Maximally-Fluid MPC with Guaranteed Output Delivery[*]

Giovanni Deligios
ETH Zurich
gdeligios@ethz.ch

Aarushi Goel
NTT Research
aarushi.goel@ntt-research.com

Chen-Da Liu-Zhang
NTT Research
chen-da.liuzhang@ntt-research.com

Abstract

To overcome the limitations of traditional secure multi-party computation (MPC) protocols that consider a *static* set of participants, in a recent work, Choudhuri et al. [CRYPTO 2021] introduced a new model called Fluid MPC, which supports *dynamic* participants. Protocols in this model allow parties to join and leave the computation as they wish. Unfortunately, known fluid MPC protocols (even with strong honest-majority), either only achieve security with abort, or require strong computational and trusted setup assumptions.

In this work, we also consider the “hardest” setting — called the *maximally-fluid* model — where each party can leave the computation after participating in a single round. We study the problem of designing maximally-fluid MPC protocols that achieve security with guaranteed output delivery, and obtain the following main results:

- We design a *perfectly secure* maximally-fluid MPC protocol, that achieves guaranteed output delivery against unbounded adversaries who are allowed to corrupt less than a third of the parties in every round/committee.
- For the case where the adversary is allowed to corrupt up to half of the parties in each committee, we present a new computationally secure maximally-fluid MPC protocol with guaranteed output delivery. Unlike prior works that require correlated setup and NIZKs, our construction only uses a common random string setup and is based on linearly-homomorphic equivocal commitments.

[*] A merged version of this paper with [DKI$^+$23] appears in CRYPTO 2023.

Contents

1 Introduction

Secure multi-party computation (MPC) [Yao86, GMW87, BGW88, CCD88, RB89] is a fundamental notion in cryptography. It allows a set of mutually distrusting parties to jointly compute a function over their private data, in a manner which ensures that nothing about their private data beyond the output of the function is leaked. Traditionally, MPC protocols were designed assuming a *static* set of participants, that are required to stay online throughout the entire duration of the protocol execution. As such, this limits their use in modern large-scale applications that are long-lived, such as federated learning algorithms.

To address this, a recent line of works consider more flexible models of computation called *Fluid MPC* [CGG$^+$21, GHK$^+$21] (see additional related work in Section 1.2), which support dynamic participation of parties. Protocols in this model allow parties to join and leave as they wish, without disrupting the execution. As a result, parties can volunteer to participate in only a few rounds, thereby eliminating the need for them to commit to their resources for very long durations. In the extreme case, called the *maximally-fluid*[1] setting, parties who wish to participate can sign up for as few as a single round of communication in the protocol. Needless to say, this gives parties the most amount of flexibility.

Unfortunately, known protocols with maximal-fluidity (even in the strong honest-majority setting), either require strong computational assumptions and trusted correlated setup [GHK$^+$21, BGG$^+$20] or only achieve so-called *security with abort* [CGG$^+$21, RS22]. Security with abort essentially gives the adversary the power to launch denial-of-service attacks — which is far from ideal in any MPC protocol — but in the fluid setting, this could be even more frustrating. Firstly, it limits the use of the fluid model of computation in *MPC-as-a-service* type applications [BHKL18]. Moreover, having large-scale distributed computations end in failure, is also a waste of time and resources of all the parties who volunteer to participate, and can potentially deter them from volunteering in the future.

To remedy this, in this work, we investigate the possibility of designing *more robust* maximally-fluid MPC protocols with information-theoretic (IT) security. We ask:

Do there exist information-theoretic maximally-fluid MPC protocols, that achieve security with guaranteed output delivery?

Before stating our results, it is instructive to discuss the communication model in fluid MPC in more detail. Similar to the work of Choudhuri et al. [CGG$^+$21], we consider fluid MPC protocols in the client-server model, where a group of "static" clients holding private inputs, delegate the task of computing a joint function on their inputs to a group of "dynamically evolving" volunteer servers. The servers participating in a given round constitute the *committee* for that round and in the maximally-fluid setting, these servers can exit the computation after receiving and sending messages in one round. In the maximally-fluid setting, the committee for any round r, is only revealed in round $r - 1$, when the committee for round $r - 1$ is ready to send messages to this committee. We assume an honest majority of servers in each committee.[2] Finally, as in all prior works [CGG$^+$21, GHK$^+$21, BGG$^+$20, RS22], we assume that the servers have access to a broadcast channel (even in the plain model) and servers in two consecutive committees can also communicate with each other via private point-to-point channels.

[1] For readers familiar with the framework of YOSO MPC [GHK$^+$21], this setting is equivalent to YOSO MPC with static triggering and future horizon $= 1$.

[2] We allow corruption of any number of clients.

1.1 Our Results

We answer the above question in the affirmative. We first show that in the information-theoretic (IT) setting without assuming trusted setup, one can simultaneously achieve maximal-fluidity and guaranteed output delivery, if the adversary is allowed to corrupt up to $1/3^{rd}$ of the servers in every committee. Next, we show that if we allow up to $1/2$-fraction of corruptions in each committee, it is possible to design such a protocol with computational security and assuming a common random string (CRS) setup. We now discuss our results in more detail.

Result 1: Maximally-Fluid MPC with Perfect Security. We present the first[3] *perfectly* secure maximally-fluid MPC that achieves guaranteed output delivery in the plain model.

Theorem (Informal) 1. *There exists a maximally-fluid MPC protocol in the plain model, that achieves perfect security with guaranteed output delivery against an unbounded adversary who is allowed to corrupt up to* $1/3^{rd}$ *of the servers in every committee.*

Previously, in the information-theoretic setting, Choudhuri et al. [CGG$^+$21] presented a satistically secure, maximally-fluid MPC that only achieves security with abort. Gentry et al. [GHK$^+$21] gave a construction of statistically secure YOSO MPC with guaranteed output delivery (allowing up to $1/2$-fraction of random corruptions in each committee). However, it is unclear how to adapt their protocol to the fluid-setting for two reasons — (1) first, their construction is only secure if the adversary gets to corrupt a "random" subset of parties in each committee; (2) moreover, their protocol requires committees to send messages not just to the immediate next committee, but also to other committees that are expected to participate (and appear online) a lot later in the future. For this they assume the existence of "secure channels to the future". As discussed in [GHK$^+$21], it is unclear how one could instantiate such channels information-theoretically with up to $1/2$-fraction of corruptions, especially when the identities of parties in future committees is unknown at the time of sending these messages.

Additionally, since both these prior works only achieve statistical security, another attractive feature of our protocol is that it is the *first fluid MPC that has perfect security*, for any corruption-threshold. In our protocol, we assume that each committee comprises of the same number of servers. This is similar to the guaranteed output delivery protocols in [GHK$^+$21], but unlike the security with abort protocol in [CGG$^+$21] that allows for variable number of parties in each committee. We leave designing a guaranteed output delivery protocol with a similar property as an interesting open problem.

Finally, we remark that given the close connections between Fluid and YOSO MPC models, our protocol also yields the first information-theoretic YOSO MPC (achieving guaranteed output delivery) with future horizon $= 1$, i.e., where each committee is only required to communicate with the immediate next committee. Previously, Gentry et al. [GHK$^+$21] presented a statistically secure YOSO MPC assuming secure channels to the future. However, as discussed above, it is unclear how to instantiate such channels information-theoretically. A YOSO MPC with future horizon $= 1$, eliminates the need for such channels and hence is more desirable.

Result 2: Maximally-Fluid MPC with Computational Security. Next, we ask whether it is possible to design a maximally-fluid MPC using common random string setup, while achieving guaranteed output delivery and increased resilience by relying on computational assumptions.

[3] In a concurrent work, [DKI$^+$23] also gave a construction of a perfectly secure maximally-fluid MPC with guaranteed output delivery for $t < n/3$.

Theorem (Informal) 2. *Assuming linearly-homomorphic equivocal commitments and a common random string setup, there exists a maximally-fluid MPC that achieves guaranteed output delivery and can tolerate up to* $1/2$*-fraction of corruptions in each committee.*

Our construction relies on linearly-homomorphic equivocal commitments, which can be instantiated using a variety of assumptions such as discrete-log (e.g., Pedersen commitments [Ped92]) or lattice based assumptions (e.g., LWE or SIS [GVW15]). As discussed earlier, Gentry et al. [GHK$^+$21] show a computationally-secure, guaranteed output delivery YOSO MPC with future horizon $= 1$ and a similar corruption threshold. Their protocol can also be adapted as a maximally-fluid MPC. However, their construction was a simple adaptation of the CDN protocol [CDN01] and makes use of NIZKs and linearly homomorphic threshold encryption, which requires trusted correlated setup. We present a protocol using a weaker cryptographic assumption, while only relying on common random string setup. We leave the problem of designing a similar protocol without setup and based on the minimal assumption of one-way functions for future work.

Overall, our results help resolve some of the questions pertaining to guaranteed output delivery that were left open in [CGG$^+$21] and [GHK$^+$21].[4] Choudhuri et al. [CGG$^+$21] had envisioned a volunteer-enabled, weighted, privacy-preserving distributed computing system as one of the motivations behind the fluid model. Having a robust protocol for such a system has many advantages. It also helps dissuade notorious entities, who might otherwise simply participate to launch a denial of service attack, from volunteering in the system.

1.2 Related Work

Recently, in a concurrent work, [DKI$^+$23] also proposed a construction of a perfectly secure maximally-fluid MPC (referred to as layered MPC in their work) with guaranteed output delivery for $t < n/3$. Similar to ours, their protocol also follows the BGW paradigm [BGW88] and demonstrates how to adapt it to the maximally-fluid/layered model.

We discuss further relevant related work. As discussed in the introduction, the fluid model of computation with dynamic participants was proposed in [CGG$^+$21]. They gave a construction of an information-theoretic maximally-fluid MPC in the honest majority setting that achieves security with abort. Recently, [RS22] designed a fluid MPC in the dishonest majority setting.

