cost per gate is $O(n)$, as desired.

Input and addition gates. We also point out that similar techniques as sketched above are used to handle input and addition gates: the parties somehow reshare the values, and some checks are performed to ensure consistency. This is done as the resharing of the triples, essentially.

1.3 Outline of this Document

Section 2 introduces multiple preliminaries. Then we introduce and instantiate three core functionalities we need for our protocol in Section 3: the generation of random sharings, and the input and output protocols. Section 4 contains the most technically challenging contribution of our work, which is our resharing protocols to transfer a set of sharings from one committee to the next with linear communication and perfect security (with abort). Finally, Section 5 puts together the building blocks discussed in previous sections and presents our end-to-end perfectly secure Fluid MPC protocol with linear communication.

2 Security Model and Preliminaries

We present some of the preliminaries required in our work. First we discuss the fluid model in Section 2.1, and then in Section 2.2 we present our security model. We utilize the universal composability framework of [Can01].

2.1 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. 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 parties, an input phase where clients provide inputs, an execution stage where the parties 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 $\mathcal{C}_i$. For simplicity, we will assume throughout that $n = n_i = |\mathcal{C}_i|$, for every 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.

- *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$^+$21; DDG$^+$23; BEP23], we only consider maximal fluidity.
- *Committee formation.* The committees used in each epoch are chosen on-the-fly throughout the execution stage. See [CGG$^+$21] for more motivation and details on committee selection. The model of [CGG$^+$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 the case in which the number of corrupted parties is at most one third of the size of the committee, that is, $3t + 1 = n$. We will sometimes refer to the honest parties of committee $\mathcal{C}_i$ as $\mathcal{H}_{\mathcal{C}_i}$, and its corrupt parties as $\mathcal{T}_{\mathcal{C}_i}$.

We consider a *malicious R-adaptive adversary* from [CGG$^+$21] and used in [RS22; BEP23]. In short, 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.

2.2 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$^+$21; BEP23]), where the adversary can prevent any clients of his or her choice from receiving output. However, similar to the protocols of [CGG$^+$21] (c.f. Appendix A) and [BEP23], 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 the adversary.

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 the adversary.

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 the adversary.

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 the adversary.

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 the adversary.

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 the adversary. Wait for input from the adversary, and if it is $\mathsf{Deliver}$, send the output to every $P_i \in \mathcal{C}_{\mathsf{clnt}}$. Otherwise, abort.

For simplicity, we omit committee selection and tracking from the description of the ideal functionalities presented in the rest of the paper. However, these components are (implicitly) exactly as presented in $\mathcal{F}_{\mathsf{DABB}}$.

2.3 Preliminaries

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

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.

Secret Sharing. We first let $[x]_d^{\mathcal{C}_i}$ denote a Shamir secret sharing of value x with degree d among the parties of $\mathcal{C}_i$. We let x^j be the j-th share of a Shamir secret sharing $[x]_d^{\mathcal{C}_i}$ (typically held by party P_j of committee $\mathcal{C}_i$). As a special case, our protocol will sometimes have a party create a Shamir secret sharing of a share they hold. In this case, we denote that Shamir secret sharing of x^j with degree d among the parties of $\mathcal{C}_i$ as $[x^j]_d^{\mathcal{C}_i}$.

We also use the *packed secret sharing* technique introduced by Franklin and Yung [FY92]. This is similar to Shamir secret sharing, except a vector of ℓ different values $\boldsymbol{x} = (x_1, \dots, x_\ell)$ are shared at once using a polynomial that evaluates to $x_1, \dots, x_\ell$ at ℓ distinct points (w.l.o.g., $-1, \dots, -\ell$)[3]. For privacy, if t players are corrupted, the polynomial must be random of degree at most $d = t + \ell - 1$. Throughout this paper, we will use $\ell = t + 1$, and thus $d = 2t$. In this case, similar to standard Shamir secret sharing of degree $d = 2t$, from a set of n shares where at most $t < n/3$ shares are corrupt, their is an algorithm that efficiently either determines the correct vector of secrets, or outputs $\perp$ (denoting an error). We denote a packed secret sharing value of vector $\boldsymbol{x} = (x_1, \dots, x_\ell)$ with degree d among the parties of $\mathcal{C}_i$ as $[\![\boldsymbol{x}]\!]_d^{\mathcal{C}_i}$. We let $\boldsymbol{x}^j$ be the j-th share of a packed secret sharing $[\![\boldsymbol{x}]\!]_d^{\mathcal{C}_i}$ (typically held by party P_j of committee $\mathcal{C}_i$).

As a special case, our protocol will sometimes have a party create a packed secret sharing of a vector of shares, each from different Shamir secret sharings, that they hold. In this case, we denote the vector of shares corresponding to sharings $([x_1]_d^{\mathcal{C}_i}, \dots, [x_\ell]_d^{\mathcal{C}_i})$ that Party P_j in Committee $\mathcal{C}_i$ holds as $\boldsymbol{x}^j_{[1,\ell]} = (x_1^j, \dots, x_\ell^j)$ and their packed secret sharing of the vector with degree d among the parties of $\mathcal{C}_i$ as $[\![\boldsymbol{x}^j_{[1,\ell]}]\!]_d^{\mathcal{C}_i}$.

Parity Check Matrices. Given a vector $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{F}^n$, we say that $\boldsymbol{x}$ is d-consistent if there exists a polynomial f of degree $\leq d$ such that $f(i) = x_i$ for $i \in [n]$. That is, $\boldsymbol{x}$ constitutes valid sharings of some secret $x = f(0)$. Testing if a vector $\boldsymbol{x}$ is d-consistent can be done by interpolating a polynomial of degree $\leq d$ from the first $d+1$ entries, and checking that the remaining $n - (d+1)$ entries are consistent with this polynomial. This can be expressed as a matrix product $(x_{d+2}, \dots, x_n)^\intercal = H' \cdot (x_1, \dots, x_{d+1})$, which can itself be written as $\mathbf{0} = H \cdot \boldsymbol{x}$, where H is a $(n - d - 1) \times n$ matrix. More formally, H is the *parity check matrix* of the Reed-Solomon code of length n and dimension $d + 1$, and it satisfies that a vector $\boldsymbol{x} \in \mathbb{F}^n$ is d-consistent if and only if $\mathbf{0} = H \cdot \boldsymbol{x} \in \mathbb{F}^{n-(d+1)}$. We will make use of this matrix throughout our work.

Super-Invertible and Hyper-Invertible Matrices. In our work we make use of super-invertible matrices, which satisfy that every square submatrix of maximal size is invertible, and also hyper-invertible matrices [BH08], which satisfy that every square submatrix of *any* size is invertible. Given a $(n \times m)$ matrix M, and given sets $C \subseteq [m]$ and $R \subseteq [n]$, we use M_R^C to denote the $|R| \times |C|$ matrix consistent of the rows of M indexed by the set R, and the columns of M indexed by C.

3 Basic Functionalities

This section discusses three crucial functionalities for our final protocol, whose instantiations are somewhat direct adaptations of previous non-fluid works. The first is a functionality

[3]The underlying field size therefore must be at least $\ell + n$.

for generating sharings of uniformly random values, which recall from the overview in Section 1.2 is one of the ingredients needed to generate multiplication triples, which are used to process multiplication gates. The other two functionalities are concerned with how the clients provide inputs, and how they get outputs once the computation is done.

3.1 Random Sharings Generation [BH08]

We define the functionality for random sharing generation $\mathcal{F}_{\mathsf{rand}}$ below. It first generates random sharings $[r_1]_t^{\mathcal{C}_{i+1}}, \ldots, [r_{t+1}]_t^{\mathcal{C}_{i+1}}$ consistent with t shares received from the adversary corresponding to corrupted parties for each. It then distributes to the honest parties of committee $\mathcal{C}_{i+1}$ their shares of these sharings, summed with their shares of error-sharings $[e_1]_{t'}^{\mathcal{C}_{i+1}}, \ldots, [e_{t+1}]_{t'}^{\mathcal{C}_{i+1}}$, respectively, received from the adversary, where $t' \leq 2t$. Finally, if any such sharing $[e_1]_{t'}^{\mathcal{C}_{i+1}}$ is not equal to the all-0 sharing, then $\mathcal{F}_{\mathsf{rand}}$ sends abort to the honest parties of $\mathcal{C}_{i+3}$.

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

1. $\mathcal{F}_{\mathsf{rand}}$ receives from the adversary the set of shares $\left(\{r_1^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}, \ldots, \{r_{t+1}^k,\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}\right)$ and sharings $([e_1]_{t'}^{\mathcal{C}_{i+1}}, \ldots, [e_{t+1}]_{t'}^{\mathcal{C}_{i+1}})$, where $t' \leq 2t$.
2. $\mathcal{F}_{\mathsf{rand}}$ then samples $r_1, r_2, \ldots, r_{n-2t}$ randomly, and generates $[r_1]_t^{\mathcal{C}_{i+1}}, \ldots, [r_{t+1}]_t^{\mathcal{C}_{i+1}}$ such that for every $j \in [t+1]$ and $k \in \mathcal{T}_{\mathcal{C}_{i+1}}$, the k-th shares of $[r_j]_t^{\mathcal{C}_{i+1}}$ are r_j^k.
3. Next, for every $j \in [t+1], l \in \mathcal{H}_{\mathcal{C}_{i+1}}$, $\mathcal{F}_{\mathsf{rand}}$ sends the l-th shares of $[r_j]_t^{\mathcal{C}_{i+1}} + [e_j]_{t'}^{\mathcal{C}_{i+1}}$ to P_l.
4. Finally, if any $[e_j]_{t'}^{\mathcal{C}_{i+1}}$ is not equal to $[0]_0^{\mathcal{C}_{i+1}}$, i.e., the all-0 sharing, then $\mathcal{F}_{\mathsf{rand}}$ sends abort to the honest parties of $\mathcal{C}_{i+3}$. Otherwise, $\mathcal{F}_{\mathsf{rand}}$ asks the adversary whether to continue, and if the adversary replies (abort, A) for $A \subseteq \mathcal{H}_{\mathcal{C}_{i+3}}$, then $\mathcal{F}_{\mathsf{rand}}$ sends abort to the honest parties A of $\mathcal{C}_{i+3}$.

Functionality $\mathcal{F}_{\mathsf{rand}}$ is instantiated by Protocol Π_{rand} in Section C.1 in the Supplementary Material. Our protocol is a simple adaptation of the random sharing generation protocol by [BH08], except messages are sent from one committee to the next. The communication complexity is amortized linear per random sharing. We state the following Lemma whose proof appears in Section C.1 in the Supplementary Material.

Lemma 1. *Protocol Π_{rand} UC-realizes $\mathcal{F}_{\mathsf{rand}}$.*

3.2 Input and Output

Functionality $\mathcal{F}_{\mathsf{input}}$ below models how clients provide inputs to the computation, and it is instantiated by Protocol Π_{input}, which is appears in Section C.2 in the Supplementary Material. The instantiation is quite standard: clients learn a random value that they can use to mask their input, sending this masked value to the parties in the committee that initiates the computation, who can unmask this element using secret-sharing. Here, we assume the client committee $\mathcal{C}_{\mathsf{clnt}}$ is the first *two* committees, or in other words, clients have fluidity 2. This can be easily removed by placing one committee before $\mathcal{C}_{\mathsf{clnt}}$ which send the random sharings to $\mathcal{C}_{\mathsf{clnt}}$, but this would require us to use our resharing protocol, which we present later in Section 4.

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

1. Let x be the input associated with the input gate belonging to the client.
2. If the client is corrupted:
   (a) $\mathcal{F}_{\mathsf{input}}$ first receives the sharing $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ or (abort, A) for $A \subseteq \mathcal{H}_{\mathcal{C}_{\mathsf{clnts}}}$ from the adversary.
   (b) In the former case $\mathcal{F}_{\mathsf{input}}$ forwards to the honest parties their shares.
   (c) In the latter case, $\mathcal{F}_{\mathsf{input}}$ outputs abort to the honest clients A of $\mathcal{C}_{\mathsf{clnt}}$.
3. If the client is honest:
   (a) $\mathcal{F}_{\mathsf{input}}$ first receives x from the client, then it receives from the adversary a set of shares $\{x^j\}_{j \in \mathcal{T}_{\mathcal{C}_{\mathsf{clnts}}}}$ or abort.
   (b) In the former case, $\mathcal{F}_{\mathsf{input}}$ then samples a sharing $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ based on the t shares x^j from the adversary, the value x, and t other randomly sampled shares, then sends to the honest parties their shares.
   (c) In the latter case, $\mathcal{F}_{\mathsf{input}}$ outputs abort to the honest clients.

We prove the following in Section C.2 in the Supplementary Material.

Lemma 2. *Π_{input} UC-realizes $\mathcal{F}_{\mathsf{input}}$ in the $\mathcal{F}_{\mathsf{rand}}$-hybrid model.*

Finally, $\mathcal{F}_{\mathsf{output}}$ models how the clients get output, and its instantiation, Π_{output}, is given in Section C.3 in the Supplementary Material. The protocol is quite simple: the parties from the committee holding shares of the output send their shares to the receiving client, who performs error detection to reconstruct the correct output (or abort).

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

1. Let $[z]_{2t}^{\mathcal{C}_\ell}$ be the sharing associated with the output gate which belongs to the client, held by the final committee $\mathcal{C}_\ell$.
2. $\mathcal{F}_{\mathsf{output}}$ first receives the shares of $[z]_{2t}^{\mathcal{C}_\ell}$ from the honest parties $\mathcal{H}_{\mathcal{C}_\ell}$ and uses them to reconstruct all of $[z]_{2t}^{\mathcal{C}_\ell}$.
3. Depending on whether the client is honest or not, there are two cases:
   (a) If the client is corrupted, $\mathcal{F}_{\mathsf{output}}$ sends the whole sharing $[z]_{2t}^{\mathcal{C}_\ell}$ to the adversary. If the adversary replies (abort, A) for $A \subseteq \mathcal{H}_{\mathcal{C}_{\mathsf{clnts}}}$, $\mathcal{F}_{\mathsf{output}}$ sends abort to all honest clients A of $\mathcal{C}_{\mathsf{clnt}}$.
   (b) If the client is honest, $\mathcal{F}_{\mathsf{output}}$ sends just the corrupt parties' shares of $[z]_{2t}^{\mathcal{C}_\ell}$ to the adversary. Then, $\mathcal{F}_{\mathsf{output}}$ asks the adversary whether it should continue. If the adversary replies abort, $\mathcal{F}_{\mathsf{output}}$ sends abort to all honest parties. Otherwise, $\mathcal{F}_{\mathsf{output}}$ sends z to the client.

The proof of the following is given in Section C.3 in the Supplementary Material.

Lemma 3. *Π_{output} UC-realizes $\mathcal{F}_{\mathsf{output}}$.*

4 Robust Linear-Overhead Resharing

A central building block needed for Fluid MPC protocols consists of transferring secret-shared values from one committee to the next. This is needed at least to transfer the "state" of the computation itself, but in our case, as in [BEP23], it is also used to transfer other information such as multiplication triples, in order to get efficient multiplication. In this section we present efficient resharing protocols for the case of perfect security with abort. In more detail, consider a committee $\mathcal{C}_i$ that holds sharings $[x]_{2t}^{\mathcal{C}_i}$. Ideally, we would design a protocol such that committee $\mathcal{C}_{i+1}$ receives sharings $[x]_{2t}^{\mathcal{C}_{i+1}}$. Unfortunately, one can easily hint why such primitive is hard to instantiate with *linear* communication: there have to be at least t honest parties in $\mathcal{C}_{i+1}$ who derive their shares from more than t messages from $\mathcal{C}_i$, since otherwise, the adversary corrupting t parties in $\mathcal{C}_i$ could learn these t receivers' messages and hence their shares. Instead, we design a protocol with *linear* communication that lets committee $\mathcal{C}_{i+2}$ obtain shares $[x]_{2t}^{\mathcal{C}_i}$, instead of $\mathcal{C}_{i+1}$. This is sufficient for our main protocol.

Our protocol operates in *batches*, resharing a group of $t+1 = \Omega(n)$ sharings with communication $O(n^2)$, which is linear amortized. The description of our protocol is divided into two steps. First, the parties in $\mathcal{C}_i$ reshare the $t+1$ secrets into an intermediate *packed* version of them towards committee $\mathcal{C}_{i+1}$. This is described in Section 4.1 below. Next, parties in $\mathcal{C}_{i+1}$ somehow "unpack" these sharings so that the parties in $\mathcal{C}_{i+2}$ obtain standard Shamir sharings of the original batch. This second part appears in Section 4.2.

4.1 Robust Resharing from Standard to Packed

Consider $t+1$ sharings $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ held by a committee $\mathcal{C}_i$. We first present a protocol in which the parties in $\mathcal{C}_{i+1}$ can obtain packed secret-sharings $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_{i+1}}$, where $\boldsymbol{x} = (x_1, \ldots, x_{t+1})$. The formal functionality is described below as $\mathcal{F}_{\textsf{robust-packed-reshare}}$. Notice that there are some technicalities involved in the definition of the functionality. First, corrupt parties in $\mathcal{C}_i$ can cheat in this protocol and cause the parties in $\mathcal{C}_{i+1}$ to obtain incorrect sharings $[\![\boldsymbol{x}+\boldsymbol{e}]\!]_{2t}^{\mathcal{C}_{i+1}}$, where $\boldsymbol{e}$ is some error vector chosen by the adversary. Our protocol guarantees that such error will be caught by the parties in $\mathcal{C}_{i+3}$, so the functionality models this fact by sending abort to the parties in this committee. In addition, the adversary may cause some of the parties in $\mathcal{C}_{i+3}$ to abort (selectively), and the functionality also accounts for this.

Functionality 5: $\mathcal{F}_{\textsf{robust-packed-reshare}}$

1. Let $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ be sharings held by the parties of $\mathcal{C}_i$, corresponding to the vector of values $\boldsymbol{x} = (x_1, \ldots, x_{t+1})$.
2. $\mathcal{F}_{\textsf{robust-packed-reshare}}$ first receives from the honest parties of $\mathcal{C}_i$ their shares of $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ (at least $2t+1$ for each of them).
3. $\mathcal{F}_{\textsf{robust-packed-reshare}}$ then reconstructs all of $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ and sends the shares of corrupted parties to the adversary.
4. $\mathcal{F}_{\textsf{robust-packed-reshare}}$ then receives from the adversary a set of shares $\{\boldsymbol{x}^k\}_{k \in \mathcal{T}_{\mathcal{C}_{i+1}}}$, and error vector $\boldsymbol{e}$.
5. Next, $\mathcal{F}_{\textsf{robust-packed-reshare}}$ computes the sharing $[\![\boldsymbol{x}+\boldsymbol{e}]\!]_{2t}^{\mathcal{C}_{i+1}}$ based on the t shares $\boldsymbol{x}^k$ from the adversary and the vector $\boldsymbol{x}+\boldsymbol{e}$, then sends to the honest parties their shares.
6. Finally, if $\boldsymbol{e} \neq (0, \ldots, 0)$, $\mathcal{F}_{\textsf{robust-packed-reshare}}$ sends abort to the honest parties of $\mathcal{C}_{i+3}$. Otherwise, $\mathcal{F}_{\textsf{robust-packed-reshare}}$ asks the adversary whether to continue, and if the adversary replies $(\textsf{abort}, A)$ for $A \subseteq \mathcal{H}_{\mathcal{C}_{i+3}}$, then $\mathcal{F}_{\textsf{robust-packed-reshare}}$ sends abort to the honest parties A of $\mathcal{C}_{i+3}$.

Our protocol is conceptually simple: the parties in $\mathcal{C}_i$ each collect their shares of the batch of secrets, and distribute random packed sharings of these. The receiving parties in $\mathcal{C}_{i+1}$ can then take an appropriate linear combination of these sharings (using Lagrange coefficients) to derive packed sharings of the underlying vector of secrets. To prevent a corrupt party from sending shares of an incorrect vector, we use the parity-check matrix defined in Section 2.3: we let H denote the $(t+1)\times n$ matrix such that $H\cdot(x^1,\dots,x^n)^\intercal=(0,\dots,0)^\intercal$ if and only if $(x^1,\dots,x^n)$ are $2t$-consistent. Via H, checking if a group of values is $2t$-consistent reduces to applying a linear combination on these and checking that the result is zero. The parties in $\mathcal{C}_{i+1}$ can apply such combination to their received packed sharings, which can be checked by the parties in a future committee. Instead of reconstructing this linear combination to the parties in $\mathcal{C}_{i+2}$, which would be too expensive, the parties perform the linear reconstruction from [DN07] that reconstructs these sharings to the parties in committee $\mathcal{C}_{i+3}$, who check that the results are zero.

Our protocol is presented formally as Protocol $\Pi_{\textsf{robust-packed-reshare}}$ below. We note here that $\Pi_{\textsf{robust-packed-reshare}}$ can successfully be used on input sharings $([x_1]_t^{\mathcal{C}_i},\dots,[x_{t+1}]_t^{\mathcal{C}_i})$ of degree-t instead of degree-$2t$. Indeed, the only property that is required of the original sharings $[x_\alpha]_t^{\mathcal{C}_i}$ for $\alpha\in[t+1]$ is that their shares lie on a polynomial of degree *at most 2t*, which is of course true. We will use this fact in our main Fluid MPC protocol.

Protocol 1: $\Pi_{\textsf{robust-packed-reshare}}$

Usage: Committee $\mathcal{C}_i$ holds standard sharings $([x_1]_{2t}^{\mathcal{C}_i},\dots,[x_{t+1}]_{2t}^{\mathcal{C}_i})$ corresponding to the vector of values $\boldsymbol{x}=(x_1,\dots,x_{t+1})$ and committee $\mathcal{C}_{i+1}$ outputs a single packed sharing $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_{i+1}}$, using committees $\mathcal{C}_{i+2}$ and $\mathcal{C}_{i+3}$ to ensure its correctness.

1. Every party P_j in committee $\mathcal{C}_i$ distributes degree-$2t$ packed sharings $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+1}}$ to committee $\mathcal{C}_{i+1}$ of its shares $\boldsymbol{x}^j_{[1,t+1]}=(x^j_1,\dots,x^j_{t+1})$ of the sharings $([x_1]_{2t}^{\mathcal{C}_i},\dots,[x_{t+1}]_{2t}^{\mathcal{C}_i})$ of the values in $\boldsymbol{x}$.
2. While the Verification phase (below) executes, the parties of committee $\mathcal{C}_{i+1}$ compute and output fresh packed shares $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_{i+1}}=\sum_{j=1}^{2t+1}L_j(0)\cdot[\![\boldsymbol{x}^j_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+1}}$, where $L_j(0)$ are Lagrange interpolation coefficients.

Verification Phase:

1. The parties P_k of committee $\mathcal{C}_{i+1}$ apply the parity check matrix H to get $([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}},\dots,[\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal\leftarrow H\cdot([\![\boldsymbol{x}^1_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+1}},\dots,[\![\boldsymbol{x}^n_{[1,t+1]}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal$.
2. Next, they apply super-invertible matrix M to get $([\![\boldsymbol{z}_1]\!]_{2t}^{\mathcal{C}_{i+1}},\dots,[\![\boldsymbol{z}_n]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal\leftarrow M\cdot([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}},\dots,[\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal$, and open $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$ to party P_l of committee $\mathcal{C}_{i+2}$.
3. Finally, each party P_l of committee $\mathcal{C}_{i+2}$ checks that the shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$ are $2t$-consistent and that they correspond to $\boldsymbol{z}_l=0^\ell$.
4. If either of the checks fail, they send $\textsf{abort}$ to the parties of committee $\mathcal{C}_{i+3}$.

The communication complexity of $\Pi_{\textsf{robust-packed-reshare}}$ is $n\cdot n=O(n^2)$ sharings in step 1, plus $n\cdot n=O(n^2)$ sharings from second step in the verification, for a total of $O(n^2/(t+1))=O(n)$ per value being reshared. The proof of the following is given in Section D in the Supplementary Material.

Lemma 4. $\Pi_{\textsf{robust-packed-reshare}}$ *UC-realizes* $\mathcal{F}_{\textsf{robust-packed-reshare}}$.

4.2 Robust Resharing from Packed to Standard

Now we turn our attention to the second part of our 2-hop resharing protocol in which the parties in $\mathcal{C}_{i+1}$ can "unpack" the sharings they have received from $\mathcal{C}_i$. In order to make this section independent of the previous one, however, we relabel the committees and assume that the committee that starts with the packed sharings is $\mathcal{C}_i$, instead of $\mathcal{C}_{i+1}$. In this case, the context is the following: $\mathcal{C}_i$ holds packed sharings $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$, and the goal is for committee $\mathcal{C}_{i+1}$ to obtain *unpacked* Shamir sharings $([x_1]_{2t}^{\mathcal{C}_{i+1}},\dots,[x_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$. We model this with

Functionality $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ below. As before, the main complication in modeling this approach is that the adversary can introduce errors in the sharings distributed, which will be caught by a future committee $\mathcal{C}_{i+3}$.

Functionality 6: $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$

1. Let $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$ be the packing held by the parties of $\mathcal{C}_i$, corresponding to the vector of values $\boldsymbol{x} = (x_1, \dots, x_{t+1})$.
2. $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ first receives from the honest parties of $\mathcal{C}_i$ their shares of $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$ (at least $2t+1$ of them).
3. $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ then reconstructs all of $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$ and sends the shares of corrupted parties to the adversary.
4. $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ then receives from the adversary a set of shares $\{(x_1^k, \dots, x_{t+1}^k)\}_{k \in \mathcal{T}_{\mathcal{C}_{i+1}}}$, and errors $(e_1, \dots, e_{t+1})$.
5. Next, $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ computes the sharings $([x_1 + e_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [x_{t+1} + e_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$ based on the t shares x_α^k from the adversary, the value $x_\alpha + e_\alpha$, and t other randomly sampled shares, for each $\alpha \in [t+1]$, then sends to the honest parties their shares.
6. Finally, if any $e_\alpha \neq 0$, $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ sends abort to the honest parties of $\mathcal{C}_{i+3}$. Otherwise, $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ asks the adversary whether to continue, and if the adversary replies (abort, A) for $A \subseteq \mathcal{H}_{\mathcal{C}_{i+3}}$, then $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ sends abort to the honest parties A of $\mathcal{C}_{i+3}$.

Our protocol to instantiate $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$, Protocol $\Pi_{\mathsf{robust\text{-}standard\text{-}reshare}}$, proceeds as follows. First, each party in $\mathcal{C}_i$, the committee holding the packed sharings, secret-shares using standard Shamir secret-sharing their own share. At this point, the parties in $\mathcal{C}_{i+1}$ can take multiple linear combinations using Lagrange coefficients on these Shamir shares to compute shares of each one of the secrets in the original vector. As with $\Pi_{\mathsf{robust\text{-}packed\text{-}reshare}}$, corrupt parties may attempt to change their shares when resharing. This once again is caught by employing the matrix H from Section 4.1, and the committee $\mathcal{C}_{i+3}$ learns whether the distributed values are consistent or not. The protocol is described below.

Protocol 2: $\Pi_{\mathsf{robust\text{-}standard\text{-}reshare}}$

Usage: Committee $\mathcal{C}_i$ holds a packed sharing $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$ and committee $\mathcal{C}_{i+1}$ outputs standard sharings $([x_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [x_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$ corresponding to the vector of values $\boldsymbol{x} = (x_1, \dots, x_{t+1})$, using committees $\mathcal{C}_{i+2}$ and $\mathcal{C}_{i+3}$ to ensure its correctness.

1. Every party P_j in committee $\mathcal{C}_i$ distributes degree-$2t$ standard sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$ to committee $\mathcal{C}_{i+1}$ of its share $\boldsymbol{x}^j$ of the packed sharing $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$.
2. While the Verification phase (below) executes, the parties of committee $\mathcal{C}_{i+1}$ compute and output fresh shares for every $\alpha \in [t+1]$: $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}} = \sum_{j=1}^{2t+1} L_j(-\alpha) \cdot [\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$, where $L_j(-\alpha)$ are Lagrange interpolation coefficients.

Verification Phase:

1. The parties P_k of committee $\mathcal{C}_{i+1}$ apply the parity check matrix H to get $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow H \cdot ([\boldsymbol{x}^1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\boldsymbol{x}^n]_{2t}^{\mathcal{C}_{i+1}})^\intercal$.
2. Next, they apply super-invertible matrix M to get $([z_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [z_n]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal$, and open $[z_l]_{2t}^{\mathcal{C}_{i+1}}$ to party P_l of committee $\mathcal{C}_{i+2}$.
3. Finally, each party P_l of committee $\mathcal{C}_{i+2}$ checks that the shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$ are $2t$-consistent and that they correspond to $z_l = 0$.
4. If either of the checks fail, they send abort to the parties of committee $\mathcal{C}_{i+3}$.

As with $\Pi_{\mathsf{robust\text{-}packed\text{-}reshare}}$, the communication complexity of $\Pi_{\mathsf{robust\text{-}standard\text{-}reshare}}$ is $n \cdot n = O(n^2)$ sharings in step 1, plus $n \cdot n = O(n^2)$ sharings from second step in the verification, for a total of $O(n^2/(t+1)) = O(n)$ per value being reshared. Security is given in the following lemma, which is proved in Section D in the Supplementary Material.

Lemma 5. $\Pi_{\mathsf{robust\text{-}standard\text{-}reshare}}$ *UC-realizes* $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$.

5 Linear-Overhead Perfect Fluid MPC with Abort

With our core construction for linear communication resharing at hand, given by protocols $\Pi_{\mathsf{robust\text{-}packed\text{-}reshare}}$ and $\Pi_{\mathsf{robust\text{-}standard\text{-}reshare}}$ from Section 4, we are ready to present in detail our end-to-end Fluid MPC protocol with linear communication, for perfect security with abort. First, in Section 5.1 we describe how multiplications are handled. Then, in Section 5.2 we present our actual Fluid MPC protocol.

5.1 Linear-Overhead Multiplication Procedure

In our MPC protocol Π_{main} from Section 5.2, the parties in a committee $\mathcal{C}_{i+1}$ will hold two groups of sharings $([x_1]_{2t}^{\mathcal{C}_{i+1}}, , \dots, [x_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$ and $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$, and the goal will be for the parties in $\mathcal{C}_{i+3}$ to obtain sharings $([x_1 y_1]_{2t}^{\mathcal{C}_{i+3}}, \dots, [x_{t+1} y_{t+1}]_{2t}^{\mathcal{C}_{i+3}})$. This specific step in the protocol is addressed by Procedure π_{mult}, which we describe below, and is caled from within Protocol Π_{main}.

At a high-level, committees $\mathcal{C}_{i+1}$ through committees $\mathcal{C}_{i+3}$ will execute beaver multiplication [Bea92], using multiple kings in $\mathcal{C}_{i+2}$, as specified by the standard technique from [DN07]. To do this, committee $\mathcal{C}_i$ first invokes $\mathcal{F}_{\mathsf{rand}}$ to output random sharings $[a_\alpha]_t^{\mathcal{C}_{i+1}}, [b_\alpha]_t^{\mathcal{C}_{i+1}}$, for $\alpha \in [t+1]$ to committee $\mathcal{C}_{i+1}$. Committee $\mathcal{C}_{i+1}$ then locally computes for each $\alpha \in [t+1]$, $[c_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [a_\alpha]_t^{\mathcal{C}_{i+1}} \cdot [b_\alpha]_t^{\mathcal{C}_{i+1}}$. Then, committee $\mathcal{C}_{i+1}$ invokes $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ on these sharings so that committee $\mathcal{C}_{i+2}$ receives packing $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$ for vector $\boldsymbol{a} = (a_1, \dots, a_{t+1})$, and the same for $[\![\boldsymbol{b}]\!]_{2t}^{\mathcal{C}_{i+2}}, [\![\boldsymbol{c}]\!]_{2t}^{\mathcal{C}_{i+2}}$. Finally, committee $\mathcal{C}_{i+2}$ invokes $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ on these packings, so that committee $\mathcal{C}_{i+3}$ receives $[a_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [b_\alpha]_{2t}^{\mathcal{C}_{i+3}}, [c_\alpha]_{2t}^{\mathcal{C}_{i+3}}$ for $\alpha \in [t+1]$, where $(a_\alpha, b_\alpha, c_\alpha)$ are the same triples that committee $\mathcal{C}_{i+1}$ had.

Note that the beaver triple used by the parties in committee $\mathcal{C}_{i+1}$ is $([a_\alpha]_t^{\mathcal{C}_{i+1}}, [b_\alpha]_t^{\mathcal{C}_{i+1}}, [a_\alpha]_t^{\mathcal{C}_{i+1}} \cdot [b_\alpha]_t^{\mathcal{C}_{i+1}})$, which is *not* a truly random triple of degree-$2t$ since (1) the first two entries are random of degree-t, not $2t$, and (2) the the underlying polynomial in the last entry is not random as it is the product of two degree-t polynomials. However, this is acceptable in our context since these sharings will be *reshared* with $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ and $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$, which are agnostic to the distribution of the input sharings, and guarantee the output sharings are freshly random, as required.

Procedure 3: π_{mult}

Usage: Multiply $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ and $[y_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ held by committee $\mathcal{C}_{i+1}$ so that committee $\mathcal{C}_{i+3}$ outputs $[x_\alpha y_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, for $\alpha \in [t+1]$.

1. Committee $\mathcal{C}_i$ first invokes $\mathcal{F}_{\mathsf{rand}}$ to output random sharings $[a_\alpha]_t^{\mathcal{C}_{i+1}}, [b_\alpha]_t^{\mathcal{C}_{i+1}}$, for $\alpha \in [t+1]$ to committee $\mathcal{C}_{i+1}$ (using committees $\mathcal{C}_{i+2}$ and $\mathcal{C}_{i+3}$ for verification, which aborts if needed).
2. The parties P_j in $\mathcal{C}_{i+1}$ next multiply for each $\alpha \in [t+1]$, $[c_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [a_\alpha]_t^{\mathcal{C}_{i+1}} \cdot [b_\alpha]_t^{\mathcal{C}_{i+1}}$.
3. Committee $\mathcal{C}_{i+1}$ then computes $[d_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [x_\alpha]_{2t}^{\mathcal{C}_{i+1}} + [a_\alpha]_t^{\mathcal{C}_{i+1}}$ and $[e_\alpha]_{2t}^{\mathcal{C}_{i+1}} \leftarrow [y_\alpha]_{2t}^{\mathcal{C}_{i+1}} + [b_\alpha]_t^{\mathcal{C}_{i+1}}$ for every $\alpha \in [t+1]$.