The most closely related model to fluid MPC was suggested in [GHK$^+$21], and is called YOSO MPC. This model also considers the notion of dynamic participants. However, the motivations in the two models are quite different. Unlike fluid MPC, where the main goal is to try and minimize the commitment required from each participant, the main motivation behind YOSO MPC is a stronger adversarial model. Some follow-up works in this line of research include [KRY22] which focuses on improving the round complexity of YOSO MPC, and [CDGK22, CDK$^+$23] that focus on designing efficient ways for encrypting to/sharing messages with future committees. The work [NRO22] deals with YOSO coin-flipping in a setting where a bounded number of committees can have dishonest majority.

Some recent works such as [AHKP22, DEP21] consider other models of dynamic participation. Prior to Fluid and YOSO MPC, the idea of dynamic participants in secret sharing schemes was explored in [BGG$^+$20, GKM$^+$20]. The notion of player replaceability (where the set of parties get replaced in every round) has previously been studied in the context of consensus primitives [Mic17, CM19, PS17, BKLZL20]. The well-studied notion of proactive security [OY91, BELO14] also considers a closely related model, where the

[4] Our results also present concrete resolutions to some of the conjectures presented by Nielsen in a recent talk `https://www.youtube.com/watch?v=PfC1lf1nuw8`.

parties remain static, but the adversary is allowed to corrupt different sets of parties at different times in the protocol.

2 Technical Overview

In this section, we give discuss the main technical ideas underlying our results. We start by recalling the fluid MPC model introduced in [CGG$^+$21].

Fluid Model. A fluid MPC is a protocol with m-clients $\mathcal{C}$ and N-parties $\mathcal{S}$ (the servers). The clients have input and receive output, and computation is carried out by the parties in a sequence of E epochs, where in each epoch ℓ, a committee $\mathcal{S}^\ell = \{P_1^\ell, \ldots, P_n^\ell\}$ of n parties (with $\mathcal{S}^\ell \subseteq \mathcal{S}$) participate as senders in a single round of communication consisting of broadcast channels whose content is accessible towards all future committees, and secure channels towards the parties in the next committee.[5] In each committee, the adversary can choose to corrupt up to any t out of n parties. We give a more detailed description in Section 3.1.

The rest of this section is organised as follows. In Section 2.1 we discuss the techniques used in our perfectly secure protocol. Finally, in Section 2.2, we give an overview of our computationally secure protocol with increased resilience threshold.

2.1 Maximally-Fluid MPC with Perfect Security

Our first main technical contribution is a maximally-fluid MPC protocol with perfect security and guaranteed output delivery, secure up to $t < n/3$ corruptions in each committee.

2.1.1 Our Approach and General Challenges

Our protocol follows the well-known BGW paradigm [BGW88], where parties evaluate a function by emulating the computation of an arithmetic circuit over a finite field. In this paradigm, the clients first distribute their inputs using a secret sharing scheme towards the parties (or servers) who then emulate the computation in a gate-by-gate manner — computing shares of the output wire of every gate using shares of the input wires of that gate. In the end, they reconstruct the values of the output wires of the circuit towards the clients.

Previously, [CGG$^+$21] showed how to adapt the *semi-honest* version of BGW to the maximally-fluid setting, however they were only able to show how to upgrade the security of this protocol to *security with abort* in the presence of an active adversary. Adapting the BGW-paradigm to achieve a robust protocol with maximal-fluidity imposes several challenges. First, note that popular (and efficient) techniques based on player elimination or dispute control [HMP00, BTH06] that enable leveraging non-robust protocols into robust protocols are not applicable in our setting. Indeed, they are based on the idea of detecting and kicking out cheating parties so that they can no longer harm the computation, and rely on the fact that the number of remaining parties decreases (while keeping the required corruption ratio). In our case, detecting a cheating party in a committee $\mathcal{S}^c$ does not help, because each party only participates in a single round of communication, and future committees can be composed of entirely different parties. As such, it seems unlikely that leveraging such techniques could help lift prior non-robust fluid protocols [CGG$^+$21] to achieve active security with guaranteed output delivery.

[5] In the original model, each epoch consists of k rounds of communication with the same committee – denoted fluidity k. In this work we only consider protocols with maximal fluidity, or fluidity 1, where each epoch contains a single round of communication.

As a result, we turn our attention to designing a robust maximally-fluid protocol in the BGW-paradigm from scratch. The robust variant of BGW [BGW88, AL17] crucially makes use of a verifiable secret sharing (VSS) scheme (cf. [CGMA85, GMW87]), instead of regular secret sharing. Therefore, our first step is to design a VSS protocol with maximal fluidity. Unfortunately, it is well known that sharing a value with VSS in 1-round is impossible [PCRR09] (in fact, we need at least 3 rounds of communication to achieve perfect security [GIKR01]). Moreover, current VSS protocols are not stateless and crucially require parties to maintain their private state across different rounds. However, in our setting, since at each round r only one committee $\mathcal{S}^r$ speaks (and parties in this committee may no longer participate), we need a mechanism that allows different committees participating in different rounds to maintain state on their behalf. On the other hand, maintaining the same state across different committees clearly violates secrecy, since the adversary can corrupt *any* set of t parties in each committee, potentially learning the state of too many parties.

2.1.2 Perfectly Secure Verifiable Secret Sharing

Before describing our ideas for resolving the above conundrum, let us first explain what we mean by a (maximally-fluid) VSS protocol. This is a protocol that consists of two phases. The first phase, $\mathsf{VSS.ShareFuture}$, allows any dealer $P_d^c \in \mathcal{S}^c$ to secret share a value s to some future committee $\mathcal{S}^{c'}$, provided that $c' \geq c+1$, in a verifiable manner. In the second phase called $\mathsf{VSS.RecFuture}$, committee $\mathcal{S}^{c'}$ should be able to reconstruct a value s' towards another committee $\mathcal{S}^{c''}$ for some $c'' > c'$.

We want this protocol to achieve the following security guarantees — (1) *correctness*, ensures that if the dealer P_d^c is honest, then the output of $\mathsf{VSS.RecFuture}$ is $s' = s$. (2) moreover, before $\mathsf{VSS.RecFuture}$ starts, the scheme must satisfy *secrecy*, which means that the adversary learns no information about an honest dealer's secret s. (3) Finally, the scheme should satisfy a *commitment* property, meaning that if the phase $\mathsf{VSS.ShareFuture}$ succeeds, then there exists a unique value s' that will be reconstructed after $\mathsf{VSS.RecFuture}$.

We now describe our ideas for designing a maximally-fluid VSS, which is secure as long as up to $t < n/3$ parties are corrupted in committees from c to c''.

Background. Our starting point is the (non-fluid) protocol due to Gennaro, Ishai, Kushilevitz, and Rabin [GIKR01], which follows the traditional approach for verifiable secret sharing based on bivariate polynomials over a finite field [BGW88]. The sharing phase of the protocol proceeds as follows:

- *Round 1.* First, the dealer P_d chooses a bivariate polynomial $F(x, y)$ of degree at most t in each variable, such that $F(0, 0) = s$ and sends to each party P_i the i-th projections $f_i(x) = F(x, i)$ and $g_i(y) = F(i, y)$. Moreover, each party P_i sends to each P_j a uniform random pad r_{ij}.
- *Round 2.* Each party P_i broadcasts the blinded values $a_{ij} = f_i(j) + r_{ij}$ and $b_{ij} = g_i(j) + r_{ji}$[6].
- *Round 3.* For each conflicting pair ($a_{ij} \neq b_{ji}$), the involved parties broadcast their local points: P_i broadcasts $f_i(j)$, P_j broadcasts $g_j(i)$ and P_d broadcasts $F(j, i)$. A party is then considered *unhappy* if his value does not match the dealer's value. If there are more than t unhappy parties, the dealer is disqualified.

[6] Here r_{ji} denotes the pad that P_i receives from P_j.

- *Round 4.* For each unhappy party, the dealer broadcasts the polynomial $f_i(x)$, and each party P_j who is happy broadcasts $g_j(i)$.
- *Local Computation.* A party P_j who was happy at the beginning of round 4, becomes unhappy if the dealer broadcasted $f_i(x)$ in round 4 such that $g_j(i) \neq f_i(j)$. If the number of unhappy parties is now more than t, the dealer is disqualified.

To reconstruct, every party P_i that was happy at the beginning of round 4 of the sharing phase broadcasts $s_i = f_i(0)$. Then, for every unhappy party at the beginning of round 4, take $s_i = f_i(0)$ where $f_i(x)$ is the polynomial broadcasted by the dealer in round 4. Let $g(y)$ be the t-degree polynomial resulting from the Reed-Solomon error-correction interpolation procedure on the shares $s_1, \ldots, s_n$, and output $g(0)$.

If the dealer D is honest, all honest parties will be happy, since their broadcasted points will always match the dealer's broadcasted points. As a consequence, there are at most t unhappy parties and the dealer is not disqualified. The shares of the honest parties then allow for the correct reconstruction of the dealer's secret s. On the other hand, if the dealer P_d is corrupted and was not disqualified during the sharing phase, the shares from happy parties completely determine the value that can be reconstructed. This is because there are at least $n - t$ happy parties, out of which at least $n - 2t \geq t + 1$ are honest, so their polynomials $g_i(y)$ uniquely determine a degree-t bivariate polynomial $F'(x, y)$. Furthermore, it is not hard to see that the share s_i distributed during the reconstruction for each honest party P_i (including unhappy honest parties), will satisfy $s_i = F'(0, i)$.

Additional Challenges. As mentioned above, current perfectly-secure VSS protocols (such as the one described above) require parties to maintain their private state across different rounds (for example, the parties need to know their projected polynomials $f_i(x)$ and $g_i(y)$ throughout all rounds). Since this is not an option in the maximally-fluid setting, our VSS protocol must instead somehow keep the states of all parties in a distributed manner — shared across all future committees that might need these states. Moreover, to avoid privacy violations, the sharings of these states must be created independently for different committees. This however, introduces new challenges. Notice that in the above protocol, all operations are performed on whole states and transmitted values. We instead need to operate over shared states. Furthermore, these operations need to be performed locally and in a single round of communication, since the information will be lost after this round is executed. For the same reason, any consistency checks (such as checking that a polynomial has degree-t or checking crosspoints) must be performed in a public manner by all parties.