4. Committee $\mathcal{C}_{i+1}$ next applies super-invertible matrix M to $([d_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [d_{t+1}]_{2t}^{\mathcal{C}_{i+1}})$ to get $([d'_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [d'_n]_{2t}^{\mathcal{C}_{i+1}})$, and $([e_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [e_{t+1}]_{2t}^{\mathcal{C}_{i+1}}$ to get $([e'_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [e'_n]_{2t}^{\mathcal{C}_{i+1}})$ and opens $[d'_k]_{2t}^{\mathcal{C}_{i+1}}, [e'_k]_{2t}^{\mathcal{C}_{i+1}}$ to party P_k of committee $\mathcal{C}_{i+2}$.
5. Simultaneously, the parties P_j in $\mathcal{C}_{i+1}$ then invoke $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ towards committee $\mathcal{C}_{i+2}$ on input standard sharings $([a_1]_t^{\mathcal{C}_{i+2}}, \ldots, [a_{t+1}]_t^{\mathcal{C}_{i+2}})$, so that $\mathcal{C}_{i+2}$ gets the packing $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$ for vector $\boldsymbol{a} = (a_1, \ldots, a_{t+1})$ (using committees $\mathcal{C}_{i+3}$ and $\mathcal{C}_{i+4}$ for verification, which aborts if needed).
6. They do the same for $([b_1]_t^{\mathcal{C}_{i+2}}, \ldots, [b_{t+1}]_t^{\mathcal{C}_{i+2}})$ and $([c_1]_{2t}^{\mathcal{C}_{i+2}}, \ldots, [c_{t+1}]_{2t}^{\mathcal{C}_{i+2}})$ to get packings $[\![\boldsymbol{b}]\!]_{2t}^{\mathcal{C}_{i+2}}$ and $[\![\boldsymbol{c}]\!]_{2t}^{\mathcal{C}_{i+2}}$ for vectors $\boldsymbol{b} = (b_1, \ldots, b_{t+1})$ and $\boldsymbol{c} = (c_1, \ldots, c_{t+1})$, respectively.
7. The parties P_k of $\mathcal{C}_{i+2}$ then reconstruct d'_k, e'_k (or abort if unsuccessful) and send them to all parties of committee $\mathcal{C}_{i+3}$.
8. Simultaneously, they run $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ towards committee $\mathcal{C}_{i+3}$ on input $[\![\boldsymbol{a}]\!]_{2t}^{\mathcal{C}_{i+2}}$ so that $\mathcal{C}_{i+3}$ gets the standard sharings $([a_1]_{2t}^{\mathcal{C}_{i+3}}, \ldots, [a_{t+1}]_{2t}^{\mathcal{C}_{i+3}})$ (using committees $\mathcal{C}_{i+4}$ and $\mathcal{C}_{i+5}$ for verification, which aborts if needed). They do the same with $[\![\boldsymbol{b}]\!]_{2t}^{\mathcal{C}_{i+2}}$ and $[\![\boldsymbol{c}]\!]_{2t}^{\mathcal{C}_{i+2}}$.
9. The parties of $\mathcal{C}_{i+3}$ run Berlekamp-Welch on $(d'_1, \ldots, d'_n)$ and $(e'_1, \ldots, e'_n)$ to get $\{d_\alpha\}_{\alpha \in [t+1]}$ and $\{e_\alpha\}_{\alpha \in [t+1]}$, respectively.
10. Finally, committee $\mathcal{C}_{i+3}$ outputs $[x_\alpha y_\alpha]_{2t}^{\mathcal{C}_{i+3}} = d_\alpha \cdot e_\alpha - d_\alpha \cdot [b_\alpha]_{2t}^{\mathcal{C}_{i+3}} - e_\alpha \cdot [a_\alpha]_{2t}^{\mathcal{C}_{i+3}} + [c_\alpha]_{2t}^{\mathcal{C}_{i+3}}$, for each $\alpha \in [t+1]$.

Procedure π_{mult} has communication complexity $O(n^2/(t+1)) = O(n)$ per multiplication. The main properties of Procedure π_{mult} that we will use are summarized in Lemmas 6 and 7 below, whose proofs are given in Section D in the Supplementary Material. We will use these Lemmas in the proof of Theorem 2, the security proof of MPC protocol Π_{main}.

Lemma 6. *Let it be the case that for every $\alpha \in [t+1]$, either the adversary completely knows the sharing $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, or the first t honest parties' shares are distributed randomly to the adversary. If given in addition to the adversary's known shares: random a^*_α for $\alpha \in [t+1]$, and also for those $\alpha \in [t+1]$ satisfying the latter case, random $x^1_\alpha, \ldots, x^t_\alpha$, then values distributed identically to the adversary as honest parties' shares of $[d'_k]_{2t}^{\mathcal{C}_{i+1}}$ received by P_k of $\mathcal{C}_{i+2}$, for $k \in \mathcal{T}_{\mathcal{C}_{i+2}}$, in π_{mult} can be determined. Moreover, values distributed identically to the adversary to d'_k sent from honest party P_k of $\mathcal{C}_{i+2}$, for $k \in \mathcal{H}_{\mathcal{C}_{i+2}}$, in π_{mult} can be determined. The same holds for the corresponding values and sharings of e'_k for $k \in [n]$.*

Lemma 7. *If no parties of committee $\mathcal{C}_{i+3}$ receive* abort *in π_{mult}, then they always correctly determine the values $(x_1 + a_1, \ldots, x_{t+1} + a_{t+1})$ and $(y_1 + b_1, \ldots, y_{t+1} + b_{t+1})$.*

5.2 Main Protocol

At this point we are finally ready to present our main Fluid MPC protocol with linear communication overhead and perfect security with abort, Π_{main}. For each input $x \in \mathbb{F}$ to the computation, the clients simply invoke $\mathcal{F}_{\mathsf{input}}$ on x. Then, for the execution phase, for every batch of multiplication gates at a given layer, the current committee $\mathcal{C}_i$ runs π_{mult}, so that committee $\mathcal{C}_{i+2}$ receives sharings of the products. For every batch of identity gates at the layer, committee $\mathcal{C}_i$ invokes $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ on the input sharings, and then committee $\mathcal{C}_{i+1}$ invokes $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ on the packed secret sharing received from $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$, so that $\mathcal{C}_{i+2}$ receives standard sharings of the inputs to the identity gates. For every batch of addition gates, committee $\mathcal{C}_i$ first adds the sharings for each gate together, then proceeds exactly as described for identity gates above. Finally, for each output z, the parties of the last committee $\mathcal{C}_\ell$ and the clients $\mathcal{C}_{\mathsf{clnt}}$ together invoke $\mathcal{F}_{\mathsf{output}}$ on the sharing $[z]_{2t}^{\mathcal{C}_\ell}$. The protocol is described in detail below.

Protocol 4: Π_{main}

Input Phase:

1. The clients invoke $\mathcal{F}_{\mathsf{input}}$ on each input x and receive back either abort or shares of $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$.

Execution Phase: Every other committee (with the help of the others) will compute the gates at each layer of the circuit as below:

Identity Gates: To forward $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$:

1. The parties of committee $\mathcal{C}_i$ invoke $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ towards committee $\mathcal{C}_{i+1}$ on the above sharings, so that $\mathcal{C}_{i+1}$ gets the packing $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$ for vector $\boldsymbol{x} = (x_1, \ldots, x_{t+1})$ (using committees $\mathcal{C}_{i+2}$ and $\mathcal{C}_{i+3}$ for verification, which aborts if needed).
2. Then, the parties of committee $\mathcal{C}_{i+1}$ invoke $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ towards committee $\mathcal{C}_{i+2}$ on input $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$, so that $\mathcal{C}_{i+2}$ gets the standard sharings $([x_1]_{2t}^{\mathcal{C}_{i+2}}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_{i+2}})$ (using committees $\mathcal{C}_{i+3}$ and $\mathcal{C}_{i+4}$ for verification, which aborts if needed).

Addition: To component-wise add (and forward) $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ and $([y_1]_{2t}^{\mathcal{C}_i}, \ldots, [y_{t+1}]_{2t}^{\mathcal{C}_i})$, the parties of committee $\mathcal{C}_i$ first directly compute

$$([x_1]_{2t}^{\mathcal{C}_i} + [y_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i} + [y_{t+1}]_{2t}^{\mathcal{C}_i}),$$

then run the identity gate procedure on these sharings.

Multiplication: To component-wise multiply $([x_1]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$ and $([y_1]_{2t}^{\mathcal{C}_i}, \ldots, [y_{t+1}]_{2t}^{\mathcal{C}_i})$, the parties of Committee $\mathcal{C}_i$ run π_{mult} on them so that the parties of committee $\mathcal{C}_{i+2}$ receive $([x_1 y_1]_{2t}^{\mathcal{C}_{i+2}}, \ldots, [x_{t+1} y_{t+1}]_{2t}^{\mathcal{C}_{i+2}})$.

Output Phase:

1. For each output gate belonging to a client, the parties of committee $\mathcal{C}_\ell$ invoke $\mathcal{F}_{\mathsf{output}}$ on the corresponding sharing $[z]_{2t}^{\mathcal{C}_\ell}$.
2. If the client receives abort from $\mathcal{F}_{\mathsf{output}}$, then it sends abort to all other clients and aborts itself.
3. The clients then wait until the verification phases of the procedures of the execution phase have ended to output their values z received from $\mathcal{F}_{\mathsf{output}}$.

Per group of $t + 1$ multiplication gates, the communication complexity of Π_{main} is that of π_{mult}, which is $O(n^2)$, for a total of $O(n^2/(t + 1)) = O(n)$ per multiplication gate. If there are $o(n)$ multiplication gates for a given layer, the communication complexity for that layer is $\Omega(n^2)$. Before we prove the security of our protocol, we consider the following simple Lemma that will prove handy.

Lemma 8. *If $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ sends the parties of committee $\mathcal{C}_{i+4}$* abort*, then they abort. Similarly, if $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ sends the parties of committee $\mathcal{C}_{i+5}$* abort*, then they abort.*

Proof. This is immediate from the definitions of $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ and $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$.

□

Theorem 2 below states that Protocol Π_{main} instantiates the ideal Fluid MPC functionality $\mathcal{F}_{\mathsf{DABB}}$. The proof appears in Section D in the Supplementary Material.

Theorem 2. *Protocol Π_{main} UC-securely computes $\mathcal{F}_{\mathsf{DABB}}$ in the presence of an R-adaptive adversary $\mathcal{A}$ in the $(\mathcal{F}_{\mathsf{input}}, \mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}, \mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}, \mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{output}})$-hybrid model.*

Disclaimer

This paper was prepared for informational 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 herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

© 2024 JPMorgan Chase & Co. All rights reserved

References

[AAPP23] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Detect, pack and batch: perfectly-secure mpc with linear communication and constant expected time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 251–281. Springer, 2023.

[AAY21] Ittai Abraham, Gilad Asharov, and Avishay Yanai. Efficient perfectly secure computation with optimal resilience. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II 19*, pages 66–96. Springer, 2021.

[BBG^{+}20] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg. Springer Berlin Heidelberg, 1992. ISBN: 978-3-540-46766-3.

[BEP23] Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid mpc. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 263–294, Cham. Springer Nature Switzerland, 2023.

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In Ran Canetti, editor, *Theory of Cryptography*, pages 213–230, Berlin, Heidelberg. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-78524-8.

[BIK$^+$17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017. DOI: 10.1145/3133956.3133982. URL: https://doi.org/10.1145/3133956.3133982.

[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.

[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.

[DDG$^+$23] Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect mpc over layered graphs. In *Annual International Cryptology Conference*, pages 360–392. Springer, 2023.

[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. Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-13190-5.

[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. DOI: 10.1007/978-3-642-32009-5_38. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_38.

[FHM98] 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.

[FY92] Matthew Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 699–710, Victoria, British Columbia, Canada. Association for Computing Machinery, 1992. ISBN: 0897915119. DOI: 10.1145/129712.129780. URL: https://doi.org/10.1145/129712.129780.

[GHK+21] 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.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional mpc with guaranteed output delivery. In *Annual International Cryptology Conference*, pages 85–114. Springer, 2019.

[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. DOI: 10.1007/978-3-030-26948-7_18. URL: https://doi.org/10.1007/978-3-030-26948-7%5C_18.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority mpc. In *Annual International Cryptology Conference*, pages 618–646. Springer, 2020.

[MWA+23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: multi-round single-server secure aggregation with applications to private federated learning. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 477–496. IEEE, 2023. DOI: 10.1109/SP46215.2023.10179434. URL: https://doi.org/10.1109/SP46215.2023.10179434.

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

[WOG88] Avi Wigderson, MB Or, and S Goldwasser. Completeness theorems for non-cryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, pages 1–10, 1988.

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.

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 unsable 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.

As in [CGG$^+$21], the formation process can be modelled with an ideal functionality that samples and broadcasts committees according to the desired mechanism. In our formalization, we simply require that all parties (both honest and adversarial) of committee

$\mathcal{C}_i$ somehow agree on the next committee $\mathcal{C}_{i+1}$. Formally, it is important to note that committee $\mathcal{C}_{i+1}$ for epoch $i+1$ is only determined and conveyed to everyone at the start of the hand-off phase of epoch i, not before. We make no assumptions or restrictions on the size of committees nor the overlap between committees.

Corruptions. We study the case in which the number of corrupted parties is at most one third of the size of the committee, that is, $3t + 1 = n$, where n is the size of each committee.

More formally, we consider a *malicious R-adaptive adversary* from [CGG⁺21] and used in [RS22]. When the clients are chosen, the adversary *statically* corrupts a set $\mathcal{T}_{\mathcal{C}_{\mathsf{clnts}}} \subseteq \mathcal{C}_{\mathsf{clnt}}$ 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, 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 $j > i$.

B Other Related Work

We already have surveyed in the introduction all relevant related work on the direction of Fluid MPC specifically. However, in the more broad direction of MPC in dynamic settings, several other references exist. In this section a survey of some of these works is provided [BEP23, Section B].

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. [FHM98] 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,n 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+21], the "You Only Speak Once" model for MPC is introduced. In this model, the basic assumption is that the adversary is able to take a party down as soon as that party sends a message - using, say, a denial of service attack. Although some number of parties are assumed to be alive and can receive messages, no particular party is guaranteed to come back (which is the major difference to our model). Instead, 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. 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. On the other hand, 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. 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.

C Supplementary Material for Section 3

C.1 Instantiating $\mathcal{F}_{\mathsf{rand}}$

As we discussed in the overview, this is an adaptation of the random sharing generation protocol by [BH08], except messages are sent from one committee to the next.

Protocol 5: Π_{rand}

Usage: With help from committee $\mathcal{C}_i$, committee $\mathcal{C}_{i+1}$ outputs $t+1$ random sharings $[r]_t^{\mathcal{C}_{i+1}}$, and uses committees $\mathcal{C}_{i+2}$ and $\mathcal{C}_{i+3}$ to ensure their t-consistency.

1. All parties P_j in $\mathcal{C}_i$ sample random s_j and share it to the parties of $\mathcal{C}_{i+1}$ using degree t.
2. The parties of $\mathcal{C}_{i+1}$ first locally apply $(n \times n)$ hyper-invertible matrix M to the sharings from $\mathcal{C}_i$ to obtain

$$([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_n]_t^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([s_1]_t^{\mathcal{C}_{i+1}}, \dots, [s_n]_t^{\mathcal{C}_{i+1}})^\intercal.$$

3. While the Verification Phase (below) is executed, committee $\mathcal{C}_{i+1}$ outputs random sharings $([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_{t+1}]_t^{\mathcal{C}_{i+1}})$.

Verification Phase:

1. Then, the parties of $\mathcal{C}_{i+1}$ open the last $2t$ sharings $[r_{t+2}]_t^{\mathcal{C}_{i+1}}, \dots, [r_n]_t^{\mathcal{C}_{i+1}}$ to the last $2t$ parties of $\mathcal{C}_{i+2}$, respectively.
2. The last $2t$ parties P_l of $\mathcal{C}_{i+2}$ then check that indeed $[r_l]_t^{\mathcal{C}_{i+1}}$ is t-consistent.
3. If this check fails for some P_j, they send abort to the parties of $\mathcal{C}_{i+3}$.

The communication complexity of Π_{rand} is $n \cdot n = O(n^2)$ sharings in step 1, plus $2t \cdot n = O(n^2)$ sharings from the verification, for a total of $O(n^2/(t+1)) = O(n)$ per random sharing. The proof of the following Lemma is given in Section C in the Supplementary Material.

Lemma 9 (Lemma 1, re-stated). *Protocol Π_{rand} UC-realizes $\mathcal{F}_{\mathsf{rand}}$.*

Proof. First we define the simulator $\mathcal{S}$:

1. $\mathcal{S}$ for $j \in \mathcal{H}_{\mathcal{C}_i}$ first samples random sharings $[s_j]_t^{\mathcal{C}_{i+1}}$, then sends the committee $\mathcal{C}_{i+1}$ corrupted parties' shares, $\{s_j^k\}_{k \in \mathcal{T}_{\mathcal{C}_{i+1}}}$ to the adversary.

2. $\mathcal{S}$ then receives from the adversary on behalf of the committee $\mathcal{C}_{i+1}$ honest parties, $\{s_j^k\}_{k \in \mathcal{H}_{\mathcal{C}_{i+1}}}$, for $j \in \mathcal{T}_{\mathcal{C}_i}$.

3. For each $j \in \mathcal{T}_{\mathcal{C}_i}$, $\mathcal{S}$ uses the first $t+1$ honest parties' shares $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, s_j^k, to reconstruct $[s_j]_t^{\mathcal{C}_{i+1}}$.

4. Then, for each $j \in \mathcal{T}_{\mathcal{C}_i}$, letting $s_{j,t}^k$ be the k-th share of $[s_j]_t^{\mathcal{C}_{i+1}}$ just reconstructed, $\mathcal{S}$ uses for the first $t+1$ honest parties in $\mathcal{H}_{\mathcal{C}_{i+1}}$, share 0, and for the remaining t parties $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, $s_j^k - s_{j,t}^k$ to reconstruct $[d_j]_{t'_j}^{\mathcal{C}_{i+1}}$, where $t'_j \leq 2t$ is the smallest value that defines a consistent sharing with the above $2t+1$ shares.

5. $\mathcal{S}$ then computes $([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_n]_t^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([s_1]_t^{\mathcal{C}_{i+1}}, \dots, [s_n]_t^{\mathcal{C}_{i+1}})^\intercal$ and $([e_1]_{t'}^{\mathcal{C}_{i+1}}, \dots, [e_n]_{t'}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot \left(0, \dots, 0, [d_{2t+2}]_{t'_{2t+2}}^{\mathcal{C}_{i+1}}, \dots, [d_n]_{t'_n}^{\mathcal{C}_{i+1}}\right)^\intercal$ on behalf of all of the honest parties of committee $\mathcal{C}_{i+1}$, where $t' = \max\{t'_j\}_{j \in \mathcal{T}_{\mathcal{C}_i}}$.

6. Using the $2t+1$ honest parties' shares of the sharings $([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_{t+1}]_t^{\mathcal{C}_{i+1}})$, $\mathcal{S}$ reconstructs the whole sharings, and sends the corrupted parties' shares to $\mathcal{F}_{\mathsf{rand}}$, along with $([e_1]_{t'}^{\mathcal{C}_{i+1}}, \dots, [e_{t+1}]_{t'}^{\mathcal{C}_{i+1}})$.

7. Then for $j \in [2t+2, n] \cap \mathcal{T}_{\mathcal{C}_{i+2}}$, $\mathcal{S}$ sends to the corresponding corrupted party of committee $\mathcal{C}_{i+2}$, the honest parties' shares of $[r_j]_t^{\mathcal{C}_{i+1}} + [e_j]_{t'}^{\mathcal{C}_{i+1}}$.

8. $\mathcal{S}$ next receives from the adversary, for $j \in [t+2, n] \cap \mathcal{H}_{\mathcal{C}_{i+2}}$, the corrupted parties' shares of $[r_j]_t^{\mathcal{C}_{i+1}} + [e_j]_{t'}^{\mathcal{C}_{i+1}}$, for the corresponding honest party of committee $\mathcal{C}_{i+2}$. Together with the honest parties' shares of these sharings, $\mathcal{S}$ checks that $[r_j]_t^{\mathcal{C}_{i+1}} + [e_j]_{t'}^{\mathcal{C}_{i+1}}$ indeed defines a t-consistent sharing. If not, then $\mathcal{S}$ sends on behalf of P_j, abort to the corrupted parties of committee $\mathcal{C}_{i+3}$.

9. Finally, if $\mathcal{S}$ sent any abort in the above step, it sets $A = \mathcal{H}_{\mathcal{C}_{i+3}}$; otherwise it sets A to be those honest parties that receive from some corrupt party abort. Then, when $\mathcal{F}_{\mathsf{rand}}$ asks $\mathcal{S}$ whether to continue, it replies with (abort, A).

Now we argue that the real world and ideal world are distributed identically to the adversary. In Step 1 of Π_{rand}, the adversary receives the t committee $\mathcal{C}_{i+1}$ corrupted parties' shares of the committee $\mathcal{C}_i$ honest parties' sharings $[s_j]_t^{\mathcal{C}_{i+1}}$. Indeed, Step 1 of $\mathcal{S}$ is to sample random sharings in the same way for each honest party, and send the corrupted committee $\mathcal{C}_{i+1}$ parties their shares. Thus, the view of the adversary is identical in the real and ideal worlds for this step. For these degree-t sharings, there are $t+1$ total (random) degrees of freedom in defining the underlying polynomial, and thus the t shares that the adversary sees are uniformly random, while leaving 1 remaining (random) degree of freedom for each sharing, or $2t+1$ in total.

In Step 3 of Π_{rand}, the honest parties output their shares of $([r_1]_{t'}^{\mathcal{C}_{i+1}}, \dots, [r_{t+1}]_{t'}^{\mathcal{C}_{i+1}})$. Let us examine what honest parties output for their shares of the degree-t' sharings in the ideal world. In Step 3 of $\mathcal{S}$, for $j \in \mathcal{T}_{\mathcal{C}_i}$, it uses the underlying first $t+1$ honest parties' shares s_j^k received from the adversary to reconstruct $[s_j]_t^{\mathcal{C}_{i+1}}$. Then, in Step 4, $\mathcal{S}$ uses the differences between all of the $2t+1$ shares $s_j^k, k \in \mathcal{H}_{\mathcal{C}_{i+1}}$ received from the adversary and those defined by $[s_j]_t^{\mathcal{C}_{i+1}}$ reconstructed above, to reconstruct sharings $[d_j]_{t'_j}^{\mathcal{C}_{i+1}}$ (of course, for the first $t+1$, the difference will be 0). Next, in Step 5, $\mathcal{S}$ computes

$$([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_n]_t^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([s_1]_t^{\mathcal{C}_{i+1}}, \dots, [s_n]_t^{\mathcal{C}_{i+1}})^\intercal$$

and

$$([e_1]_{t'}^{\mathcal{C}_{i+1}}, \dots, [e_n]_{t'}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot \left(0, \dots, 0, [d_{2t+2}]_{t'_{2t+2}}^{\mathcal{C}_{i+1}}, \dots, [d_n]_{t'_n}^{\mathcal{C}_{i+1}}\right)^\intercal .$$

Finally, in Step 6, $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{rand}}$ the corrupted parties' shares of $([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_{t+1}]_t^{\mathcal{C}_{i+1}})$, along with $([e_1]_{t'}^{\mathcal{C}_{i+1}}, \dots, [e_{t+1}]_{t'}^{\mathcal{C}_{i+1}})$. $\mathcal{F}_{\mathsf{rand}}$ then for each $j \in [t+1]$, samples random r_j and degree-t sharing consistent with this value and the t corrupted parties' shares of $[r_j]_t^{\mathcal{C}_{i+1}}$ received from $\mathcal{S}$, reconstructs the corresponding sharing, and sends to the honest parties of $\mathcal{C}_{i+1}$ their shares of this sharing, plus their share of $[e_j]_{t'}^{\mathcal{C}_{i+1}}$. Now, let $\mathcal{H}_{\mathcal{C}_i} = \{h_1, \dots, h_{2t+1}\}$, $H_1 = \{h_1, \dots, h_{t+1}\}$, $C = [n] \setminus H_1 = \{c_1, \dots, c_{2t}\}$, and $\mathcal{T}_{\mathcal{C}_i} = \{\tau_1, \dots, \tau_t\}$ (all ordered). For each $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, let us look at honest party P_k's computation of $(r_1^k, \dots, r_{t+1}^k)^\intercal \leftarrow M_{[t+1]}^{H_1} \cdot (s_{h_1}^k \dots s_{h_{t+1}}^k)^\intercal + M_{[t+1]}^{C} \cdot (s_{c_1}^k \dots s_{c_{2t}}^k)^\intercal$. Since M is hyper-invertible, $M_{[t+1]}^{H_1}$ is invertible, which means that the $t+1$ values r_j^k have a one-to-one correspondence with $s_{h_j}^k$, for $j \in [t+1]$. Since for each sharing $[s_{h_j}]_t^{\mathcal{C}_{i+1}}$, $j \in [t+1]$ there is 1 remaining (random) degree of freedom, we can for the first honest party, conclude that r_j^1 is random. Since $\mathcal{F}_{\mathsf{rand}}$ only needs to sample 1 random value for each of the $t+1$ random sharings $([r_1]_t^{\mathcal{C}_{i+1}}, \dots, [r_{t+1}]_t^{\mathcal{C}_{i+1}})$, the real world and ideal world are distributed identically in determining these sharings. Note also that $[s_{h_{t+2}}]_t^{\mathcal{C}_{i+1}}, \dots, [s_{h_{2t+1}}]_t^{\mathcal{C}_{i+1}}$ still have 1 remaining (random) degrees of freedom, each, at this point. Furthermore, observe that for $k \in \mathcal{H}_{\mathcal{C}_{i+1}}, j \in \mathcal{T}_{\mathcal{C}_i}$ the k-th share of $[s_j]_t^{\mathcal{C}_{i+1}}$ added to that of $[d_j]_{t'_j}^{\mathcal{C}_{i+1}}$, will be exactly that s_j^k received from the adversary. Moreover, by linearity, the k-th shares of $M \cdot ([s_1]_t^{\mathcal{C}_{i+1}}, \dots, [s_n]_t^{\mathcal{C}_{i+1}})^\intercal + M \cdot (0, \dots, 0, [d_{2t+2}]_{t'_j}^{\mathcal{C}_{i+1}}, \dots, [d_n]_{t'_j}^{\mathcal{C}_{i+1}})^\intercal$ will thus be exactly $M \cdot (s_1^k, \dots, s_n^k)^\intercal$. Therefore, the values output by the honest parties in the ideal world are distributed identically to those output in the real world.

In Step 1 of the **Verification Phase** of Π_{rand}, for each member of $\upsilon_j \in [t+2, n] \cap \mathcal{T}_{\mathcal{C}_{i+1}} = T = \{\upsilon_1, \dots, \upsilon_m\}$, the adversary receives on behalf of the corresponding corrupted party of $\mathcal{C}_{i+2}$: the honest parties' shares of $[r_{\upsilon_j}]_{t'}^{\mathcal{C}_{i+1}}$. Let $H' = \{h_{t+2}, \dots, h_{t+m+1}\}$ and $C' = [n] \setminus H' = \{\sigma_1, \dots, \sigma_{n-m}\}$. For these sharings, for $k \in H_1$, let us look at honest party P_k's computation of $(r_{\upsilon_1}^k, \dots, r_{\upsilon_m}^k)^\intercal \leftarrow M_T^{H'} \cdot (s_{h_{t+2}}^k \dots s_{h_{t+m+1}}^k)^\intercal + M_T^{C'} \cdot (s_{\sigma_1}^k \dots s_{\sigma_{n-m}}^k)^\intercal$. Since M is hyper-invertible, $M_T^{H'}$ is invertible, which means that the m values $r_{\upsilon_1}^k, \dots, r_{\upsilon_m}^k$ have a one-to-one correspondence with $s_{h_l}^k$, for $l \in [t+2, t+m+1]$. Since for each sharing

$[s_{h_l}]_t^{\mathcal{C}_{i+1}}$, $l \in [t+2, t+m+1]$ there is 1 remaining (random) degree of freedom, we can for $j \in T$, conclude that the share of the first honest party $r_j^{h_1}$ is random. $\mathcal{S}$ in Step 7 computes and sends the simulated shares of honest parties of $[r_{v_j}]_{t'}^{\mathcal{C}_{i+1}}$ by separating $[r_{v_j}]_{t'}^{\mathcal{C}_{i+1}}$ into $[r_{v_j}]_t^{\mathcal{C}_{i+1}} + [e_{v_j}]_{t'}^{\mathcal{C}_{i+1}}$. For $[r_{v_j}]_t^{\mathcal{C}_{i+1}}$, since the first honest party share is random using the same argument as above, and the last t are consistent with the first, and the adversary's t shares, the ideal world is identically distributed to the real world at this point.

When $\mathcal{S}$ receives from the adversary, for $j \in [t+2, n] \cap \mathcal{H}_{\mathcal{C}_{i+1}}$, the corrupted parties' shares of $[r_j]_{t'}^{\mathcal{C}_{i+1}}$, it performs the same consistency check on behalf of the honest parties as they do in the real world. Thus, the distribution of abort messages from the honest parties of $\mathcal{C}_{i+2}$ to the corrupt parties of $\mathcal{C}_{i+3}$, and whether the honest parties of $\mathcal{C}_{i+3}$ abort, are identical in the real and ideal worlds.

Finally, we need to show that if for some $j \in [t+1]$, $[e_j]_{t'}^{\mathcal{C}_{i+1}} \neq [0]_0^{\mathcal{C}_{i+1}}$ (i.e., the all-0 sharing) in the real world, then the honest parties of committee $\mathcal{C}_{i+3}$ abort (as they are forced to in the ideal world). We do so by proving the contrapositive; i.e., if the honest parties do not abort, then for every $j \in [t+1]$, $[e_j]_{t'}^{\mathcal{C}_{i+1}} = [0]_0^{\mathcal{C}_{i+1}}$. For this, we again use the power of hyper-invertible matrix M. Let $H'' = \{\eta_1, \ldots, \eta_t\}$ be the first t honest parties of $[t+2, n] \cap \mathcal{H}_{\mathcal{C}_{i+2}}$. For each $k \in [n]$, party P_k of committee $\mathcal{C}_{i+1}$ computes

$$(r_{\eta_1}^k, \ldots, r_{\eta_t}^k)^\intercal \leftarrow M_{H''}^{\mathcal{T}_{\mathcal{C}_i}} \cdot (s_{\tau_1}^k \ldots s_{\tau_t}^k)^\intercal + M_{H''}^{\mathcal{H}_{\mathcal{C}_i}} \cdot (s_{h_1}^k \ldots s_{h_{2t+1}}^k)^\intercal.$$

Since M is hyper-invertible, $M_{H''}^{\mathcal{T}_{\mathcal{C}_i}}$ is invertible, so we can write

$$\begin{aligned}(s_{\tau_1}^k \ldots s_{\tau_t}^k)^\intercal = \left(M_{H''}^{\mathcal{T}_{\mathcal{C}_i}}\right)^{-1} \cdot (r_{\eta_1}^k, \ldots, r_{\eta_t}^k)^\intercal - \\ \left(M_{H''}^{\mathcal{T}_{\mathcal{C}_i}}\right)^{-1} \cdot M_{H''}^{\mathcal{H}_{\mathcal{C}_i}} \cdot (s_{h_1}^k \ldots s_{h_{2t+1}}^k)^\intercal.\end{aligned}$$

Now, since the honest parties did not abort, their checks passed, which means that for $j \in [t]$, the shares $r_{\eta_j}^1, \ldots, r_{\eta_j}^n$ are t-consistent. In particular, this means that for each $j' \in [t]$, the vector of the $\tau_{j'}$-th elements of $\left(M_{H''}^{\mathcal{T}_{\mathcal{C}_i}}\right)^{-1} \cdot (r_{\eta_1}^k, \ldots, r_{\eta_t}^k)^\intercal$ across $k \in [n]$ are t-consistent. Similarly, for $h_l \in \mathcal{H}_{\mathcal{C}_i}$, $s_{h_l}^1, \ldots, s_{h_l}^n$ are generated by honest parties and thus are t-consistent. Therefore, for each $j' \in [t]$, the vector of the j'-th elements of $\left(M_{H''}^{\mathcal{T}_{\mathcal{C}_i}}\right)^{-1} \cdot M_{H''}^{\mathcal{H}_{\mathcal{C}_i}} \cdot (s_{h_1}^k \ldots s_{h_{2t+1}}^k)^\intercal$ across $k \in [n]$ are t-consistent. Putting together the observations above, we have that for each $j' \in [t]$, the vector of the j'-th elements of $(s_{\tau_1}^k \ldots s_{\tau_t}^k)^\intercal$ across $k \in [n]$ are t-consistent.

Finally, since for $k \in [n]$:

$$(r_1^k, \ldots, r_{t+1}^k)^\intercal \leftarrow M_{[t+1]}^{\mathcal{T}_{\mathcal{C}_i}} \cdot (s_{\tau_1}^k \ldots s_{\tau_t}^k)^\intercal + M_{[t+1]}^{\mathcal{H}_{\mathcal{C}_i}} \cdot (s_{h_1}^k \ldots s_{h_{2t+1}}^k)^\intercal,$$

we see that for $j \in [t+1]$, $(r_j^1, \ldots, r_j^n)$ are t-consistent. Thus, every $[e_j]_{t'}^{\mathcal{C}_{i+1}} = [0]_0^{\mathcal{C}_{i+1}}$, as required.

In conclusion, we have proved that the real and ideal worlds are distributed identically and thus Π_{rand} UC-realizes $\mathcal{F}_{\mathsf{rand}}$. $\square$

C.2 Instantiating $\mathcal{F}_{\mathsf{input}}$

Protocol 6: Π_{input}

Usage: A client in the client set $\mathcal{C}_{\mathsf{clnt}}$ distributes a sharing $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ of their circuit input x to the other clients.

1. The clients P_j of $\mathcal{C}_{\mathsf{clnt}}$ first invoke $\mathcal{F}_{\mathsf{rand}}$ to get random sharing $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ (where $\mathcal{C}_{\mathsf{clnt}}$ acts as all committees used in $\mathcal{F}_{\mathsf{rand}}$).
2. The clients then open $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ to the input client, who then (if $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ is a correct degree-t sharing) computes $x + r$, samples $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ and distributes to all other clients in $\mathcal{C}_{\mathsf{clnt}}$ their shares.
3. Each party in $\mathcal{C}_{\mathsf{clnt}}$ then computes and outputs their sharing of x as $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} = [x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} - [r]_t^{\mathcal{C}_{\mathsf{clnt}}}$.