We note that these types of challenges were partially addressed in the paper by Gentry et al. [GHK$^+$21] for their statistically secure YOSO MPC protocol, for which they also need to adapt a VSS to their model. However, as discussed in the introduction, they work in an abstract model where they assume that parties can communicate with committees in the future, and that the adversary can only perform random corruptions chosen independently in each committee. Moreover, their protocol only achieves statistical security, while we seek to design a perfectly secure protocol. As such, it is unclear how their protocol (including their VSS protocol) can be directly adapted to our maximally-fluid setting, and hence we must design our perfectly secure VSS from scratch.

To address the challenges in our setting, we start by designing two subprotocols that will be useful in the design of our maximally-fluid VSS protocol.

1. *Secure Channels to the Future.* We first introduce a simple mechanism called $\mathsf{SendFuture}$ that allows a sender $P_s^c \in \mathcal{S}^c$ to send a message m to a recipient $P_r^{c'} \in \mathcal{S}^{c'}$, with $c' > c$ in a private manner, as long as the corruption threshold $t < n/3$ is maintained in each intermediate committee.

If $c' = c + 1$, P_s^c can directly send the message to $P_r^{c'}$. Otherwise, the protocol simply lets P_s^c distribute m towards the next committee using a t-out-of-n Shamir's secret sharing, i.e. sample a random degree-t polynomial f with constant coefficient m, and sends to each party P_i^{c+1} in the next committee a share $f(i)$. Each intermediate committee before $\mathcal{S}^{c'-1}$ obtains a share and re-shares all the received values towards the next committee. The last committee $\mathcal{S}^{c'-1}$ simply sends all the received shares to $P_r^{c'}$, who will run standard Reed-Solomon decoding to recover a message. It is easy to see that if P_s^c and $P_r^{c'}$ are honest, the decoded message is m, and the adversary learns no information about it.

Note that since each committee re-shares all the received values, the total incurred communication is $O(n^{c'-c-1})$ messages. Recall that an important feature of fluid MPC is that the work done by each committee must be independent of the depth of the circuit. Therefore, looking ahead, to avoid depth proportional overhead on any individual committee in our main protocol, we want to make sure to only use SendFuture with a constant number of intermediate committees. Fortunately, to instantiate our VSS protocol we will only need to send messages to committees that are a constant number of epochs ahead.

2. *Sharing to the Future.* The previous primitive also allows us to implement a sub-protocol ShareFuture that allows an honest dealer $P_d^c \in \mathcal{S}^c$ to secret-share a value s towards committee $\mathcal{S}^{c'}$, in such a way that the adversary does not obtain any information about the secret s, and the secret can be further reconstructed towards another committee $\mathcal{S}^{c''}$. This is achieved by simply letting P_d^c sample a uniform random degree-t polynomial f with $f(0) = s$, and use SendFuture to distribute each share $f(j)$ to party $P_j^{c'}$. Reconstruction simply lets each party $P_j^{c'}$ broadcast their share, and parties in $\mathcal{S}^{c''}$ will use Reed-Solomon decoding to recover the secret.

Achieving Maximal Fluidity. With the above sub-protocols, we are ready to describe our maximally-fluid perfect VSS protocol, which substantially differs from the protocol in [GIKR01] at several points. To begin with, the dealer samples a bivariate polynomial F with degree-t in both variables and $F(0,0) = s$. Instead of sending projected polynomials to each party in the next committee, and letting this single party (who may no longer participate after a round) check that the projection has degree-t, we change the distribution step so that the check becomes verifiable by *all* parties in future committees.

To achieve this, the dealer distributes each of the $(t+1)^2$ coefficients of F towards all future committees (in the VSS protocol) using independent instances of ShareFuture. Moreover, each party P_i^c samples a blinding degree-t polynomial $f^i(x)$ and distributes its coefficients using independent instances of ShareFuture to all future committees.[7] Parties in committee $\mathcal{S}^{c+1}$ can now broadcast blinded shares of $a_{uv} = F(u,v) + f^u(v)$ and $b_{uv} = F(u,v) + f^v(u)$.

The general idea is to use the broadcasted blinded horizontal and vertical polynomials as base polynomials defined by the points $\{a_{uv}\}_{v\in[1,n]}$ and $\{b_{vu}\}_{v\in[1,n]}$ to later reconstruct the secret.[8] If we make sure that these polynomials are of degree-t and each points a_{uv} and b_{uv} are consistent, i.e. blind the same point $F(u,v)$, then a future committee $\mathcal{S}^{c'}$ can later reconstruct the secret by reconstructing the blindings (which they have, since each party P_i^c in the first committee used ShareFuture to distribute a

[7] This step differs from the protocol in [GIKR01], which lets each party send a random pad for each point. This is because in their protocol each party P_i gets (without loss of generality) a projected polynomial from the dealer which is guaranteed to have degree-t. If it is not, P_i can assume a default one. However, in our case the projected polynomials (or rather, the coefficients) are shared and care needs to be taken to check that the sharing has degree-t.

[8] Crucially, note that we cannot use the re-randomized sharings of $F(x,y)$ that the dealer distributed in the first step, since we did not check that they are of degree t. Instead, we will use those to fix any inconsistencies that we find on the broadcasted blinded horizontal and vertical polynomials.

sharing of the blinding to parties in $\mathcal{S}^{c'}$). Of course, blindings distributed by corrupted parties in $\mathcal{S}^c$ can be arbitrary, but those coming from honest parties will lie on a degree-t polynomial. These $n-t$ blindings will be correctly reconstructed and will allow all honest parties to correctly unblind a unique degree-t bivariate polynomial F.

Checking the degree of the blinded polynomials. In order to check the degree of the blinded horizontal and vertical polynomials, parties in committee $\mathcal{S}^{c+2}$ simply use the shares broadcasted by committee $\mathcal{S}^{c+1}$ to reconstruct the points $\{a_{uv}\}_{v\in[1,n]}$ and check that they lie on a degree-t polynomial (and the same for the points $\{b_{vu}\}_{v\in[1,n]}$).[9] If any of these sets do not define a degree-t polynomial, we know that either party P_u^c or the dealer P_d^c is corrupted. In this case, we tentatively mark the index u as *unhappy*. Note that all parties from any future committees mark the same set of indices, since this check is done with broadcasted values.

Checking that the blinded polynomials are consistent. In order to verify that the points a_{uv} and b_{uv} are consistent, i.e. blinding the same point $F(u,v)$, the parties in committee $\mathcal{S}^{c+1}$ also broadcast shares of the difference of the blinding polynomials $c_{uv} = f^u(v) - f^v(u)$. Note that we cannot broadcast shares of $f^u(v)$ and $f^v(u)$, since this would immediately violate secrecy. Committee $\mathcal{S}^{c+2}$ reconstructs these values c_{uv} as well, and checks that $a_{uv} - b_{uv} = c_{uv}$. If this is not true, the parties publicly reconstruct using broadcast the (re-randomized) values $f^u(v)$, $f^v(u)$ and $F(u,v)$ that they hold (received from committee $\mathcal{S}^c$). Parties in the next committee $\mathcal{S}^{c+3}$ can then check whether $a_{uv} - f^u(v) = F(u,v)$ (resp. $b_{uv} - f^v(u) = F(u,v)$), and if the check fails, mark index u (resp. index v) as unhappy, and broadcast shares of the horizontal and vertical non-blinded polynomials at index u (resp. index v), i.e. shares of $F(u,j)$, $F(j,u)$ (resp. shares of $F(v,j)$, $F(j,v)$) for all j. Similar to [GIKR01], the idea here is then to take these new horizontal and vertical polynomials as the valid projections at index u, as long as they are consistent with enough of the other blinded polynomials defined by the public points $\{a_{uv}\}$ and $\{b_{uv}\}$. For this, we actually also need to let $\mathcal{S}^{c+3}$ to broadcast shares of $f^j(u)$ when index u is marked (resp. shares of $f^j(v)$ when index v is marked) for all j, so that $\mathcal{S}^{c+4}$ can check if these new reconstructed non-blinded polynomials are consistent with enough points of the blinded polynomials (and the broadcasted blindings), where for each inconsistent point satisfying $a_{ju} - f^j(u) \neq F(j,u)$ or $b_{vj} - f^j(v) \neq F(v,j)$, we mark index j as unhappy.

The dealer is then disqualified if the set of unhappy indices is (strictly) larger than t. We defer more details to Section 4.3.

Given this maximally-fluid VSS protocol, we can now begin to describe our approach towards designing a maximally-fluid MPC with guaranteed output delivery and perfect security. For this, we first need two "augmented" variants of our VSS protocol which we be helpful in the design of our perfectly secure maximally-fluid MPC with guaranteed output delivery. The idea of these "augmented" variants is borrowed from Gentry et al. [GHK$^+$21]. However, looking ahead, the way we use them to obtain our final construction, will necessarily have to be different. As discussed before, this is because, the protocol in [GHK$^+$21] is only statistically secure and assumes random corruptions of parties in each committee.

Duplicating VSS. In our MPC protocol it will be crucial that a dealer P_d^c can produce sharings of *the same* value s towards two distinct committees $\mathcal{S}^{c'}$, $\mathcal{S}^{c''}$, in such a way that even if the dealer is corrupted, then committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ agree on whether both sharings succeeded, and if so, when any of the committees executes the reconstruction procedure, they obtain the same value. This can be achieved with a minor modification of protocol VSS.ShareFuture: each P_i^c simply invokes ShareFuture for their blinding polynomial $f^{(i)}(x)$ towards committees $\mathcal{S}^k$ for $k \in \{c+1, c+2, c+3, c'\}$ (same as in the previous section) *but also*

[9] If any point cannot be reconstructed, parties set it to a default value.

towards committee $S^{c''}$. Since P_d^c can only get disqualified based on broadcast information, committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ agree on whether the protocol succeeded or not.