Lemma 10 (Lemma 2 re-stated)**.** *Π_{input} UC-realizes $\mathcal{F}_{\mathsf{input}}$ in the $\mathcal{F}_{\mathsf{rand}}$-hybrid model.*

Proof. First, we define the simulator $\mathcal{S}$:

1. $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{rand}}$. That is, it receives from the adversary shares r^j for $j \in \mathcal{T}_{\mathcal{C}_{\mathsf{clnts}}}$ and sharing $[e]_{t'}^{\mathcal{C}_{\mathsf{clnt}}}$, then samples r uniformly at random, generates random sharing $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ such that each j-th share is r^j, aborts if $[e]_{t'}^{\mathcal{C}_{\mathsf{clnt}}} \neq [0]_0^{\mathcal{C}_{\mathsf{clnt}}}$ and finally asks the adversary whether to continue. If the adversary responds with abort, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{input}}$.
2. Then, if the input client is corrupt:
   (a) $\mathcal{S}$ sends the honest clients' shares of $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ to the adversary.
   (b) If the client responds with abort to honest clients $A \in \mathcal{H}_{\mathcal{C}_{\mathsf{clnts}}}$, $\mathcal{S}$ forwards it (abort, A) to $\mathcal{F}_{\mathsf{input}}$.
   (c) Otherwise, the client responds with the honest parties' shares of $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ ($2t+1$ of them), from which $\mathcal{S}$ reconstructs the whole sharing and then forwards the sharing of $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} = [x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} - [r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ to $\mathcal{F}_{\mathsf{input}}$.
3. Otherwise, if the input client is honest:
   (a) $\mathcal{S}$ receives from the adversary the corrupt parties' shares of $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$.
   (b) If any of the shares it receives are not equal to r^j received above, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathsf{input}}$.
   (c) Otherwise, $\mathcal{S}$ sends random s^j for $j \in \mathcal{T}_{\mathcal{C}_{\mathsf{clnts}}}$ to the adversary and on behalf of the corrupt parties, computes their shares $s^j - r^j$ and forwards them to $\mathcal{F}_{\mathsf{input}}$.

Now we argue that the real world and ideal world are distributed identically to the adversary. It is clear that $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{rand}}$ exactly. If the input client is corrupt, then in both the real world and ideal world, the adversary first receives from the honest clients their shares of $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ (in the latter case, via the simulator $\mathcal{S}$). Then, based on the adversary's sharing $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$, the honest clients in both worlds output $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} - [r]_t^{\mathcal{C}_{\mathsf{clnt}}}$. Therefore, the two worlds are distributed identically in this case.

If the client is honest, then in the real world, the client first checks if the shares of $[r]_t^{\mathcal{C}_{\mathsf{clnt}}}$ it receives are t-consistent. In the ideal world, since the only corrupt party shares that are t-consistent with the honest parties' shares are those that $\mathcal{S}$ received from the adversary originally, $\mathcal{S}$ properly aborts if it does not receive these shares from the adversary. Then, in the real world, the corrupt parties receive from the honest client their shares of fresh sharing $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$. Since this is a fresh sharing, by the properties of Shamir secret sharing, the corrupt parties' shares are uniformly random, which is what $\mathcal{S}$ sends to the adversary in the ideal world. Finally, based on the sharing $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$, the honest clients in the real world output $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} - [r]_{t}^{\mathcal{C}_{\mathsf{clnt}}}$. Note that the adversary only has t shares of $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$, and thus, there are at least t random degrees of freedom left in it. Therefore, since $\mathcal{S}$ computes the corrupt parties' shares of $[x+r]_{2t}^{\mathcal{C}_{\mathsf{clnt}}} - [r]_{t}^{\mathcal{C}_{\mathsf{clnt}}}$ exactly as in the real world, as $s^j - r^j$, then $\mathcal{F}_{\mathsf{input}}$ computes $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ based on these t shares, the value x, and t other randomly sampled shares; the shares of $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$ output by honest clients in the ideal world are distributed identically to those of the real world.

Therefore, the real and ideal worlds are distributed identically. □

C.3 Instantiating $\mathcal{F}_{\mathsf{output}}$

Protocol 7: Π_{output}

Usage: The last committee $\mathcal{C}_\ell$ reconstructs to a client the sharing $[z]_{2t}^{\mathcal{C}_\ell}$ of their output z.

1. The last committee $\mathcal{C}_\ell$ simply opens output $[z]_{2t}^{\mathcal{C}_\ell}$ to the client, P_j of $\mathcal{C}_{\mathsf{clnt}}$.
2. The client P_j then checks if $[z]_{2t}^{\mathcal{C}_\ell}$ is a correct degree-$2t$ sharing. If so, it outputs z; otherwise, it sends abort to all other clients in $\mathcal{C}_{\mathsf{clnt}}$.

Lemma 11 (Lemma 3 re-stated)**.** Π_{output} *UC-realizes* $\mathcal{F}_{\mathsf{output}}$.

Proof. First, we define the simulator $\mathcal{S}$:

1. If the client is corrupted:
   (a) $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{output}}$ the whole sharing $[z]_{2t}^{\mathcal{C}_\ell}$ and forwards to the adversary the honest parties' shares.
   (b) Finally, if $\mathcal{S}$ receives on behalf of the honest clients $A \subseteq \mathcal{H}_{\mathcal{C}_{\mathsf{clnts}}}$, abort, from the adversary, $\mathcal{S}$ sends (abort, A) to $\mathcal{F}_{\mathsf{output}}$.
2. If the client is honest:
   (a) $\mathcal{S}$ first receives from $\mathcal{F}_{\mathsf{output}}$ the corrupt parties' shares $\{z_j\}_{j\in\mathcal{T}_{\mathcal{C}_\ell}}$ of $[z]_{2t}^{\mathcal{C}_\ell}$.
   (b) Then, $\mathcal{S}$ receives on behalf of the honest client $\{z'_j\}_{j\in\mathcal{T}_{\mathcal{C}_\ell}}$ from the adversary.
   (c) If any $z'_j \neq z_j$, then $\mathcal{S}$ sends abort to the corrupt clients of $\mathcal{T}_{\mathcal{C}_{\mathsf{clnts}}}$ and then also to $\mathcal{F}_{\mathsf{output}}$.

Now we show that the real and ideal world are identically distributed to the adversary. If the client is corrupted, it is clear that in both worlds, the adversary receives the honest parties' shares of $[z]_{2t}^{\mathcal{C}_\ell}$. Thus, the two worlds are identically distributed in this case.

If the client is honest, in the real world they receive all of the committee $\mathcal{C}_\ell$ parties' shares of $[z]_{2t}^{\mathcal{C}_\ell}$ and if the shares are not $2t$-consistent, then the client sends abort to all other clients; otherwise, they output z. In the ideal world $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{output}}$ the shares of the corrupt parties of $\mathcal{C}_\ell$ that are (the only shares that are) $2t$-consistent with the honest parties. Therefore, since $\mathcal{S}$ aborts if and only if any shares it receives from the adversary are different from those received from $\mathcal{F}_{\mathsf{output}}$, the simulation is perfect in this case. □

D Missing Proofs

Lemma 12 (Lemma 4 re-stated)**.** $\Pi_{\mathsf{robust\text{-}packed\text{-}reshare}}$ *UC-realizes* $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$.

Proof. First, we define the simulator $\mathcal{S}$:

1. $\mathcal{S}$ first receives from $\mathcal{F}_{\textsf{robust-packed-reshare}}$ the shares of the corrupted parties of $([x_1]_{2t}^{\mathcal{C}_i}, \dots, [x_{t+1}]_{2t}^{\mathcal{C}_i})$: $(x_1^j, \dots, x_{t+1}^j)_{j \in \mathcal{T}_{\mathcal{C}_i}}$

2. For each $j' \in \mathcal{H}_{\mathcal{C}_i}$ and $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, $\mathcal{S}$ samples random $x_{j'k'}$ and sends it to the adversary, to emulate $P_{k'}$'s share of $P_{j'}$'s packed sharing $[\![\boldsymbol{x}_{[1,t+1]}^{j'}]\!]_{2t}^{\mathcal{C}_{i+1}}$ in Step 1 of $\Pi_{\textsf{robust-packed-reshare}}$.

3. $\mathcal{S}$ then receives from the adversary for each $j \in \mathcal{T}_{\mathcal{C}_i}$ and $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, value x_{jk}, corresponding to P_k's share of P_j's packed sharing $[\![\hat{\boldsymbol{x}}_{[1,t+1]}^{j}]\!]_{2t}^{\mathcal{C}_{i+1}}$. For each j, $\mathcal{S}$ uses the $2t+1$ values x_{jk} to compute the sharing $[\![\hat{\boldsymbol{x}}_{[1,t+1]}^{j}]\!]_{2t}^{\mathcal{C}_{i+1}} = (x_{j1}, \dots, x_{jn})$, and the underlying block of secrets $\hat{\boldsymbol{x}}_{[1,t+1]}^{j} = (\hat{x}_1^j, \dots, \hat{x}_{t+1}^j)$.

4. Next, $\mathcal{S}$ computes for each $j \in \mathcal{T}_{\mathcal{C}_i}$, their error vector
$$\boldsymbol{e}_j = (e_{j,1}, \dots, e_{j,t+1}) \leftarrow (\hat{x}_1^j - x_1^j, \dots, \hat{x}_{t+1}^j - x_{t+1}^j),$$
and then the overall error vector
$$\boldsymbol{e} = \sum_{j \in \mathcal{T}_{\mathcal{C}_i} \cap [2t+1]} L_j(0) \cdot \boldsymbol{e}_j.$$

5. $\mathcal{S}$ then computes for $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, $\boldsymbol{x}^{k'} = \sum_{j=1}^{2t+1} L_j(0) \cdot x_{jk'}$ and sends these values, along with $\boldsymbol{e}$ to $\mathcal{F}_{\textsf{robust-packed-reshare}}$.

6. Next, $\mathcal{S}$ computes for each $\alpha \in [t+2]$, the matrix-vector product
$$(y_{1,\alpha}, \dots, y_{n-2t-1,\alpha})^\intercal \leftarrow H \cdot (e_{1,\alpha}, \dots, e_{n,\alpha})^\intercal,$$
where $e_{j,\alpha} = 0$ for all $j \in \mathcal{H}_{\mathcal{C}_i}, \alpha \in [t+1]$, and defines the vector $\boldsymbol{y}_\mu = (y_{\mu,1}, \dots, y_{\mu,t+1})$ for $\mu \in [n-2t-1]$.

7. $\mathcal{S}$ also computes for each $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$:
$$(y_1^{k'}, \dots, y_{n-2t-1}^{k'})^\intercal \leftarrow H \cdot (x_{1k'}, \dots, x_{nk'})^\intercal.$$

8. For each $\mu \in [n-2t-1]$, using the blocks $\boldsymbol{y}_\mu$ and the shares $y_\mu^{k'}$ of $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$ (i.e., $2t+1$ total degrees of freedom), $\mathcal{S}$ computes the sharing $[\![\boldsymbol{y}_\mu]\!]_{2t}^{\mathcal{C}_{i+1}}$.

9. Next, $\mathcal{S}$ computes $([\![\boldsymbol{z}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{z}_n]\!]_{2t}^{\mathcal{C}_{i+1}}) \leftarrow M \cdot ([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})$, and sends the honest parties' shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$ to each corrupted party P_l of committee $\mathcal{C}_{i+2}$.

10. For each honest party P_l of committee $\mathcal{C}_{i+2}$, $\mathcal{S}$ receives from the adversary: committee $\mathcal{C}_{i+1}$ corrupted parties' shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$. If these shares do not match what $\mathcal{S}$ had already computed, or the underlying computed value $\boldsymbol{z}_l \neq (0, \dots, 0)$, $\mathcal{S}$ sends abort on behalf of P_l to all corrupted parties of committee $\mathcal{C}_{i+3}$.

11. Finally, if $\mathcal{S}$ sent any abort in the above step it sets $A = \mathcal{H}_{\mathcal{C}_{i+3}}$; otherwise, it sets A to be those honest parties that receive abort from any corrupt party abort. Then, when $\mathcal{F}_{\textsf{robust-packed-reshare}}$ asks $\mathcal{S}$ whether to continue, it replies with abort.

Now we show that the real world and ideal world are identically distributed to the adversary. In Step 1 of $\Pi_{\textsf{robust-packed-reshare}}$, the adversary receives the t committee $\mathcal{C}_{i+1}$ corrupted parties' shares of the committee $\mathcal{C}_i$ honest parties' packed sharings $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$. Since these are degree-$2t$ packed sharings that share $t+1$ underlying values, there are t remaining degrees of freedom in defining the underlying polynomial, and thus the shares that the adversary sees are uniformly random. Indeed, Step 2 of $\mathcal{S}$ is to send uniformly random shares to the adversary, corresponding to the packed sharings, and thus the view of the adversary is identical in the real and ideal worlds for this step.

In Step 2 of $\Pi_{\textsf{robust-packed-reshare}}$, the honest parties output their shares $[\![\boldsymbol{x}]\!]^{\mathcal{C}_{i+1}}_{2t} = \sum_{j=1}^{2t+1} L_j(0) \cdot [\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$ based on their own as well as possibly some corrupted sharings $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$. The honest parties' shares uniquely define a degree-$2t$ polynomial, where for $\alpha \in [t+1]$ the polynomial evaluated on $-\alpha$ gives

$$\sum_{k\in\mathcal{H}_{\mathcal{C}_{i+1}}} L_k(-\alpha) \sum_{j=1}^{2t+1} L_j(0)\cdot x_{jk} = \sum_{j=1}^{2t+1} L_j(0) \sum_{k\in\mathcal{H}_{\mathcal{C}_{i+1}}} L_k(-\alpha) x_{jk} = \sum_{j=1}^{2t+1} L_j(0)\hat{x}^j_\alpha$$

$$= \sum_{j\in\mathcal{H}_{\mathcal{C}_{i+1}}\cap[2t+1]} L_j(0) x^j_\alpha + \sum_{j\in\mathcal{T}_{\mathcal{C}_{i+1}}\cap[2t+1]} L_j(0)\cdot(x^j_\alpha + e_{j,\alpha}) = x_\alpha + \sum_{j\in\mathcal{T}_{\mathcal{C}_{i+1}}\cap[2t+1]} L_j(0)\cdot e_{j,\alpha},$$

where $e_{j,\alpha}$ is some error injected by the adversary for corrupt party P_j. Similarly, for $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, the polynomial evaluated on k', i.e., $P_{k'}$'s share, gives

$$\sum_{j=1}^{2t+1} L_j(0) \sum_{k\in\mathcal{H}_{\mathcal{C}_{i+1}}} L_k(k') x_{jk} = \sum_{j=1}^{2t+1} L_j(0) x_{jk'},$$

where $x_{jk'}$ is the corrupted party $P_{k'}$'s share of $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$. In the ideal world, $\mathcal{S}$ in Step 3 for $j \in \mathcal{T}_{\mathcal{C}_i}$ reconstructs the entire packed sharings $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$. This allows $\mathcal{S}$ to reconstruct the corresponding corrupted party $P_{k'}$'s shares of each of these packed sharings, $x_{jk'}$, as well as the underlying $(\hat{x}^j_1, \dots, \hat{x}^j_{t+1})$, which are supposed to be P_j's shares of the original $(t+1)$ standard sharings. The former allows it to, along with the corrupted party's shares of honest party's sharings $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$, compute the corrupted parties' shares of $[\![\boldsymbol{x}]\!]^{\mathcal{C}_{i+1}}_{2t}$, which correspond to exactly those in the real world, as computed above. The latter allows it to compute the error $\boldsymbol{e}_j$ of these shares, as compared to those received from $\mathcal{F}_{\textsf{robust-packed-reshare}}$ in Step 1, and from this, the overall error $\boldsymbol{e}$, which is exactly the error in the real world, as computed above. Finally, $\mathcal{S}$, gives to $\mathcal{F}_{\textsf{robust-packed-reshare}}$, the corrupted party's shares of $[\![\boldsymbol{x}]\!]^{\mathcal{C}_{i+1}}_{2t}$, and the error $\boldsymbol{e}$, from which $\mathcal{F}_{\textsf{robust-packed-reshare}}$ further computes the perturbed underlying vector $\boldsymbol{x} + \boldsymbol{e}$. This gives $\mathcal{F}_{\textsf{robust-packed-reshare}}$ $2t+1$ points on a degree-$2t$ polynomial, which are exactly the same in the real world, and based on this $\mathcal{F}_{\textsf{robust-packed-reshare}}$ reconstructs $[\![\boldsymbol{x}]\!]^{\mathcal{C}_{i+1}}_{2t}$ and gives honest parties their shares. Thus, the shares output by the honest parties in the real and ideal world are distributed identically.

In Step 2 of the **Verification Phase** of $\Pi_{\textsf{robust-packed-reshare}}$, the adversary receives on behalf of corrupt party P_l of committee $\mathcal{C}_{i+2}$, the honest parties' shares of $[\![\boldsymbol{z}_l]\!]^{\mathcal{C}_{i+1}}_{2t}$. These shares come from them computing

$$M \cdot H \cdot \left([\![\hat{\boldsymbol{x}}^1_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}, \dots, [\![\hat{\boldsymbol{x}}^n_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}\right)^{\mathsf{T}} =$$

$$M \cdot H \cdot \left(\left([\![\boldsymbol{x}^1_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}, \dots, [\![\boldsymbol{x}^n_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}\right)^{\mathsf{T}} + \left([\![\boldsymbol{e}_1]\!]^{\mathcal{C}_{i+1}}_{2t}, \dots, [\![\boldsymbol{e}_n]\!]^{\mathcal{C}_{i+1}}_{2t}\right)^{\mathsf{T}}\right),[4]$$

[4]Such decomposition of the maliciously shared $[\![\hat{\boldsymbol{x}}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t}$ is always possible since we can define some canonical packed sharing $[\![\boldsymbol{x}^j_{[1,t+1]}]\!]^{\mathcal{C}_{i+1}}_{2t} := (x_{j1}, \dots, x_{jn})$, and $[\![\boldsymbol{e}_j]\!]^{\mathcal{C}_{i+1}}_{2t}$ as the rest.

where $[\![\boldsymbol{e}_j]\!]_{2t}^{\mathcal{C}_{i+1}} = 0$ for $j \in \mathcal{H}_{\mathcal{C}_i}$,

$$= ([\![\boldsymbol{0}]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{0}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal + M \cdot H \cdot \left([\![\boldsymbol{e}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{e}_n]\!]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal,$$

where the identity of the first term is due to the fact that the underlying shares $x_\alpha^1, \dots, x_\alpha^n$ that are packed into the α-th slot of the respective packed sharings $[\![\boldsymbol{x}_{[1,t+1]}^j]\!]_{2t}^{\mathcal{C}_{i+1}}$, for $\alpha \in [t+1]$ indeed correspond to valid points of a polynomial of degree $\le 2t$, and therefore applying H to them results in zeroes in the α-th slot of each element of the output vector.

In the ideal world, $\mathcal{S}$ computes $H \cdot (\boldsymbol{e}_1, \dots, \boldsymbol{e}_n)^\intercal$, where $\boldsymbol{e}_j = \boldsymbol{0}$ for $j \in \mathcal{H}_{\mathcal{C}_i}$, and the corrupted parties' shares of $([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow H \cdot \left([\![\hat{\boldsymbol{x}}_{[1,t+1]}^1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\hat{\boldsymbol{x}}_{[1,t+1]}^n]\!]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal$. This gives $\mathcal{S}$: $2t+1$ evaluations of the polynomials underlying $([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})$; from which it can reconstruct the whole sharings. It can then compute

$$([\![\boldsymbol{z}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{z}_n]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal,$$

from which it can send the corrupted parties P_l all of the honest parties' shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$. Thus, the real world and ideal world are identically distributed at this step.

In step 2 of the **Verification Phase** of $\Pi_{\text{robust-packed-reshare}}$, if the corrupted parties' shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$ are not consistent with those of the honest parties, then we know from the error-detection properties of packed sharings that honest party P_l will send abort to the parties of committee $\mathcal{C}_{i+3}$, and similarly if $\boldsymbol{z}_l \neq 0^{t+1}$. In the ideal world, $\mathcal{S}$ in Step 10, first checks if the corrupted parties' shares of $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$, for honest party P_l of $\mathcal{C}_{i+2}$, match that which it had already computed (i.e., are consistent with those of honest parties), and then also if $\boldsymbol{z}_l = (0, \dots, 0)$. If either of these checks fail, $\mathcal{S}$ sends abort on behalf of P_l to all corrupted parties of committee $\mathcal{C}_{i+3}$. Thus, the real world and ideal world are identically distributed at this step.

Finally, we need to show that if $\boldsymbol{e} \neq 0^{t+1}$, then the honest parties abort in the real world (as they are forced to in the ideal world). We do so by showing the contrapositive: i.e., if the honest parties in the real world do not abort, then $\boldsymbol{e} = 0$. If the honest parties do not abort, then it must be that for each of the $2t+1$ honest parties P_l in committee $\mathcal{C}_{i+2}$, $[\![\boldsymbol{z}_l]\!]_{2t}^{\mathcal{C}_{i+1}}$ are $2t$-consistent and correspond to $\boldsymbol{z}_l = 0^{t+1}$. By the error-correction properties of super-invertible matrix M, this must also mean that $\boldsymbol{y}_1 = \dots = \boldsymbol{y}_{n-2t-1} = 0^{t+1}$ (because any $n - 2t - 1 \le 2t + 1$ of the honest codeword symbols can be multiplied by the inverse of the corresponding submatrix of M to get back the original message, which therefore must be all zeroes). As written above, we have that $([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal =$

$$H \cdot \left(\left([\![\boldsymbol{x}_{[1,t+1]}^1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{x}_{[1,t+1]}^n]\!]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal + \left([\![\boldsymbol{e}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{e}_n]\!]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal\right)$$

$$= ([\![\boldsymbol{0}]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{0}]\!]_{2t}^{\mathcal{C}_{i+1}})^\intercal + H \cdot \left([\![\boldsymbol{e}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{e}_n]\!]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal,$$

where $[\![\boldsymbol{e}_j]\!]_{2t}^{\mathcal{C}_{i+1}} = 0$ for $j \in \mathcal{H}_{\mathcal{C}_i}$. Let $\mathcal{T}_{\mathcal{C}_i} = \{\tau_1, \dots, \tau_t\}$; E be the $n \times n$ matrix in which the rows corresponding to indices $\tau \in \mathcal{T}_{\mathcal{C}_i}$ are the shares of $[\![\boldsymbol{e}_\tau]\!]_{2t}^{\mathcal{C}_{i+1}}$, and the rows corresponding to indices $j \in \mathcal{H}_{\mathcal{C}_i}$ are the shares of $[\![\boldsymbol{e}_j]\!]_{2t}^{\mathcal{C}_{i+1}}$ (i.e., all 0's); and the matrix of shares of the sharings $([\![\boldsymbol{y}_1]\!]_{2t}^{\mathcal{C}_{i+1}}, \dots, [\![\boldsymbol{y}_{n-2t-1}]\!]_{2t}^{\mathcal{C}_{i+1}})$ be Y (each column is the set of shares held by each party). Then,

$$
\begin{aligned}
Y &= \begin{pmatrix} \boldsymbol{y}_1^1 & \cdots & \boldsymbol{y}_1^n \\ \vdots & \ddots & \vdots \\ \boldsymbol{y}_{n-2t-1}^1 & \cdots & \boldsymbol{y}_{n-2t-1}^n \end{pmatrix} = \begin{pmatrix} \mathbf{0}_1^1 & \cdots & \mathbf{0}_1^n \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n-2t-1}^1 & \cdots & \mathbf{0}_{n-2t-1}^n \end{pmatrix} + H \cdot E \\
&= \begin{pmatrix} \mathbf{0}_1^1 & \cdots & \mathbf{0}_1^n \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n-2t-1}^1 & \cdots & \mathbf{0}_{n-2t-1}^n \end{pmatrix} + H^{\mathcal{H}_{\mathcal{C}_i}} \cdot \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} + H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \begin{pmatrix} \boldsymbol{e}_{\tau_1}^1 & \cdots & \boldsymbol{e}_{\tau_1}^n \\ \vdots & \ddots & \vdots \\ \boldsymbol{e}_{\tau_t}^1 & \cdots & \boldsymbol{e}_{\tau_t}^n \end{pmatrix}.
\end{aligned}
$$

Testing if a vector $\boldsymbol{x} \in \mathbb{F}^{2t+1}$ corresponds to a degree-$2t$ packed sharing of $\mathbf{0} \in \mathbb{F}^{t+1}$ can be done by interpolating a polynomial f of degree $\leq 2t$ from the $2t+1$ entries, and checking that $f(-\alpha) = 0$ for each $\alpha \in [t+1]$. This can be directly expressed as a matrix product $\mathbf{0} = G \cdot \boldsymbol{x}$, where G is a $(n-2t) \times (2t+1)$ matrix (since $n - 2t = t+1$). Let $\mathcal{H}_{\mathcal{C}_{i+1}} = \{h_1, \ldots, h_{2t+1}\}$, $\mathcal{T}_{\mathcal{C}_{i+1}} = \{\eta_1, \ldots, \eta_t\}$, and G' be the $(n-2t) \times n$ matrix in which the columns corresponding to indices in $\mathcal{H}_{\mathcal{C}_{i+1}}$ are the $2t+1$ columns of G, and all other entries are 0. Since each $\boldsymbol{y}_j$ is equal to $\mathbf{0} \in \mathbb{F}^{t+1}$, the honest entries of each row of Y, multiplied by $G^\intercal$, gives zero, or in other words:

$$
\begin{aligned}
0 = \begin{pmatrix} \boldsymbol{y}_1^1 & \cdots & \boldsymbol{y}_1^n \\ \vdots & \ddots & \vdots \\ \boldsymbol{y}_{n-2t-1}^1 & \cdots & \boldsymbol{y}_{n-2t-1}^n \end{pmatrix} \cdot (G')^\intercal &= \overbrace{\begin{pmatrix} \mathbf{0}_1^{h_1} & \cdots & \mathbf{0}_1^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n-2t-1}^{h_1} & \cdots & \mathbf{0}_{n-2t-1}^{h_{2t+1}} \end{pmatrix} \cdot G^\intercal}^{0} \\
&\quad + \overbrace{\begin{pmatrix} \mathbf{0}_1^{\eta_1} & \cdots & \mathbf{0}_1^{\eta_t} \\ \vdots & \ddots & \vdots \\ \mathbf{0}_{n-2t-1}^{\eta_1} & \cdots & \mathbf{0}_{n-2t-1}^{\eta_t} \end{pmatrix} \cdot 0}^{0} + H \cdot E \cdot (G')^\intercal \\
&= H \cdot E \cdot (G')^\intercal. \\
&= H^{\mathcal{H}_{\mathcal{C}_i}} \cdot \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \cdot (G')^\intercal + H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \begin{pmatrix} \boldsymbol{e}_{\tau_1}^1 & \cdots & \boldsymbol{e}_{\tau_1}^n \\ \vdots & \ddots & \vdots \\ \boldsymbol{e}_{\tau_t}^1 & \cdots & \boldsymbol{e}_{\tau_t}^n \end{pmatrix} \cdot (G')^\intercal \\
&= H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \left(\begin{pmatrix} \boldsymbol{e}_{\tau_1}^{h_1} & \cdots & \boldsymbol{e}_{\tau_1}^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{e}_{\tau_t}^{h_1} & \cdots & \boldsymbol{e}_{\tau_t}^{h_{2t+1}} \end{pmatrix} \cdot G^\intercal + \begin{pmatrix} \boldsymbol{e}_{\tau_1}^{\eta_1} & \cdots & \boldsymbol{e}_{\tau_1}^{\eta_t} \\ \vdots & \ddots & \vdots \\ \boldsymbol{e}_{\tau_t}^{\eta_1} & \cdots & \boldsymbol{e}_{\tau_t}^{\eta_t} \end{pmatrix} \cdot 0 \right)
\end{aligned}
$$

However, denoting

$$
E' = \begin{pmatrix} \boldsymbol{e}_{\tau_1}^{h_1} & \cdots & \boldsymbol{e}_{\tau_1}^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{e}_{\tau_t}^{h_1} & \cdots & \boldsymbol{e}_{\tau_t}^{h_{2t+1}} \end{pmatrix},
$$

which is the $t \times (2t+1)$ matrix whose rows correspond to the *honest parties' shares* of the errors, this is not possible unless $E' \cdot G^\intercal = 0$. Otherwise, $E \cdot (G')^\intercal$ would contain one non-zero column of weight at most t that maps to zero under H, or in other words, this column is consistent with a polynomial of degree at most $2t$. This is not possible since this vector has at least $2t+1$ zero entries. As a result, we see that $E' \cdot G^\intercal = 0$, so the shares $[\![\boldsymbol{e}_j]\!]_{2t}^{\mathcal{C}_{i+1}}$ (of the honest parties) indeed are shares $[\![\mathbf{0}]\!]_{2t}^{\mathcal{C}_{i+1}}$, as desired. $\square$

Lemma 13 (Lemma 5 re-stated)**.** $\Pi_{\text{robust-standard-reshare}}$ *UC-realizes* $\mathcal{F}_{\text{robust-standard-reshare}}$.

Proof. First, we define the simulator $\mathcal{S}$:

1. $\mathcal{S}$ first receives from $\mathcal{F}_{\textsf{robust-standard-reshare}}$ the shares of the corrupted parties of $[\![\boldsymbol{x}]\!]_{2t}^{\mathcal{C}_i}$: $(\boldsymbol{x}^j)_{j\in\mathcal{T}_{\mathcal{C}_i}}$

2. For each $j' \in \mathcal{H}_{\mathcal{C}_i}$ and $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, $\mathcal{S}$ samples random $x_{j'k'}$ and sends it to the adversary.

3. $\mathcal{S}$ then receives from the adversary for each $j \in \mathcal{T}_{\mathcal{C}_i}$ and $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, value x_{jk}. For each j, $\mathcal{S}$ uses the $2t+1$ values x_{jk} to compute the sharing $[\widehat{\boldsymbol{x}^j}]_{2t}^{\mathcal{C}_{i+1}} = (x_{j1}, \dots, x_{jn})$, and the underlying secret $\widehat{\boldsymbol{x}^j}$ (which is a share of a packed sharing).

4. Next, $\mathcal{S}$ computes for each $j \in \mathcal{T}_{\mathcal{C}_i}$, their error
$$e_j \leftarrow \widehat{\boldsymbol{x}^j} - \boldsymbol{x}^j,$$
and then for $\alpha \in [t+1]$, the overall errors
$$e_\alpha \leftarrow \sum_{j\in\mathcal{T}_{\mathcal{C}_i}\cap[2t+1]} L_j(-\alpha)\cdot e_j.$$

5. $\mathcal{S}$ then computes for $\alpha \in [2t+1], k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, $x_\alpha^{k'} = \sum_{j=1}^{2t+1} L_j(-\alpha)\cdot x_{jk'}$ and sends these values along with $e_1, \dots, e_{t+1}$ to $\mathcal{F}_{\textsf{robust-standard-reshare}}$.

6. Next, $\mathcal{S}$ computes the matrix-vector product
$$(y_1, \dots, y_{n-2t-1})^\intercal \leftarrow H\cdot(e_1,\dots,e_n)^\intercal,$$
where $e_j = 0$ for $j \in \mathcal{H}_{\mathcal{C}_i}$

7. $\mathcal{S}$ also computes for each $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$:
$$(y_1^{k'}, \dots, y_{n-2t-1}^{k'})^\intercal \leftarrow H\cdot(x_{1k'},\dots,x_{nk'})^\intercal.$$

8. Additionally, $\mathcal{S}$ for the first t parties $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$ samples random $(y_1^k, \dots, y_{n-2t-1}^k)$.

9. For each $\mu \in [n-2t-1]$, using y_μ and the shares $y_\mu^{k'}$ of $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$ and y_μ^k of the first t parties $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$ (i.e., $2t+1$ total degrees of freedom), $\mathcal{S}$ computes the sharing $[y_\mu]_{2t}^{\mathcal{C}_{i+1}}$.

10. Next, $\mathcal{S}$ computes $([z_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [z_n]_{2t}^{\mathcal{C}_{i+1}}) \leftarrow M\cdot([y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})$, and sends the honest parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$ to each corrupted party P_l of committee $\mathcal{C}_{i+2}$.

11. For each honest party P_l of committee $\mathcal{C}_{i+2}$, $\mathcal{S}$ receives from the adversary: committee $\mathcal{C}_{i+1}$ corrupted parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$. If these shares do not match what $\mathcal{S}$ had already computed, or the underlying computed value $z_l \neq 0$, $\mathcal{S}$ sends $\textsf{abort}$ on behalf of P_l to all corrupted parties of committee $\mathcal{C}_{i+3}$.

12. Finally, if $\mathcal{S}$ sent any $\textsf{abort}$ in the above step, it sets $A \leftarrow \mathcal{H}_{\mathcal{C}_{i+3}}$; otherwise, it sets A to be those honest parties that receive from any corrupt party $\textsf{abort}$. Then, when $\mathcal{F}_{\textsf{robust-standard-reshare}}$ asks $\mathcal{S}$ whether to continue, it replies with $(\textsf{abort}, A)$.

Now we show that the real world and ideal world are identically distributed to the adversary. In Step 1 of $\Pi_{\textsf{robust-standard-reshare}}$, the adversary receives the t committee $\mathcal{C}_{i+1}$ corrupted parties' shares of the committee $\mathcal{C}_i$ honest parties' sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$. Since these are degree-$2t$ standard Shamir sharings, there are $2t$ remaining (random) degrees of freedom in defining the underlying polynomial, and thus the t shares that the adversary sees are uniformly random, while leaving t remaining (random) degrees of freedom for each sharing, or $t \cdot (2t+1)$ in total. Indeed, Step 2 of $\mathcal{S}$ is to send uniformly random shares to the adversary, corresponding to the sharings, and thus the view of the adversary is identical in the real and ideal worlds for this step.