2.1.3 2-Level Verifiable Secret Sharing

We now describe how to enhance our VSS protocol, to achieve 1) A simplified sharing state, where committee $\mathcal{S}^{c'}$ holds an actual Shamir-sharing of s, meaning that each honest party $P_i^{c'}$ holds value $s_i := f_s(i)$ where $f_s(x)$ is a polynomial of degree chosen by the dealer such that $f_s(0) = s$; 2) Commitment to shares: parties will in addition hold shares, in the sense of the state after protocol $\mathsf{VSS.ShareFuture}$, of *each coefficient* of the polynomial $f_s(x)$. Since the VSS is linear, each $P_i^{c'}$ can compute a share of each other parties points $s_j = f_s(j)$ for $j \in [1, n]$. These states will be important for our multiplication protocol, and are not hard to achieve, given a VSS protocol.

The idea, adapted from the *Augmented VSS* of [GHK+21], works as follows. The dealer P_d^c chooses a polynomial of some degree t such that $f(0) = s$, and uses (the duplicate version of) $\mathsf{VSS.ShareFuture}(f_k)$ to distribute towards committees $\mathcal{S}^{c_1}$, $\mathcal{S}^{c'}$ for each coefficient f_k of $f(x)$ for $k \in [0, t]$[10]. Then, each P^{c_1} locally computes a share of $[f(i)]$ as $\sum_{k=0}^{t}[f_k]i^k$, and participates in protocol $\mathsf{VSS.RecFuture}(f(i))$ *privately* towards party $P_i^{c'}$. More details in Section 4.5, where we also describe a simple method to duplicate sharings of our 2-level VSS.

Given these variants of the VSS protocol, we are now ready to describe how to securely evaluate a circuit in the maximally-fluid model whilst achieving guaranteed output delivery.

2.1.4 Circuit Evaluation

Clients use our 2-level VSS protocol to distribute their inputs towards a future committee of servers. For the computation, we now want to maintain the following invariant for each computation gate g with input wire values x, y and output wire value z: there is some committee $\mathcal{S}^c$ holding states corresponding to a 2-level VSS of x and y. Our goal is that a future committee $\mathcal{S}^{c'}$ for $c' \geq c$ holds a 2-level VSS state of z.

If g is an addition gate, then parties in $\mathcal{S}^c$ can non-interactively compute the state for $z = x + y$ by simply adding their local states, exploiting the linear properties of the 2-level VSS.

For multiplication gates, we adapt the well-known perfectly-secure BGW multiplication approach to the fluid setting. Due to the interactive nature of this protocol, this does not come free of challenges. Recall that each party in $\mathcal{S}^c$ holds as part of their 2-level VSS states corresponding to x and y, evaluations $x_i = f_x(i)$ and $y_i = f_y(i)$ for some polynomials f_x and f_y such that $f_x(0) = x$ and $f_y(0) = y$. We ask that each party executes a 2-level VSS for the product $z_i = x_i \cdot y_i$ towards a future committee. The basic idea, is that, because one can write $z = \mathcal{L}(z_1, \ldots, z_n)$[11], each party in a future committee can then (locally) compute a 2-level VSS state for z, by applying $\mathcal{L}$ to the 2-level VSS states for all values z_i. However, it is clear that a dishonest party might share a value *different* from $x_i \cdot y_i$. Therefore, we need a mechanism to reassure the recipient committee that the 2-level VSS state received from party P_i^c is indeed a state of $z_i = x_i \cdot y_i$.

To carry out such a *multiplication proof*, each party P_i^c provides the recipient committee with 2-level VSS states of x_i, y_i, and z_i, but with a few tweaks 1) the polynomials chosen for the sharings of x_i and y_i must be consistent with the original 2-level VSS states for x and y,[12] and 2) party P_i^c provides two

[10] Here, it must hold that at least $c_1 \geq c + 4$.

[11] The explicit coefficients w_i can be derived via Lagrange interpolation.

[12] The type of consistency required between these polynomials and the known 2-level VSS states are provided in Section 4.6.

consistent 2-level VSS states for z_i: one using a degree t polynomial, and one using the degree-$2t$ product of the polynomials used to share x_i and y_i respectively.

In the non-fluid setting (where committee $\mathcal{S}^c$ can stay online for many rounds), each recipient P_j^c locally checks, for all parties P_i^c, that $x_{ij} \cdot y_{ij} = z_{ij}$, where x_{ij} denotes the share of x_i held by P_j^c. Intuitively, if these checks succeed, this means that the degree $2t$ polynomial used to share z_i evaluates to $x_i \cdot y_i$ in 0 (because $n - t \geq 2t + 1$ values completely determine the polynomial). The consistency guarantees on the different polynomials (the one with degree t and the one with degree $2t$) used to share z_i then implies that the degree t 2-level VSS state for z_i is also a correct state for $x_i \cdot y_i$. However, if the check $x_{ij} \cdot y_{ij} \neq z_{ij}$ fails, then party P_j^c raises a complaint towards P_i^c, who, in order to not fail their proof, must answer the complaint by broadcasting P_j's shares of all values x_i, y_i, and z_i.

These last steps constitutes a challenge in the fluid model, since at this point P_j^c is no longer online. To overcome this, their state must be held in a distributed fashion among an auxiliary committee $\mathcal{S}^{c_1}$ to enable this committee to answer the complaints on P_j^c's behalf. However, once a complaint has been answered by committee $\mathcal{S}^{c_1}$ (by reconstructing the values x_i and y_i of a party P_i^c that failed their multiplication proof), committee $\mathcal{S}^{c_1}$ themselves are not online anymore, and all the progress made so far is lost. To overcome this, we must make sure that all relevant information held by committee $\mathcal{S}^c$ is duplicated towards *two additional* committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c_2}$, one to broadcast complaints and one to answer them. With the help of these two auxiliary committees, the recipient committee $\mathcal{S}^{c'}$ is finally able to agree on a set of multiplication proofs that succeeded.

To complete the evaluation of the gate, in the non-fluid setting parties reconstruct the shares x_i and y_i of parties P_i^c that failed in their multiplication proofs. By now, it should be clear that executing this last step in the fluid-model means that, once again, the state achieved is lost. Fixing this last issue results in the need for one last auxiliary committee, agreeing on what multiplication proofs succeeded and reconstructing the values of cheaters to the committee that will finally hold the output state of the multiplication gate. More details can be found in Section 4.6 and Section 4.7.

Finally, for output gates, parties in committee $\mathcal{S}^c$ can simply run the reconstruction procedure for the 2-level VSS to reconstruct the output value towards the appropriate clients.

2.2 Maximally-Fluid MPC with Computational Security

As a second contribution, we also provide a maximally-fluid MPC cryptographic protocol with guaranteed output delivery based on (non-interactive) equivocal linearly homomorphic commitments (c.f. Definition 4), secure up to $t < n/2$ corruptions in each committee.

Our Approach. This protocol also follows the BGW-paradigm of emulating the computation on a gate-by-gate basis, where clients use a VSS scheme to distribute their inputs to the parties, who will compute shares of each of the wires, and finally reconstruct the output to the clients. We design similar sub-protocols as in our protocol with perfect security, and many of these will use homomorphic commitments as a crucial building block.

Comparison with YOSO. The basic blueprint of this protocol is inspired from the information-theoretic YOSO MPC by Gentry et al. [GHK$^+$21], that achieves security with guaranteed output delivery against random corruptions of $t < n/2$ parties in every committee. As discussed previously, their construction crucially relies on parties having access to secure channels to the future, which they are unable to instantiate information-theoretically for $t < n/2$ corruptions. Our first step is to design a mechanism that emulates these channels using homomorphic commitments. However, we note that simply combining this with the rest of their construction does not give us our maximally-fluid MPC. This is because, their construction works when the adversary is only allowed to make random corruptions in each committee. As such, we must devise new techniques using homomorphic commitments that allow us to achieve security even under arbitrary adversarial strategies. We now discuss our techniques in more detail.

Secure Channels to the Future. We implement secure channels to future committees similarly as in the case with perfect security. However, since we consider a higher corruption threshold, $t < n/2$, we cannot simply rely on plain error correction. Instead, parties will broadcast commitments to (coefficients of) the polynomials used to share their values. Even more, every time a party wishes to re-share an intermediate value, it will be crucial to re-use the commitment to the constant coefficient, thereby ensuring that the proper value is being re-shared. In more detail, consider a sender $P_s^c \in \mathcal{S}^c$ who wants to send a message to a recipient in a future committee $P_r^{c'} \in \mathcal{S}^{c'}$, with $c' > c + 2$ (sending to nearer committees is straightforward). The idea is that the sender P_s^c first samples a degree-t polynomial $f(x)$ such that $f(0) = m$, broadcast commitments to all its coefficients, and send to P_i^{c+1} message $m_i = f(i)$ along with its opening information. Then, each party P_i^{c+1} acts as the sender in the described protocol, with input m_i *but* using instead the publicly known commitment to m_i as the commitment to the constant term of the polynomial used to share m_i. This is repeated until committee $\mathcal{S}^{c'-1}$, who will send all the respective values and opening information to $P_r^{c'}$, who can perform reconstruction step-by-step.

Distributed Commitments. This primitive allows a dealer P_d^c to commit to a value s towards a future committee $\mathcal{S}^{c'}$, and later open the original value towards another further committee $\mathcal{S}^{c''}$. If the dealer P_d^c is honest, the opened value is s, and no information about s is revealed before the opening phase. Moreover, even if the dealer P_d^c is corrupted, the commit phase determines a unique value s', such that the opening phase can only output s' or $\perp$. This primitive is also refered to as *weak* secret sharing in the literature [GIKR01].

The idea is as follows. The sender $P_s^c \in \mathcal{S}^c$ samples random degree-t polynomials $f(x)$ and $r(x)$, such that $f(0) = s$, and broadcast public commitments $\mathsf{com}_0, \ldots, \mathsf{com}_t$, where $\mathsf{com}_k = \mathsf{Commit}(f_k, r_k)$, is a commitment to the k-th coefficient f_k of the polynomial $f(x)$, using randomness r_k which is k-th coefficient of the polynomial $r(x)$. The sender then sends the evaluation points $(s(i), r(i))$ using send future to party $P_i^{c'}$. In order to reconstruct, the parties in committee $\mathcal{S}^{c'}$ will broadcast these pairs, and each party $P_i^{c''} \in \mathcal{S}^{c''}$ checks for each received pair whether it is consistent with the corresponding commitment. If there are more than t pairs, interpolate a degree-t polynomial $f'(x)$ and output $f'(0)$. Otherwise, output $\perp$.

Remark 1. *We can achieve a distributed commitment that allows to commit to the same value towards separate committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$, such that even if P_d^c is corrupted, there exists a unique value s' such that the value that is opened by either committee is s' or $\perp$. This is achieved by letting the dealer P_d^c execute the above protocol towards committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ with polynomials $f(x)$ and $f'(x)$ such that $f(0) = s = f'(0)$, but using the same commitment $\mathsf{com}_0 := \mathsf{Commit}(f_0, r_0)$ for the constant coefficient.*

Verifiable Secret Sharing. In order to achieve verifiable secret sharing, we design a protocol $\mathsf{VSS.ShareFuture}$ that works as follows: the dealer P_d^c, who holds input s, samples a uniform random degree-t polynomial $f(x)$ with $f(0) = s$, and (duplicate) commits to each coefficient f_k, $k \in [0, t]$, towards committees $\mathcal{S}^{c_1}, \mathcal{S}^{c'}$.

This results in a matrix $\mathbf{D} = [\mathsf{com}_{i,j}]_{0 \leq i,j \leq t}$ of public commitments, where $\mathsf{com}_{i,j}$ is a commitment to the j-th coefficient of the polynomial used to share f_i. Note that by linearity of the commitment scheme, all parties implicitly hold commitments to each point $f(i)$.

The dealer P_d^c sends $s_i = f(i)$ and the opening information for this value (which can also be computed by linearity) to $P_i^{c_1}$, using the send to the future protocol. Each party $P_i^{c_1}$ can check that the received information is consistent with the published commitments, and broadcast a complaint if the check does not succeed. If the check succeeds, $P_i^{c_1}$ commits to s_i towards committee $\mathcal{S}^{c'}$. And to ensure that party $P_i^{c_1}$ commits to the value they received from P_c^d, the party uses the commitment to the constant term that is implicit from the published commitments in $\mathbf{D}$. If a complaint was raised by $P_i^{c_1}$, parties in committee $\mathcal{S}^{c'}$ publicly open s_i (and if the opened value is $\perp$, the dealer is disqualified).

The reconstruction is as follows: for each index i corresponding to a party that did not complain, parties jointly reconstruct s_i. The final committee $\mathcal{S}^{c''}$ then uses any $t+1$ reconstructed shares to interpolate the secret.

Remark 2. *Observe that by having both P_d^c and parties in committee $\mathcal{S}^{c_1}$ duplicate distribute commitments of s_i for all $i \in [1, n]$ towards committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{\tilde{c}}$ for some $\tilde{c} \geq c'$, one guarantees that if P_d^c is not disqualified, then both $\mathcal{S}^{c'}$ and $\mathcal{S}^{\tilde{c}}$ hold sharings of the same value.*

2-Level Verifiable Secret Sharing. To simplify the description of the MPC protocol, it will be helpful that each party holds as part of their state a Shamir share of each wire value. For that, we modify the VSS as follows. The dealer P_d^c uses the above duplicate VSS to distribute shares of f_k for all $k \in [0, t]$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$, where $f(x)$ is a uniform random degree-t polynomial with $f(0) = s$. Then, each party in $\mathcal{S}^{c_1}$ (privately) reconstructs towards party $P_i^{c'}$ the value $s_i = f(i)$. Notice that committee $\mathcal{S}^{c'}$ also holds sharings of all values $f(j)$ for $j \in [1, n]$ thanks to linearity of our VSS. This version of VSS can also be duplicated similarly to the previous one.

Circuit Evaluation. Similarly to the protocol with perfect security, clients use our committing (cryptographic) VSS protocol to distribute their inputs towards a future committee, and for each computation gate, there is an initial committee $\mathcal{S}^c$ holding states resulting from a 2-level VSS of the input wire values x and y, and in the end another future $\mathcal{S}^{c'}$ for $c' \geq c$ holds a 2-level VSS state of the value z for the output wire. Addition gates are processed locally.

For multiplication gates, we adapt the well-known protocol of Cramer et al. [CDD$^+$99]. Each party in $\mathcal{S}^c$ holds, as part of their 2-level VSS states to x and y, actual Shamir-shares x_i and y_i of each value, and computes a 2-level VSS for x_i, y_i (but using the known state as a sharing to the constant coefficient of the used polynomial) and a fresh 2-level VSS for $z_i = x_i \cdot y_i$ towards a future committee.

As before, parties carry out a *multiplication proof*. For this, each party $P_i^c \in \mathcal{S}^c$ samples a random value β and computes $\gamma = x_i\beta$, and uses duplicate 2-level VSS to share both values towards the next 3 committees $\mathcal{S}^{c_1}, \mathcal{S}^{c_2}, \mathcal{S}^{c_3}$. In addition, each party also uses 2-level VSS to share a random value r_i towards $\mathcal{S}^{c_1}$.

Parties in $\mathcal{S}^{c_1}$ compute $r = \sum_j r_j$ (for the indices j that were had a successful sharing) and publicly reconstruct r. Committee $\mathcal{S}^{c_2}$ then reconstructs $r' = ry + \beta$. And $\mathcal{S}^{c_3}$ reconstructs $r'' = r'x - rz - \gamma$. Finally, $\mathcal{S}^{c'}$ simply declares the proof successful if and only if $r'' = 0$. As shown in [CDD$^+$99], the probability that $z_i \neq x_i \cdot y_i$ when $r'' = 0$ is $1/\mathbb{F}$ (this only happens for one possible value of r, which is chosen uniformly at random).

3 Preliminaries

In this section, we recall the model of Fluid MPC and the definition of linearly-homomorphic equivocal commitements.

3.1 Fluid MPC: Model and Security

We start by recalling the security model of Fluid MPC. Some of the text in this section is taken verbatim from [CGG$^+$21].

Client-Server Model. We consider a network of m-clients and N-servers $\mathcal{S}$ and denote by $(\vec{n} = (n_1, \ldots, n_E), E)$ the partitioning of servers[13] into E tuples (corresponding to epochs) where we assume that each tuple has n parties (corresponding to the committee in that epoch), i.e. $\mathcal{S}^\ell \subset \mathcal{S}$ such that $\forall \ell \in [E]$, $|\mathcal{S}^\ell| = n$.

Similar to the client-server setting in standard MPC, only the m clients have an input (and receive output), computing a function $f : X_1 \times \cdots \times X_m \to Y_1 \times \cdots \times Y_m$, where for each $i \in [m]$, X_i and Y_i are the input and output domains of the i-th client. We assume that a protocol in this model proceeds in three stages: (1) in the *input stage*, the clients hand their inputs to the first committee of servers (2) in the *execution stage*, only the servers participate in the computation of the function and (3) finally in the *output stage*, the clients to participate to reconstruct and learn the outputs. Progression of the execution stage takes place in epochs. We work in the *maximally-fluid* setting, where each epoch ℓ only comprises of a single round of communication from committee $\mathcal{S}^\ell$ to $\mathcal{S}^{\ell+1}$.[14]

Layered Circuits. For simplicity, in this work, we will work with a *layered arithmetic circuit representation* of the function f. By layered circuits, we mean circuits that can be decomposed into well-defined layers such that the output of gates on a layer ℓ are only used as input to the gates on layer $\ell + 1$. As discussed in [CGG$^+$21], all arithmetic circuits can be transformed into such a circuit with polynomial overhead.

Corruption Model. The environment $\mathcal{E}$ can determine to corrupt a party, and on doing so, hands the local state of the corrupted party to the adversary $\mathcal{A}$. We consider the *honest-majority* setting, where we restrict $(\mathcal{A}, \mathcal{E})$ to the setting where the adversary $\mathcal{A}$ controls a minority of the clients as well as a minority of servers in every committee in an epoch. For clients, we assume that the environment $\mathcal{E}$ specifies the list of corrupted clients at the start of the protocol, i.e. we assume *static corruption* for the clients.

The servers perform the bulk of the computation, and their participation is dynamic. Following the terminology from [CGG$^+$21], we consider two types of corruption strategies: R-adaptive and NR-adaptive. Here R and NR denote whether or not the servers are corrupted with *retroactive effect* or not. In particular, if the corrupts take place with retroactive effect, the adversary is allowed limited information from prior epochs. Specifically, when corrupting a server $S \in \mathcal{S}^\ell$ in epoch ℓ, the adversary learns private states of the server in all prior epochs (if the server has been in a committee before). Therefore, the server S is then assumed to have been (passively) corrupt in every epoch $j < \ell$. In order to prevent the adversary from arbitrarily learning information about prior epochs, the adversary is limited to corrupting servers in epoch ℓ as long as corrupting a server S and its retroactive effect of considering S to be corrupted in all prior epochs does not cross the designated corruption threshold in any epoch. We now formally define R-adaptive and NR-adaptive.