In Step 2 of $\Pi_{\textsf{robust-standard-reshare}}$, the honest parties output for $\alpha \in [t+1]$ their shares $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}} = \sum_{j=1}^{2t+1} L_j(-\alpha) \cdot [\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$ based on their own as well as possibly some corrupted sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$. The honest parties' shares uniquely define a degree-$2t$ polynomial for $\alpha \in [t+1]$, which evaluated on 0 gives

$$\sum_{k \in \mathcal{H}_{\mathcal{C}_{i+1}}} L_k(0) \sum_{j=1}^{2t+1} L_j(-\alpha) \cdot x_{jk} = \sum_{j=1}^{2t+1} L_j(-\alpha) \sum_{k \in \mathcal{H}_{\mathcal{C}_{i+1}}} L_k(0) x_{jk} = \sum_{j=1}^{2t+1} L_j(-\alpha) \widehat{\boldsymbol{x}^j}$$

$$= \sum_{j \in \mathcal{H}_{\mathcal{C}_{i+1}} \cap [2t+1]} L_j(-\alpha) \cdot \boldsymbol{x}^j + \sum_{j \in \mathcal{T}_{\mathcal{C}_{i+1}} \cap [2t+1]} L_j(-\alpha) \cdot (\boldsymbol{x}^j + e_j) = x_\alpha + \sum_{j \in \mathcal{T}_{\mathcal{C}_{i+1}} \cap [2t+1]} L_j(-\alpha) \cdot e_j,$$

where e_j is some error injected by the adversary for corrupt party P_j. Similarly, for $k' \in \mathcal{T}_{\mathcal{C}_{i+1}}$, the polynomial evaluated on k', i.e., $P_{k'}$'s share, gives

$$\sum_{j=1}^{2t+1} L_j(-\alpha) \sum_{k \in \mathcal{H}_{\mathcal{C}_{i+1}}} L_k(k') x_{jk} = \sum_{j=1}^{2t+1} L_j(-\alpha) x_{jk'},$$

where $x_{jk'}$ is the corrupted party $P_{k'}$'s share of $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$. In the ideal world, $\mathcal{S}$ in Step 3 for $j \in \mathcal{T}_{\mathcal{C}_i}$ reconstructs the entire sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$. This allows $\mathcal{S}$ to reconstruct the corresponding corrupted party $P_{k'}$'s shares of each of these sharings, $x_{jk'}$, as well as the underlying $\widehat{\boldsymbol{x}^j}$, which is supposed to be P_j's share of the original packed sharing. The former allows it to, along with the corrupted party's shares of honest party's sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$, compute the corrupted parties' shares of $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, for $\alpha \in [t+1]$ which correspond to exactly those in the real world, as computed above. The latter allows it to compute the error e_j of these shares, as compared to those received from $\mathcal{F}_{\textsf{robust-standard-reshare}}$ in Step 1, and from this, the overall errors $e_1, \ldots, e_{t+1}$, which is exactly the error in the real world, as computed above. Finally, $\mathcal{S}$, gives to $\mathcal{F}_{\textsf{robust-standard-reshare}}$, for $\alpha \in [t+1]$, the corrupted party's shares of $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, and the error e_α, from which $\mathcal{F}_{\textsf{robust-standard-reshare}}$ further computes the perturbed underlying values $x_\alpha + e_\alpha$. This gives $\mathcal{F}_{\textsf{robust-standard-reshare}}$ $t+1$ points on degree-$2t$ polynomials, which are distributed the same as in the real world, and based on this and t other randomly sampled points on each polynomial, $\mathcal{F}_{\textsf{robust-standard-reshare}}$ reconstructs $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ and gives honest parties their shares. Note that we had $2t^2 + t$ total degrees of freedom above and this uses up only $t \cdot (t+1)$ of them. Thus, the shares output by the honest parties in the real and ideal world are distributed identically. Indeed, for the first $t+1$ honest parties $\alpha \in \mathcal{H}_{\mathcal{C}_i}$, we can use the t remaining random shares of $[\boldsymbol{x}^\alpha]_{2t}^{\mathcal{C}_{i+1}}$, $x_{\alpha 1}, \ldots, x_{\alpha t}$ to randomly perturb the shares of $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ of the first $t+1$ honest parties of $\mathcal{H}_{\mathcal{C}_{i+1}}$, as desired. Moreover, we have t^2 remaining (random) degrees of freedom, corresponding to the t remaining random shares of $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$, for the last $t+1$ honest parties $j \in \mathcal{H}_{\mathcal{C}_i}$.

In Step 2 of the **Verification Phase** of $\Pi_{\textsf{robust-standard-reshare}}$, the adversary receives on behalf of corrupt party P_l of committee $\mathcal{C}_{i+2}$, the honest parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$. These shares come from them computing

$$M \cdot H \cdot \left([\widehat{\boldsymbol{x}^1}]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [\widehat{\boldsymbol{x}^n}]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal =$$

$$M \cdot H \cdot \left(\left([\boldsymbol{x}^1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [\boldsymbol{x}^n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal + \left([e_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [e_n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal\right), [5]$$

where $[e_j]_{2t}^{\mathcal{C}_{i+1}} = 0$ for $j \in \mathcal{H}_{\mathcal{C}_i}$

$$= ([0]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [0]_{2t}^{\mathcal{C}_{i+1}})^\intercal + M \cdot H \cdot \left([e_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [e_n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal,$$

where the identity of the first term is due to the fact that the underlying shares $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n$ of the respective sharings $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$ indeed correspond to valid points of a polynomial of degree $\leq 2t$, and therefore applying H to them results in zeroes in each element of the output vector.

In the ideal world, $\mathcal{S}$ computes $H \cdot (e_1, \ldots, e_n)^\intercal$ and the corrupted parties' shares of $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow H \cdot \left([\widehat{\boldsymbol{x}^1}]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [\widehat{\boldsymbol{x}^n}]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal$. This gives $\mathcal{S}$: $t+1$ evaluations of the polynomials underlying $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})$; from which, with t more random evaluations, it can reconstruct the whole sharings. Recall that we had t^2 remaining degrees of freedom, and this is exactly t^2 random values, since $n - 2t - 1 = t$, so this results in an identical distribution as the real world. Indeed, for the last $t + 1$ honest parties $j \in \mathcal{H}_{\mathcal{C}_i}$, we can use the t remaining random shares of $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}}$, $x_{j1}, \ldots, x_{jt}$ to randomly perturb the first t shares of $[y_j]_{2t}^{\mathcal{C}_{i+1}}$, respectively, as desired. It can then compute

$$([z_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [z_n]_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([y_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal,$$

from which it can send the corrupted parties P_l all of the honest parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$. Thus, the real world and ideal world are identically distributed at this step.

In step 2 of the **Verification Phase** of $\Pi_{\text{robust-standard-reshare}}$, if the corrupted parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$ are not consistent with those of the honest parties, then we know from the error-detection properties of packed sharings that honest party P_l will send abort to the parties of committee $\mathcal{C}_{i+3}$, and similarly if $z_l \neq 0$. In the ideal world, $\mathcal{S}$ in Step 11, first checks if the corrupted parties' shares of $[z_l]_{2t}^{\mathcal{C}_{i+1}}$, for honest party P_l of $\mathcal{C}_{i+2}$, match that which it had already computed (i.e., are consistent with those of honest parties), and then also if $z_l = 0$. If either of these checks fail, $\mathcal{S}$ sends abort on behalf of P_l to all corrupted parties of committee $\mathcal{C}_{i+3}$. Thus, the real world and ideal world are identically distributed at this step.

Finally, we need to show that if any $e_\alpha \neq 0$, then the honest parties abort in the real world (as they are forced to in the ideal world). We do so by showing the contrapositive: i.e., if the honest parties in the real world do not abort, then each $e_\alpha = 0$. If the honest parties do not abort, then it must be that for each of the $2t + 1$ honest parties P_l in committee $\mathcal{C}_{i+2}$, $[z_l]_{2t}^{\mathcal{C}_{i+1}}$ are $2t$-consistent and correspond to $z_l = 0$. By the error-correction properties of super-invertible matrix M, this must also mean that $y_1 = \cdots = y_{n-2t-1} = 0$ (because any $n - 2t - 1 \leq 2t + 1$ of the honest codeword symbols can be multiplied by the inverse of the corresponding submatrix of M to get back the original message, which therefore must be all zeroes). As written above, we have that $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal =$

$$H \cdot \left(\left([\boldsymbol{x}^1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [\boldsymbol{x}^n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal + \left([e_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [e_n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal\right)$$

[5] Such decomposition of the maliciously shared $[\widehat{\boldsymbol{x}^j}]_{2t}^{\mathcal{C}_{i+1}}$ is always possible since we can define some canonical packed sharing $[\boldsymbol{x}^j]_{2t}^{\mathcal{C}_{i+1}} := (x_{j1}, \ldots, x_{jn})$, and $[e_j]_{2t}^{\mathcal{C}_{i+1}}$ as the rest.

$$= ([0]_{2t}^{\mathcal{C}_{i+1}}, \dots, [0]_{2t}^{\mathcal{C}_{i+1}})^\intercal + H \cdot \left([e_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [e_n]_{2t}^{\mathcal{C}_{i+1}}\right)^\intercal,$$

where $[e_j]_{2t}^{\mathcal{C}_{i+1}} = 0$ for $j \in \mathcal{H}_{\mathcal{C}_i}$. Let $\mathcal{T}_{\mathcal{C}_i} = \{\tau_1, \dots, \tau_t\}$; E be the $n \times n$ matrix in which the rows corresponding to indices $\tau \in \mathcal{T}_{\mathcal{C}_i}$ are the shares of $[e_\tau]_{2t}^{\mathcal{C}_{i+1}}$, and the rows corresponding to indices $j \in \mathcal{H}_{\mathcal{C}_i}$ are the shares of $[e_j]_{2t}^{\mathcal{C}_{i+1}}$ (i.e., all 0's); and the matrix of shares of the sharings $([y_1]_{2t}^{\mathcal{C}_{i+1}}, \dots, [y_{n-2t-1}]_{2t}^{\mathcal{C}_{i+1}})$ be Y (each column is the set of shares held by each party). Then,

$$Y = \begin{pmatrix} y_1^1 & \cdots & y_1^n \\ \vdots & \ddots & \vdots \\ y_{n-2t-1}^1 & \cdots & y_{n-2t-1}^n \end{pmatrix} = \begin{pmatrix} 0_1^1 & \cdots & 0_1^n \\ \vdots & \ddots & \vdots \\ 0_{n-2t-1}^1 & \cdots & 0_{n-2t-1}^n \end{pmatrix} + H \cdot E$$
$$= \begin{pmatrix} 0_1^1 & \cdots & 0_1^n \\ \vdots & \ddots & \vdots \\ 0_{n-2t-1}^1 & \cdots & 0_{n-2t-1}^n \end{pmatrix} + H^{\mathcal{H}_{\mathcal{C}_i}} \cdot \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} + H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \begin{pmatrix} e_{\tau_1}^1 & \cdots & e_{\tau_1}^n \\ \vdots & \ddots & \vdots \\ e_{\tau_t}^1 & \cdots & e_{\tau_t}^n \end{pmatrix}.$$

Testing if a vector $\boldsymbol{x} \in \mathbb{F}^{2t+1}$ corresponds to a degree-$2t$ sharing of 0 can be done by interpolating a polynomial f of degree $\leq 2t$ from the $2t+1$ entries, and checking that $f(0) = 0$. This can be directly expressed as an inner product $0 = \boldsymbol{g}^\intercal \cdot \boldsymbol{x}$, where $\boldsymbol{g}$ is a $(2t+1)$-dimensional vector. Let $\mathcal{H}_{\mathcal{C}_{i+1}} = \{h_1, \dots, h_{2t+1}\}$, $\mathcal{T}_{\mathcal{C}_{i+1}} = \{\eta_1, \dots, \eta_t\}$, and $\boldsymbol{g}'$ be the n-dimensional vector in which the elements corresponding to indices in $\mathcal{H}_{\mathcal{C}_{i+1}}$ are the $2t+1$ elements of $\boldsymbol{g}$ and all other entries are 0. Since each y_j is equal to 0, the honest entries of each row of Y, multiplied by $\boldsymbol{g}$, gives zero, or in other words:

$$0 = \begin{pmatrix} y_1^1 & \cdots & y_1^n \\ \vdots & \ddots & \vdots \\ y_{n-2t-1}^1 & \cdots & y_{n-2t-1}^n \end{pmatrix} \cdot \boldsymbol{g}' = \overbrace{\begin{pmatrix} 0_1^{h_1} & \cdots & 0_1^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ 0_{n-2t-1}^{h_1} & \cdots & 0_{n-2t-1}^{h_{2t+1}} \end{pmatrix} \cdot \boldsymbol{g}}^{0}$$
$$+ \overbrace{\begin{pmatrix} 0_1^{\eta_1} & \cdots & 0_1^{\eta_t} \\ \vdots & \ddots & \vdots \\ 0_{n-2t-1}^{\eta_1} & \cdots & 0_{n-2t-1}^{\eta_t} \end{pmatrix} \cdot 0}^{0} H \cdot E \cdot \boldsymbol{g}'$$
$$= H^{\mathcal{H}_{\mathcal{C}_i}} \cdot \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix} \cdot \boldsymbol{g}' + H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \begin{pmatrix} e_{\tau_1}^1 & \cdots & e_{\tau_1}^n \\ \vdots & \ddots & \vdots \\ e_{\tau_t}^1 & \cdots & e_{\tau_t}^n \end{pmatrix} \cdot \boldsymbol{g}'$$
$$= H^{\mathcal{T}_{\mathcal{C}_i}} \cdot \left(\begin{pmatrix} e_{\tau_1}^{h_1} & \cdots & e_{\tau_1}^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ e_{\tau_t}^{h_1} & \cdots & e_{\tau_t}^{h_{2t+1}} \end{pmatrix} \cdot \boldsymbol{g} + \begin{pmatrix} e_{\tau_1}^{\eta_1} & \cdots & e_{\tau_1}^{\eta_t} \\ \vdots & \ddots & \vdots \\ e_{\tau_t}^{\eta_1} & \cdots & e_{\tau_t}^{\eta_t} \end{pmatrix} \cdot 0 \right)$$

However, denoting

$$E' = \begin{pmatrix} e_{\tau_1}^{h_1} & \cdots & e_{\tau_1}^{h_{2t+1}} \\ \vdots & \ddots & \vdots \\ e_{\tau_t}^{h_1} & \cdots & e_{\tau_t}^{h_{2t+1}} \end{pmatrix},$$

which is the $t \times (2t+1)$ matrix whose rows correspond to the *honest parties' shares* of the errors, this is not possible unless $E' \cdot \boldsymbol{g} = 0$, otherwise, $E \cdot \boldsymbol{g}'$ would contain one non-zero column of weight at most t that maps to zero under H, or in other words, this column is consistent with a polynomial of degree at most $2t$. This is not possible since this vector has at least $2t+1$ zero entries. As a result, we see that $E' \cdot \boldsymbol{g} = 0$, so the shares $[e_j]_{2t}^{\mathcal{C}_{i+1}}$ (of the honest parties) indeed are shares $[0]_{2t}^{\mathcal{C}_{i+1}}$, as desired. ◻

Lemma 14 (Lemma 6, re-stated)**.** *Let it be the case that for every $\alpha \in [t+1]$, either the adversary completely knows the sharing $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$, or the first t honest parties' shares are distributed randomly to the adversary. If given in addition to the adversary's known shares: random a_α^* for $\alpha \in [t+1]$, and also for those $\alpha \in [t+1]$ satisfying the latter case, random $x_\alpha^1, \ldots, x_\alpha^t$, then values distributed identically to the adversary to honest parties' shares of $[d_k']_{2t}^{\mathcal{C}_{i+1}}$ received by P_k of $\mathcal{C}_{i+2}$, for $k \in \mathcal{T}_{\mathcal{C}_{i+2}}$, in π_{mult} can be determined. Moreover, values distributed identically to the adversary to d_k' sent from honest party P_k of $\mathcal{C}_{i+2}$, for $k \in \mathcal{H}_{\mathcal{C}_{i+2}}$, in π_{mult} can be determined. The same holds for the corresponding values and sharings of e_k' for $k \in [n]$.*

Proof. Let $\mathcal{H}_{\mathcal{C}_{i+1}} = \{h_1, \ldots, h_{2t+1}\}$. From the definition of $\mathcal{F}_{\mathsf{rand}}$, we can conclude that the shares $(a_1^{h_{2t+1}}, \ldots, a_{t+1}^{h_{2t+1}}) = (a_1^*, \ldots, a_{t+1}^*)$ of the last honest party $P_{h_{2t+1}}$ of sharings $([a_1]_t^{\mathcal{C}_{i+1}}, \ldots, [a_{t+1}]_t^{\mathcal{C}_{i+1}})$, respectively, received in Step 1 of π_{mult} are distributed uniformly at random to the adversary. Now, let us analyze the first case that for some x_α, $\alpha \in [t+1]$, sharing $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ is completely known to the adversary. Then, given random $a_\alpha^{h_{2t+1}}$, the t corrupted parties' shares of $[a_\alpha]_t^{\mathcal{C}_{i+1}}$, and the error sharing $[e_\alpha]_{t'}^{\mathcal{C}_{i+1}}$; the first $2t+1$ honest parties' shares $a_\alpha^{h_1} + e_\alpha^{h_1}, \ldots, a_\alpha^{h_{2t+1}} + e_\alpha^{h_{2t+1}}$ can be determined. Therefore, given random $a_\alpha^{h_{2t+1}}$, and the shares $x_\alpha^{h_1}, \ldots, x_\alpha^{h_{2t+1}}$ that the adversary knows, $x_\alpha^{h_1} + a_\alpha^{h_1} + e_\alpha^{h_1}, \ldots, x_\alpha^{h_{2t+1}} + a_\alpha^{h_{2t+1}} + e_\alpha^{h_{2t+1}}$ can all be determined.