Definition 1 (R-adaptive Adversary)**.** *We say that the $(\mathcal{A}, \mathcal{E})$ results in an* R-adaptive *adversary $\mathcal{A}$ if the environment $\mathcal{E}$ can statically corrupt a set T of the clients (at the start of the protocol) and corrupt the servers in an adaptive manner. Specifically, in epoch ℓ, the environment $\mathcal{E}$ can adaptively choose to corrupt a set of servers $T^\ell \subset [n_\ell]$ from the set $\mathcal{S}^\ell$, where T^ℓ corresponds to a canonical mapping based on the ordering of servers in $\mathcal{S}^\ell$. On $\mathcal{E}$ corrupting the server, $\mathcal{A}$ learns its entire past state and can send messages on its behalf in*

[13] We use servers and parties interchangeably throughout the paper.

[14] Therefore, we use the terms epoch and round interchangeably throughout this paper.

epoch ℓ. The set of servers that $\mathcal{E}$ can corrupt, and its corresponding retroactive effect, will be determined by the corruption threshold τ specifying that $\forall \ell,\ |T^\ell| < \tau \cdot n_\ell$.

Definition 2 (NR-adaptive Adversary)**.** *We say that the $(\mathcal{A}, \mathcal{E})$ results in an* NR-adaptive *adversary $\mathcal{A}$ if the environment $\mathcal{E}$ can statically corrupt a set T of the clients (at the start of the protocol) and corrupt the servers in an adaptive manner with no retroactive effect. The corruption process is similar to the case of* R-adaptive *adversaries, except that the environment $\mathcal{E}$ can corrupt any server in epoch ℓ as long as the number of corrupted servers in epoch ℓ are within the corruption threshold. Any protocol that achieves security against such an adversary necessarily requires either (a) erasure of state, or (b) disjoint committees.*

Committee Selection. As discussed before, we assume that the committees are determined dynamically such that committee for epoch $\ell + 1$ is determined and known to everyone before $\mathcal{S}^\ell$ wants to send a message to $\mathcal{S}^{\ell+1}$ in epoch ℓ. We consider the functionality $f_{\mathsf{committee}}$ to capture this setting. In an epoch ℓ, if the environment $\mathcal{E}$ provides input nominate to a party at the start of the round, it relays this message to $f_{\mathsf{committee}}$ to indicate that it wants to be considered in the committee for epoch $\ell + 1$. The functionality computes the committee using the sampling function Sample, from the set of parties $\mathcal{P}$ that have been "nominated." The environment $\mathcal{E}$ is also allowed a separate input elect that specifies the cut-off point for the functionality to compute the committee. The cut-off point corresponds to the start of round ℓ where the parties in $\mathcal{S}^\ell$ are made aware of the committee $\mathcal{S}^{\ell+1}$ via a broadcast from $f_{\mathsf{committee}}$. Similar to [CGG$^+$21], our protocol design is agnostic to this choice and only requires that the committee $\mathcal{S}^\ell$ knows committee $\mathcal{S}^{\ell+1}$ before they start sending them messages.

Functionality $f_{\mathsf{committee}}$

Hardcoded: Sampling function Sample : $\mathcal{P} \mapsto \mathcal{P}$.

1. Set $\mathcal{P} := \varnothing$
2. When party P_i sends input nominate, $\mathcal{P} := \mathcal{P} \cup \{P_i\}$.
3. When the environment $\mathcal{E}$ sends input elect, compute $\mathcal{P}' \leftarrow \mathsf{Sample}(\mathcal{P})$ and broadcast $\mathcal{P}'$ as the selected committee.

Security Definition. We provide a definition of fluid MPC that corresponds to security with *guaranteed output delivery*. The security of a protocol (with respect to a functionality f) is defined by comparing the real-world execution of the protocol with an ideal-world evaluation of f by a trusted party. More concretely, it is required that for every adversary $\mathcal{A}$, which attacks the real execution of the protocol, there exist an adversary Sim, also referred to as a simulator in the ideal-world such that no environment $\mathcal{E}$ can tell whether it is interacting with $\mathcal{A}$ and parties running the protocol or with Sim and parties interacting with f. As mentioned earlier, the environment $\mathcal{E}$ (i) determines the inputs to the parties running the protocol in each round; (ii) sees the outputs to the protocol; and (iii) interacts in an arbitrary manner with the adversary $\mathcal{A}$. In this context, one can view the environment $\mathcal{E}$ as an interactive distinguisher.

In the **real execution** of the $(\vec{n}, E)$-party protocol Π for computing f in the presence of $f_{\mathsf{committee}}$ proceeds first with the environment passing the inputs to all the clients, who then pre-process their inputs and hand it off to the servers in $\mathcal{S}^1$. The protocol then proceeds in epochs as described earlier in the presence of an adversary $\mathcal{A}$ and environment $\mathcal{E}$. The environment $\mathcal{E}$ at the start of the protocol chooses a subset of clients $T \subset [m]$ to corrupt and hands their local states to $\mathcal{A}$. As discussed, the corruption of the clients is static, and thus fixed for the duration of the protocol. The honest parties follow the instructions of π. Depending on whether $\mathcal{A}$ is R-adaptive or NR-adaptive, $\mathcal{E}$ proceeds with corrupting servers accordingly and handing over their states to $\mathcal{A}$ who then sends messages on their behalf.

The execution of the above protocol defines $\mathsf{REAL}_{\Pi,\mathcal{A},T,\mathcal{E},f_{\mathsf{committee}}}(z)$, a random variable whose value is determined by the coin tosses of the adversary and the honest parties. This random variable contains (a) the output of the adversary (which may be an arbitrary function of its view); (b) the outputs of the uncorrupted clients; and (c) list of all the corrupted servers $\{T^{\ell}\}_{\ell\in[E]}$.

The **ideal world execution** is defined similarly to prior works. We formally define the ideal execution for the case of retroactive adaptive security, and the analogous definition for non-retroactive adaptive security can be obtained by appropriate modifications. Roughly, in the ideal world execution, the participants have access to a trusted party who computes the desired functionality f. The participants send their inputs to this trusted party who computes the function and returns the output to the participants.

More formally, an ideal world execution for a function f in the presence of $f_{\mathsf{committee}}$ with adversary Sim proceeds as follows:

- **Clients send inputs to the trusted party:** The clients send their inputs to the trusted party, and we let x'_i denote the value sent by client C_i. The adversary Sim sends inputs on behalf of the corrupted clients.
- **Corruption Phase of servers:** The trusted party initializes $\ell = 1$. Until Sim indicates the end of the current phase (see below), the following steps are executed:
    1. Trusted party sends ℓ to Sim and initializes an *append-only* list $\mathsf{Corrupt}^{\ell}$ to be $\varnothing$.
    2. Sim then sends pairs of the form (j, i) where j denotes epoch number and i denotes the *index of the corrupted server* in epoch $j \leq \ell$. Upon receiving this, the trusted party appends i to the list $\mathsf{Corrupt}^{j}$. This step can be repeated multiple times.
    3. Sim sends continue to the trusted party, and the trusted party increments ℓ by 1.
- **Trusted party sends output to all clients:** The trusted party computes $f(x'_1, \ldots, x'_m) = (y_1, \ldots, y_m)$ and sends $\{y_i\}_{i\in T}$ to the adversary Sim and all uncorrupted clients.
- **Outputs:** Sim outputs an arbitrary function of its view, and the honest parties output the values obtained from the trusted party.

Sim also interacts with the environment $\mathcal{E}$ in an identical manner to the real execution interaction between $\mathcal{E}$ and $\mathcal{A}$. In particular this means, Sim cannot rewind $\mathcal{E}$ or look at its internal state. The above ideal execution defines a random variable $\mathsf{IDEAL}_{\Pi,\mathsf{Sim},T,\mathcal{E},f_{\mathsf{committee}}}(z)$ whose value is determined by the coin tosses of the adversary and the honest players. This random variable containing the (a) output of the ideal adversary Sim; (b) output of the honest parties after an ideal execution with the trusted party computing f where Sim has control over the adversary's input to f; and (c) the lists $\{\mathsf{Corrupt}^{\ell}\}_{\ell}$ of corrupted servers output by the trusted party. If Sim sends abort in the *corruption phase of the server*, the trusted party outputs the lists that have been updated until the point the abort message was received from Sim.

Having described the real and the ideal worlds, we now define security.

Definition 3. *Let $f : X_1 \times \cdots \times X_m \to Y_1 \times \cdots \times Y_m$ be a functionality and let Π be a fluid MPC protocol for computing f with m clients, N servers and E epochs. We say that Π achieves τ-perfect/statistical/computational retroactive adaptive security (resp. non-retroactive adaptive security) with guaranteed output delivery, if for every probabilistic adversary $\mathcal{A}$ corrupting up to τ-fraction of parties in every committee in the real world, there exists a probabilistic simulator* Sim *in the ideal world such that for every* *probabilistic environment $\mathcal{E}$ if $\mathcal{A}$ is* R-adaptive *(resp.* NR-adaptive*) controlling a subset of servers $T^\ell \subseteq \mathcal{S}^\ell$, $\forall \ell \in [E]$ s.t. $|T^\ell| < \tau \cdot n_\ell$ and less than $\tau \cdot m$ clients, it holds that for all auxiliary input $z \in \{0,1\}^*$*

$$\mathsf{IDEAL}_{f,\mathsf{Sim},T,\mathcal{E},f_{\mathit{committee}}}(z) \approx \mathsf{REAL}_{\Pi,\mathcal{A},T,\mathcal{E},f_{\mathit{committee}}}(z)$$

where $\approx$ denotes identically distributed/statistically close/computationally close.

3.2 Linearly-Homomorphic Equivocal Commitments

We state the definition of non-interactive linearly-homomorphic equivocal commitments.

Definition 4. *We define a* ***non-interactive commitment scheme*** *with message space $\mathcal{M}$, randomness space $\mathcal{R}$, and commitment space $\mathcal{C}$ to be a 4-tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Equiv})$ *such that:*

- $(\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\kappa)$*: The setup algorithm takes as input the security parameter κ and produces a set of public parameters* pp *and a trapdoor τ.*

- $\mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, m, r)$*: The commit algorithm takes as input the public parameters* pp*, a message $m \in \mathcal{M}$, and randomness $r \in \mathcal{R}$ and produces a commitment* $\mathsf{com} \in \mathcal{C}$*;*

- $\mathsf{b} \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, m, r)$*: The open algorithm takes as input the public parameters* pp*, a commitment* $\mathsf{com} \in \mathcal{C}$*, a message $m \in \mathcal{M}$, and randomness $r \in \mathcal{R}$ and outputs a bit* $\mathsf{b} \in \{0,1\}$*;*

- $r \leftarrow \mathsf{Equiv}(\mathsf{pp}, \tau, \mathsf{com}, m)$*: The equivocation algorithm takes as input the public parameters* pp*, a trapdoor τ, a commitment* $\mathsf{com} \in \mathcal{C}$*, and a message $m \in \mathcal{M}$ and outputs randomness $r \in \mathcal{R}$;*

We say that such a commitment scheme is:

- ***Perfectly Hiding*** *if for all $m' \neq m$ and for* $(\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\kappa)$ *the distributions* $(\mathsf{Commit}(\mathsf{pp}, m, r), \tau)$ *and* $(\mathsf{Commit}(\mathsf{pp}, m', r), \tau)$ *are equal, where the distribution is over the randomness r;*

- ***Computationally Binding*** *if any PPT adversary $\mathcal{A}$ that does not know τ has a negligible probability (in the security parameter κ) of outputting (m, r) and (m', r') such that $m \neq m'$ and* $\mathsf{Commit}(\mathsf{pp}, m, r) = \mathsf{Commit}(\mathsf{pp}, m', r')$*;*

- ***Linearly Homomorphic*** *if there exist a binary function $+ : \mathcal{C}^2 \to \mathcal{C}$ and an element* $\mathsf{0} \in \mathcal{C}$ *such that* $(\mathcal{C}, +, \mathsf{0})$ *is an abelian group, a binary function $+ : \mathcal{M}^2 \to \mathcal{M}$ and an element* $\mathsf{0} \in \mathcal{M}$ *such that* $(\mathcal{M}, +, \mathsf{0})$ *is an abelian group, and a binary function $+ : \mathcal{R}^2 \to \mathcal{R}$ and an element* $\mathsf{0} \in \mathcal{R}$ *such* $(\mathcal{R}, +, \mathsf{0})$ *is an abelian group, and for all $m, m' \in \mathcal{M}$ and all $r, r' \in \mathcal{R}$ it holds that* $\mathsf{Commit}(\mathsf{pp}, m + m', r + r') = \mathsf{Commit}(\mathsf{pp}, m, r) + \mathsf{Commit}(\mathsf{pp}, m', r')$ *for all* $(\mathsf{pp}, \tau) \leftarrow \mathsf{Setup}(1^\kappa)$*;*

- ***Equivocal*** *if* $\mathsf{Open}(\mathsf{pp}, \mathsf{com}, m, \mathsf{Equiv}(\mathsf{pp}, \tau, \mathsf{com}, m)) = 1$ *for all* $\mathsf{com} \in \mathcal{C}$*, $m \in \mathcal{M}$, and* $(\mathsf{pp}, \kappa) \leftarrow \mathsf{Setup}(1^\kappa)$.

In our setting we will consider $\mathcal{M} = \mathcal{R} = \mathbb{F}$ with the group structure induced by the addition operation. We exploit the $\mathbb{Z}$-module structure induced by the addition on $\mathcal{C}$ throughout the paper and employ the notation $\mathsf{com} \cdot n$ *for some $n \in \mathbb{N}$ to mean the addition* $\mathsf{com} + \cdots + \mathsf{com}$ *with itself n-many times.*

Commitments with these property can be instantiated using a variety of assumptions such as discrete-log (like Pedersen commitments [Ped92]) or lattice based assumptions (like LWE or SIS [GVW15]). For simplicity of notation, we make the public parameters pp implicit in the Commit function.

4 Perfectly-Secure Maximally-Fluid MPC with Guaranteed Output Delivery

In this section, we present our first and main protocol which is a perfectly secure maximally-fluid MPC that achieves guaranteed output delivery against an unbounded adversary who corrupts up to $1/3^{\text{rd}}$ of the parties in each committee. We assume that each committee in this protocol consists of n-servers. We use the terms server and party interchangeably throughout this section. As discussed in Section 2.1, towards designing our main protocol, we first need to design several other subprotocols.

For simplicity, in each of the lemmas and theorems in this section we simply use the term "adversary", without specifying if the adversary is R-adaptive or NR-adaptive (c.f. Definitions 1 and 2). All of our formal proofs for these lemmas and theorems implicitly assumes an NR-adaptive adversary — this is only simplicity of presentation. At the end of this section, we briefly discuss how all of these subprotocols and our main protocol are also secure against an R-adaptive adversary.

4.1 Secure Channels to Future Committees

We begin by presenting a maximally-fluid subprotocol called SendFuture, that allows a party $P_s^c \in \mathcal{S}^c$ (called the sender party) in committee $\mathcal{S}^c$ to securely send a message m to another party $P_r^{c'} \in \mathcal{S}^{c'}$ (called the recipient party) in some future committee $\mathcal{S}^{c'}$. In the fluid model, since the identity of the parties in future committees are not known in advance, intermediate committees must help to pass this message on. However, this must be done in a manner which ensures that that an adversary controlling at most $t < n/3$ servers in each intermediate committee, can neither tamper with, nor learn any information about, the message m. One can think of this protocol as constructing a one-directional secure channel between parties in two non-consecutive committees using one-directional secure channels between all parties in consecutive committees. We formalize the properties of this protocol in Lemmas 1 and 2.

Protocol SendFuture(m)

Party P_s^c is the sender, with input m, and party $P_r^{c'}$, for some $c' > c$ is the receiver. For all $i \in [1, n], i \neq s$ party P_i^c initializes $\mathbf{m}_i := \perp$, while party P_s^c initializes $\mathbf{m}_i := m^a$.

For all committees $k \in [c, c']$, each party P_i^k does the following:

- If $k \geq c + 1$, let $\mathbf{m}_i := (\mathbf{m}_{1i}, \ldots, \mathbf{m}_{ni})$, where $\mathbf{m}_{ji}$ denotes the message received from P_j^{k-1};
- If $k = c' - 1$, send $\mathbf{m}_i$ to $P_r^{c'}$;
- If $k \neq c' - 1$, for each entry y of $\mathbf{m}_i$, sample a uniform random polynomial $f_y(x)$ such that $\deg f_y(x) = t$ and $f_y(0) = y$. Send $(f_y(j))_{y \in \mathbf{m}_i}$ to P_j^{k+1} for all $j \in [1, n]$;

Upon receiving messages $\mathbf{m}_i$ from $P_i^{c'-1}$ for all $i \in [1, n]$, party $P_r^{c'}$ performs Reed-Solomon decoding layer-by-layer to recover a value m', where $m' = \perp$ if the last reconstruction fails.[b]

[a] For each party $P_i^{c+\ell}$ then $\mathbf{m}_i \in \mathbb{F}^{\ell+1}$.

[b] To illustrate this, assume $c' = c + 3$. Party $P_r^{c'}$ receives matrices $\mathbf{m}_{ir} = (y_{ijk})_{j,k \in [1,n]}$. For all $i \in [1, n]$, they first perform, for all $j \in [1, n]$ Reed-Solomon decoding on vector $(y_{ij1}, \ldots, y_{ijn})$ to recover value y_{ij}. Then, they repeat the operation on vector $(y_{i1}, \ldots, y_{in})$ to recover value y_i, and finally, they use Reed-Solomon decoding one last time on vector $(y_1, \ldots, y_n)$ to recover a value m' (possibly $\perp$).

Lemma 1 (Robustness in SendFuture)**.** *Let P_s^c be an honest party with input m. For any unbounded adver-*

sary $\mathcal{A}$, who corrupts $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c'-1]$ in SendFuture*, the output of $P_r^{c'}$ is $m' = m$.*

Proof. Assuming the adversary corrupts $t < n/3$ parties in each committee, there are at least $2t+1$ honest parties in each committee. P_s^c uses a degree t polynomial to secret share m towards committee $\mathcal{S}^{c+1}$. It is easy to see that as long as at least $2t+1$ honest parties in committee $\mathcal{S}^{c+1}$, honestly secret share the shares received from P_s^c, it is possible to use Reed-Solomon decoding on their shares of shares to correctly decode the m. Similarly, the parties in committee $\mathcal{S}^{c+1}$ further secret share the share that they received from P_s^c towards the parties in committee $\mathcal{S}^{c+2}$. Now, given these "shares of shares" received by the honest parties in $\mathcal{S}^{c+2}$, it is possible to reconstruct the shares of the honest parties in $\mathcal{S}^{c+1}$ using Reed-Solomon decoding. And as argued earlier, given these $2t+1$ shares, it is possible to recover m.

This same recursive argument can be extended to any number of intermediate committees. Therefore, as long as $2t+1$ honest parties in each committee compute the shares of their shares honestly, the recipient can always reconstruct the output using Reed-Solomon decoding. □

Lemma 2 (Privacy in SendFuture). *Let P_s^c and $P_r^{c'}$ be honest parties. Then the view of an unbounded adversary who corrupts $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c'-1]$ is identically distributed in executions* SendFuture(m) *and* SendFuture(m')*, for any $m \neq m'$.*

Proof. For all polynomials sampled by all the honest parties (including the sender P_s^c and honest parties in the intermediate committees) in SendFuture, the adversary only receives t evaluations of these polynomials. However, since each of these are degree-t polynomials, t evaluations do not reveal anything about the secret and hence it is easy to see that the view of the adversary is independent of the message. □

4.2 Shamir Sharing to Future Committees

Building on our subprotocol SendFuture from the previous section, we now present a maximally-fluid protocol called ShareFuture that allows a party $P_d^c \in \mathcal{S}^c$ (called the dealer) in committee c to secret share a value s towards the parties in committee $\mathcal{S}^{c'}$. Similar to SendFuture, we will take the help of intermediate committees to accomplish this task. However, for security this must be done in such a way that, if the dealer is honest, an adversary controlling at most $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c, c']$ does not learn any information about the secret s and they should not be able to tamper with the shares of honest parties in $\mathcal{S}^{c'}$. Moreover, we want that if the dealer was honest, then the shares received by the parties in $\mathcal{S}^{c'}$ must be linearly homomorphic. We formalize these properties in Lemmas 3, 4, 5.