The other case is that for x_α, the adversary only knows the corrupted parties' shares of sharing $[x_\alpha]_{2t}^{\mathcal{C}_{i+1}}$. This means that $x_\alpha^{h_1}, \ldots, x_\alpha^{h_t}$ are distributed randomly to the adversary, in addition to a_α^{2t+1}. Therefore, all of $x_\alpha^{h_1} + a_\alpha^{h_1}, \ldots, x_\alpha^{h_t} + a_\alpha^{h_t}$ and $x_\alpha^{h_{2t+1}} + a_\alpha^{h_{2t+1}}$ are distributed randomly to the adversary. Given these $t+1$ random shares and the t corrupted parties' shares of $[x_\alpha + a_\alpha]_t^{\mathcal{C}_{i+1}}$, the remaining shares $x_\alpha^{h_{t+1}} + a_\alpha^{h_{t+1}}, \ldots, x_\alpha^{h_{2t}} + a_\alpha^{h_{2t}}$ can be determined.

Putting the observations of the above two cases together, for all $\alpha \in [t+1]$, given some values that are uniformly random to the adversary, the shares of $[x_\alpha + a_\alpha]_{2t}^{\mathcal{C}_{i+1}}$ of the honest parties can be determined. Therefore, so too can the honest parties' shares of

$$([d_1']_{2t}^{\mathcal{C}_{i+1}}, \ldots, [d_n']_{2t}^{\mathcal{C}_{i+1}})^\intercal \leftarrow M \cdot ([x_1 + a_1]_{2t}^{\mathcal{C}_{i+1}}, \ldots, [x_{t+1} + a_{t+1}]_{2t}^{\mathcal{C}_{i+1}})^\intercal.$$

It then also trivially follows that the values $d_1', \ldots, d_n'$ can be determined.

It is clear that the same argument can be applied for the values and sharings of $e_1', \ldots, e_n'$. ◻

Lemma 15 (Lemma 7, re-stated)**.** *If no parties of committee $\mathcal{C}_{i+3}$ receive* abort *in π_{mult}, then they always correctly determine the values $(x_1 + a_1, \ldots, x_{t+1} + a_{t+1})$ and $(y_1 + b_1, \ldots, y_{t+1} + b_{t+1})$.*

Proof. This follows directly from the error-detection of reconstructions and linearity of degree-$2t$ Shamir secret sharings, as well as the the error-correcting properties of super-invertible matrix M. In particular, if none of the parties abort, then by the aforementioned properties of Shamir secret sharings, the honest parties of committee $\mathcal{C}_{i+2}$ reconstruct the k-th symbol d_k' of the codeword $(d_1', \ldots, d_n')$, for $k \in \mathcal{H}_{\mathcal{C}_{i+2}}$, of message $(d_1, \ldots, d_{t+1}) = (x_1 + a_1, \ldots, x_{t+1} + a_{t+1})$, and forward them to the parties of committee $\mathcal{C}_{i+3}$. The parties of committee $\mathcal{C}_{i+3}$ then receive the whole codeword, of which up to t symbols (received from the corrupted parties) are erroneous. Therefore, by the error-correcting properties of super-invertible matrix M, the Berlekamp-Welch algorithm will recover the correct original message $(d_1, \ldots, d_{t+1})$ for the parties. The same clearly applies for $(e_1, \ldots, e_{t+1})$. ◻

Theorem 3 (Theorem 2, restated)**.** *Protocol Π_{main} UC-securely computes $\mathcal{F}_{\mathsf{DABB}}$ in the presence of an R-adaptive adversary $\mathcal{A}$ in the ($\mathcal{F}_{\mathsf{input}}$, $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$, $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$, $\mathcal{F}_{\mathsf{rand}}$, $\mathcal{F}_{\mathsf{output}}$)-hybrid model.*

Proof. First, we define simulator $\mathcal{S}$:

1. For each input x, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{input}}$ – in case the client is corrupted, $\mathcal{S}$ gets from the adversary the sharing $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$, reconstructs x and sends it to $\mathcal{F}_{\mathsf{DABB}}$; otherwise, $\mathcal{S}$ gets from the adversary the t corrupted clients' shares of $[x]_{2t}^{\mathcal{C}_{\mathsf{clnt}}}$.

2. During the execution phase, for each wire value x, the corresponding sharing $[x]_{2t}^{\mathcal{C}_i}$ might have some additive error ε_x (for the first layer, each $\varepsilon_x = 0$). $\mathcal{S}$ will maintain the invariant of computing the corrupted parties' shares of $[x + \varepsilon_x]_{2t}^{\mathcal{C}_i}$ for every wire value x – as a base case, it already has these for all circuit inputs.

3. For identity gates:

   (a) $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ – $\mathcal{S}$ sends to the adversary the corrupted parties' shares of $[x_1 + \varepsilon_{x_1}]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1} + \varepsilon_{x_{t+1}}]_{2t}^{\mathcal{C}_i}$ that it has already computed, then receives back a set of shares $\{\boldsymbol{x}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$ and error vector $\boldsymbol{\Delta}$.

   (b) Then $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ – $\mathcal{S}$ sends to the adversary the just received shares $\{\boldsymbol{x}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$ and receives back set of shares $\{(x_1^k, \ldots, x_{t+1}^k)\}_{k\in\mathcal{T}_{\mathcal{C}_{i+2}}}$ (thus it has all of the corrupted parties' shares of the gates' output wires) and errors $(\delta_1, \ldots, \delta_{t+1})$.

   (c) $\mathcal{S}$ next computes the new error $\varepsilon'_{x_\alpha} \leftarrow \Delta_\alpha + \delta_\alpha$ on the output wire of the gate, for $\alpha \in [t+1]$.

   (d) Finally, if $\boldsymbol{\Delta} \neq \boldsymbol{0}$ then $\mathcal{S}$ aborts for committee $\mathcal{C}_{i+3}$ and if $(\delta_1, \ldots, \delta_{t+1}) \neq (0, \ldots, 0)$, then $\mathcal{S}$ aborts for committee $\mathcal{C}_{i+4}$.

4. Addition gates proceed similarly.

5. For multiplication gates:

   (a) $\mathcal{S}$ first emulates $\mathcal{F}_{\mathsf{rand}}$ – it receives from the adversary the set of shares $\{(a_1^k, b_1^k, \ldots, a_{t+1}^k, b_{t+1}^k)\}_{k\in\mathcal{T}_{\mathcal{C}_i}}$ and sharings $([\eta_1]_{t'}^{\mathcal{C}_i}, \ldots, [\eta_{t+1}]_{t'}^{\mathcal{C}_i})$.

   (b) Then, $\mathcal{S}$ samples random $a_1^*, \ldots, a_{t+1}^*$ and for each $\alpha \in [t+1]$ s.t. x_α was not input by a corrupted party, $x_\alpha^1, \ldots, x_\alpha^t$, and similarly random $b_1^*, \ldots, b_{t+1}^*$ and for each $\alpha \in [t+1]$ s.t. y_α was not input by a corrupted party, $y_\alpha^1, \ldots, y_\alpha^t$.

   (c) Based on these and the corrupted parties' shares of the inputs $[x_1+\varepsilon_{x_1}]_{2t}^{\mathcal{C}_i}, \ldots, [x_{t+1}+\varepsilon_{x_{t+1}}]_{2t}^{\mathcal{C}_i}$ and $[y_1 + \varepsilon_{y_1}]_{2t}^{\mathcal{C}_i}, \ldots, [y_{t+1} + \varepsilon_{y_{t+1}}]_{2t}^{\mathcal{C}_i}$ that $\mathcal{S}$ has already computed (or all such shares for those x_α, y_α input by corrupted parties), $\mathcal{S}$ computes the honest parties' shares of $[d'_k]_{2t}^{\mathcal{C}_i}$ and $[e'_k]_{2t}^{\mathcal{C}_i}$ for $k \in \mathcal{T}_{\mathcal{C}_{i+1}}$, as well as values d'_k, e'_k for $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, according to the proof of Lemma 6.

   (d) $\mathcal{S}$ then sends to the adversary the honest parties' shares of $[d'_k]_{2t}^{\mathcal{C}_i}$ and $[e'_k]_{2t}^{\mathcal{C}_i}$ just computed.

   (e) Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ three independent times – $\mathcal{S}$ sends to the adversary the above received corrupted parties' shares $\{(a_1^k, \ldots, a_{t+1}^k)\}_{\mathcal{T}_{\mathcal{C}_i}}$ and $\{(b_1^k, \ldots, b_{t+1}^k)\}_{\mathcal{T}_{\mathcal{C}_i}}$, as well as $\{(a_1^k \cdot b_1^k, \ldots, a_{t+1}^k \cdot b_{t+1}^k)\}_{\mathcal{T}_{\mathcal{C}_i}}$.

   (f) It receives back a set of shares $\{\boldsymbol{a}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$, $\{\boldsymbol{b}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$, and $\{\boldsymbol{c}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$, as well as error vectors $\boldsymbol{\Delta_a}$, $\boldsymbol{\Delta_b}$, and $\boldsymbol{\Delta_c}$.

   (g) $\mathcal{S}$ then sends to the adversary the values d'_k, e'_k for $k \in \mathcal{H}_{\mathcal{C}_{i+1}}$, computed above.

(h) Next, $\mathcal{S}$ emulates $\mathcal{F}_{\text{robust-standard-reshare}}$ three independent times – $\mathcal{S}$ sends to the adversary the just received shares $\{\boldsymbol{a}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$, $\{\boldsymbol{b}^k\}_{k\in i+1}$, and $\{\boldsymbol{c}^k\}_{k\in\mathcal{T}_{\mathcal{C}_{i+1}}}$.

(i) It then receives back sets of shares $\{(a_1^k,\ldots,a_{t+1}^k)\}_{k\in\mathcal{T}_{\mathcal{C}_{i+2}}}$, $\{(b_1^k,\ldots,b_{t+1}^k)\}_{k\in\mathcal{T}_{\mathcal{C}_{i+2}}}$, and $\{(c_1^k,\ldots,c_{t+1}^k)\}_{k\in\mathcal{T}_{\mathcal{C}_{i+2}}}$ as well as errors $(\delta_{a_1},\ldots,\delta_{a_{t+1}})$, $(\delta_{b_1},\ldots,\delta_{b_{t+1}})$, and $(\delta_{c_1},\ldots,\delta_{c_{t+1}})$.

(j) Then $\mathcal{S}$ computes $\{d_\alpha\cdot e_\alpha - d_\alpha\cdot b_\alpha^k - e_\alpha\cdot a_\alpha^k + c_\alpha^k\}_{\alpha\in[t+1],k\in\mathcal{T}_{\mathcal{C}_{i+2}}}$ as the corrupted parties' shares of the gates' output wires $(d_1, e_1,\ldots,d_{t+1},e_{t+1}$ can be directly computed from $d_1', e_1',\ldots,d_{t+1}',e_{t+1}')$.

(k) $\mathcal{S}$ next computes the new error $\varepsilon_{z_\alpha}\leftarrow d_\alpha\cdot(\Delta_{b_\alpha}+\delta_{b_\alpha})-e_\alpha\cdot(\Delta_{a_\alpha}+\delta_{a_\alpha})+\Delta_{c_\alpha}+\delta_{c_\alpha}$ on the output wire of the gate, for $\alpha\in[t+1]$.

(l) Finally, if any of $\boldsymbol{\Delta_a},\boldsymbol{\Delta_v},\boldsymbol{\Delta_c}$ are not $\mathbf{0}$ then $\mathcal{S}$ aborts for committee $\mathcal{C}_{i+3}$ and if any of $(\delta_{a_1},\ldots,\delta_{a_{t+1}}),(\delta_{b_1},\ldots,\delta_{b_{t+1}}),(\delta_{c_1},\ldots,\delta_{c_{t+1}})$ are not $(0,\ldots,0)$, then $\mathcal{S}$ aborts for committee $\mathcal{C}_{i+4}$.

6. Finally, for each output z, $\mathcal{S}$ emulates $\mathcal{F}_{\text{output}}$:

(a) If the client is corrupted, $\mathcal{S}$ receives z from $\mathcal{F}_{\text{DABB}}$, and using the above computed error ε_z, the corrupted parties' shares of $[z+\varepsilon_z]_{2t}^{\mathcal{C}_\ell}$ that it has computed, and t more random values for the first t honest parties shares, reconstructs all of $[z+\varepsilon_z]_{2t}^{\mathcal{C}_\ell}$ then sends it to the adversary.

(b) Otherwise, if the client is honest $\mathcal{S}$ sends the corrupted parties' shares of outputs $[z+\varepsilon_z]_{2t}^{\mathcal{C}_\ell}$ that it has computed to the adversary.

Now we argue that the real world and ideal world are distributed identically to the adversary. It is clear that $\mathcal{S}$ correctly reconstructs the corrupted parties' inputs x in the ideal world. Therefore all outputs received from $\mathcal{F}_{\text{DABB}}$ by $\mathcal{S}$ will be consistent with the adversary's inputs. Additionally, it is clear that $\mathcal{S}$ has the corrupted parties' shares of the input sharings corresponding to the real world. We will show that $\mathcal{S}$ maintains the invariant of computing the corrupted parties' shares for every wire. Furthermore, it is clear that for all inputs that do not come from corrupted clients, the first t honest parties' shares are distributed randomly to the real world adversary. We will furthermore show that this invariant is maintained for all wires that are not corrupted clients' inputs.

For identity and addition gates, assuming the invariant, $\mathcal{S}$ clearly sends to the adversary the same corrupted parties' shares that it would receive in the real world. $\mathcal{S}$ additionally aborts when the parties in the real world would (according to Lemma 8). It is also clear that $\mathcal{S}$ maintains the invariant of computing the corrupted parties' shares for every wire. Furthermore, from the definition of $\mathcal{F}_{\text{robust-standard-reshare}}$ it is clear that the invariant that the first t honest parties' shares are distributed randomly to the real world adversary for every wire is maintained.

For multiplication gates, because of the invariant that the first t honest parties' shares are distributed randomly to the real world adversary for every wire, we know that the assumption on which Lemma 6 is based is true. Therefore using the lemma, we know that $\mathcal{S}$ computes and sends to the adversary honest parties' shares of $[d_k']_{2t}^{\mathcal{C}_i}$ and $[e_k']_{2t}^{\mathcal{C}_i}$ for $k\in\mathcal{T}_{\mathcal{C}_{i+1}}$, as well as values d_k', e_k' for $k\in\mathcal{H}_{\mathcal{C}_{i+1}}$, that are distributed identically to those in the real world. It is also clear that $\mathcal{S}$ sends to the adversary the same corrupted parties' shares when emulating $\mathcal{F}_{\text{robust-packed-reshare}}$ and $\mathcal{F}_{\text{robust-standard-reshare}}$ that it would receive in the real world. $\mathcal{S}$ additionally aborts when the parties in the real world would (according to Lemma 8). Finally, $\mathcal{S}$ also maintains the invariant of computing the corrupted parties' shares for every wire, as it can compute $d_1,e_1,\ldots,d_{t+1},e_{t+1}$ directly and receives from the adversary the corrupted parties' shares of the reshared multiplication triples. Furthermore, from the definition of $\mathcal{F}_{\text{robust-standard-reshare}}$, it is clear that the first t honest parties' shares of the multiplication triple are distributed randomly to the real world adversary, and thus the invariant that the first t honest parties' shares are distributed randomly to the real world adversary for every wire is maintained.

Thus the execution phase is simulated perfectly and all that remains to show is that the output phase is simulated perfectly. First, we recall that from Lemma 7 that during every run of π_{mult} of the execution phase, the parties of committee $\mathcal{C}_{i+2}$ always correctly determine the values of $(x_1 + \varepsilon_{x_1} + a_1, y_1 + \varepsilon_{y_1} + b_1, \ldots, x_{t+1} + \varepsilon_{x_{t+1}} + a_{t+1}, y_{t+1} + \varepsilon_{y_{t+1}} + b_{t+1})$. Now, for any circuit layer starting with committee $\mathcal{C}_i$, observe that if the adversary injects any error via $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ or $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$, then in the real world, at the latest, the honest parties of committee $\mathcal{C}_{i+4}$ will abort. Since for the next layer, the shares of the output wires of the gates are not computed until $\mathcal{C}_{i+4}$, that means that if the parties do not abort at the end of computing a layer, then the errors on the output wires from $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ and $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ can only come from the computation of that layer, and not any previous layers. Similarly, if the adversary injects any error via $\mathcal{F}_{\mathsf{rand}}$, then in the real world, at the latest, the honest parties of committee $\mathcal{C}_{i+2}$ will abort, which means that the errors on the output wires for any layer can only come from $\mathcal{F}_{\mathsf{robust\text{-}packed\text{-}reshare}}$ and $\mathcal{F}_{\mathsf{robust\text{-}standard\text{-}reshare}}$ for that layer. Therefore, $\mathcal{S}$ properly computes the errors on the wires of the output gates. Based on these, and since the invariant that the first t honest parties' shares are distributed randomly to the real world adversary for every wire holds, it is clear that $\mathcal{S}$ simulates the output phase perfectly. Indeed, when emulating $\mathcal{F}_{\mathsf{output}}$, $\mathcal{S}$ uses the same underlying secret $z + \varepsilon_z$ as in the real world, the corrupted parties' t shares and t random values for the first t honest parties shares to reconstruct the sharing $[z + \varepsilon_z]_{2t}^{\mathcal{C}_\ell}$ to the adversary. This concludes the proof. $\square$