Protocol ShareFuture$(s) \to (s_1, \ldots, s_n)$

The dealer $P_d^c \in \mathcal{S}^c$, who holds input s, wants to secret share s with servers in committee $\mathcal{S}^{c'}$, and does the following:

- Sample a uniform random polynomial $f(x)$ of degree at most t such that $f(0) = s$;
- For all $j \in [1, n]$ do SendFuture$(f(j))$ towards party $P_j^{c'}$.

Protocol RecFuture$(s_1, \ldots, s_n) \to s$

- **Committee** c'**:** Each party $P_i^{c'}$ broadcast their share s_i[a];

- **Committee c'':** Each party $P_i^{c''}$ performs Reed-Solomon decoding on the vector $(s_1, \dots, s_n)$ to recover s. If this fails, output $\perp$.

[a] By replacing broadcast with an execution of SendFuture(s_i) towards party $P_j^{c''}$ we can obtain a private version of RecFuture.

Lemma 3 (Robustness of ShareFuture)**.** *Let P_d^c be an honest party with input s. For any unbounded adversary $\mathcal{A}$, who corrupts $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$ in* ShareFuture*, then the output of* RecFuture *is s.*

Proof. The dealer in ShareFuture simply uses an invocation of SendFuture for each share. From Lemma 1 (i.e., robustness of SendFuture), we know that the each party in committee $\mathcal{S}^{c'}$ will receive the correct share. Now as long as all the honest parties in this committee (i.e. $\geq 2t+1$ parties) broadcast their correct shares, it is possible to reconstruct the correct secret s. □

Lemma 4 (Linearity of ShareFuture)**.** *Assume that committee $\mathcal{S}^{c'}$ hold states* ShareFuture(x) *and* ShareFuture(y) *relative to honest (possibly distinct) dealers $P_d^{c_1}$ and $P_{d'}^{c_2}$. For all linear functions $\mathcal{L} : \mathbb{F}^2 \to \mathbb{F}$ it holds that*[15] RecFuture$(\mathcal{L}(x_1, y_1), \dots, \mathcal{L}(x_n, y_n)) = \mathcal{L}(x, y)$.

Proof. Since $P_d^{c_1}$ and $P_{d'}^{c_2}$ are honest, there exist polynomials $f_x(x)$ and $f_y(x)$ of degree at most t and such that $f_x(0) = x$ and $f_y(0) = y$. From Lemma 1 we know that, for each honest $P_i^{c'}$, it holds that $x_i = f_x(i)$ and $y_i = f_y(i)$. Therefore, at least $n - t \geq 2t + 1$ shares $\mathcal{L}(x_i, y_i)$ will lie on polynomial $f_{\mathcal{L}(x,y)} := \mathcal{L}(f_x(x), f_y(x))$ and the output of RecFuture will be $\mathcal{L}(x, y) = f_{\mathcal{L}(x,y)}(0)$. □

Lemma 5 (Privacy of ShareFuture)**.** *Let P_d^c be an honest party. Then the view of an unbounded adversary who corrupts $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$ is identically distributed until the end of executions* ShareFuture(s) *and* ShareFuture(s')*, for any $s \neq s'$.*

Proof. Privacy of the shares sent by the dealer P_d^c to the honest parties in committee c' follows from Lemma 2 (i.e., privacy of SendFuture). As a result, the adversary only sees t evaluations of the polynomial f used by the dealer P_d^c to secret share s. Since f is a degree t polynomial, these t evaluations keep s hidden and the view of the adversary remains independent of s.

□

4.3 Verifiable Secret Sharing

Our subprotocol ShareFuture from the previous section does not provide any guarantee when the dealer is dishonest. To remedy this, in this section, we design a maximally-fluid verifiable secret sharing protocol VSS := (VSS.ShareFuture, VSS.RecFuture). It guarantees that even if the dealer $P_d^c \in \mathcal{S}^c$ is corrupted, after the sharing phase of the protocol terminates, a designated committee $\mathcal{S}^{c'}$ for some $c' \geq c + 4$ holds a state that uniquely determines a value that can be reconstructed during the reconstruction phase. We formalize the main properties of this protocol in Lemmas 8, 9, 10,11. This is the main technical tool in our maximally-fluid MPC protocol.

[15] This generalizes easily to the case of n-sharings and linear functions $\mathbb{F}^n \to \mathbb{F}$.

Protocol VSS.ShareFuture$(s) \to (\mathbf{s}_1, \ldots, \mathbf{s}_n)$

- **Commitee** c**:**
  - The dealer $P_d^c \in \mathcal{S}^c$ with input s samples a uniform random bivariate polynomial $F(x,y) = \sum_{k,\ell=0}^{t} f_{k\ell} x^k y^\ell$ of degree at most t in each variable such that $F(0,0) = s$, and for each $k, \ell \in [0,t]$ it invokes ShareFuture$(f_{k\ell})$ (independently) towards committees $\mathcal{S}^q$ for $q \in \{c_1, c_2, c_3\}$.
  - For all $i \in [1,n]$ party P_i^c samples a uniform random polynomial $f^{(i)}(y) = \sum_{k=0}^{t} f_k^{(i)} y^k$ of degree at most t, and for all $k \in [0,t]$ invokes ShareFuture$(f_k^{(i)})$ (independently) towards committees $\mathcal{S}^q$ for $q \in \{c_1, c_2, c_3, c'\}$[a].

- **Commitee** c_1**:** Let $a_{ij} := F(i,j) + f^{(i)}(j); \quad b_{ij} := F(i,j) + f^{(j)}(i); \quad c_{ij} := f^{(i)}(j) - f^{(j)}(i)$. For each $i, j \in [1,n]$ each party computes shares of a_{ij}, b_{ij}, and c_{ij} as follows:
  - $[a_{ij}] = \sum_{k,\ell=0}^{t} [f_{k\ell}]\, i^k j^\ell + \sum_{k=0}^{t} \left[f_k^{(i)}\right] j^k$
  - $[b_{ij}] = \sum_{k,\ell=0}^{t} [f_{k\ell}]\, i^k j^\ell + \sum_{k=0}^{t} \left[f_k^{(j)}\right] i^k$
  - $[c_{ij}] = \sum_{k=0}^{t} \left[f_k^{(i)}\right] j^k - \sum_{k=0}^{t} \left[f_k^{(j)}\right] i^k$

  and inputs them to respective $3n^2$ instances of RecFuture.

- **Commitee** c_2**:** Each party does the following:
  - Obtain as output of RecFuture values a_{ij}, b_{ij}, c_{ij}. If any instance outputs $\bot$, set the output to a default value[b];
  - If for some fixed $i \in [1,n]$ the values $\{a_{ij}\}_{j\in[1,n]}$ ($\{b_{ji}\}_{j\in[1,n]}$) do not lie on a polynomial of degree at most t, mark P_i as *unhappy*[c];
  - If $a_{ij} - b_{ij} \neq c_{ij}$ for some i, j, input the following shares

$$\left[f^{(i)}(j)\right] = \sum_{k=0}^{t} \left[f_k^{(i)}\right] j^k; \quad \left[f^{(j)}(i)\right] = \sum_{k=0}^{t} \left[f_k^{(j)}\right] i^k; \quad [F(i,j)] = \sum_{k,\ell=0}^{t} [f_{k\ell}]\, i^k j^\ell$$

  to respective instances of RecFuture. Let InconsPairs denote the set of such (i,j) pairs.

- **Committee** c_3**:** For each $(i,j) \in$ InconsPairs, each party does the following:
  - Obtain value $f^{(i)}(j)$ (respectively $f^{(j)}(i)$) as output of RecFuture, and if the output is $\bot$, mark P_i (respectively P_j) as *unhappy* and set the output to a default value;
  - Obtain value $F(i,j)$ as output of RecFuture, and if the output is $\bot$ disqualify the dealer;
  - Check if $a_{ij} - f^{(i)}(j) = F(i,j)$ (respectively $b_{ij} - f^{(j)}(i) = F(i,j)$). If not, mark P_i (respectively P_j) as *unhappy*;
  - Let EarlyUnhappy denote the set of indices of parties marked as *unhappy* so far. For all $i \in$ EarlyUnhappy input shares of $[F(i,j)]$, $[F(j,i)]$, and $[f^{(j)}(i)]$ for all $j \in [1,n]$ (respectively shares of $[F(j,i)]$, $[F(i,j)]$, and $[f^{(i)}(j)]$ for all $i \in [1,n]$) to respective instances of RecFuture;
  - If $|\text{EarlyUnhappy}| > t$ disqualify the dealer.

- **Committee** c'**:** For each $i \in$ EarlyUnhappy, each party does the following:

- Obtain values $F(i,j), F(j,i)$ for all $j \in [1,n]$ as outputs of respective instances of $\mathsf{RecFuture}$. If any output is $\perp$, or if the values do not define polynomials of degree at most t, disqualify the dealer.
- Obtain values $f^{(j)}(i)$ for all $j \in [1,n]$ as outputs from respective instances of $\mathsf{RecFuture}$. If any output is $\perp$, mark P_j as *unhappy*.
- Check that $a_{ji} - f^{(j)}(i) = F(j,i)$, and $b_{ij} - f^{(j)}(i) = F(i,j)$. If not, mark P_j as *unhappy*.
- Let Happy denote the set of indices of happy (not *unhappy*) parties. If $|\mathsf{Happy}| < n-t$ disqualify the dealer.

[a] We denote independent sharings of the same value a resulting from different invocations of $\mathsf{ShareFuture}\,(a)$ by $[a]$ to ease notation. As long as $c_1 < c_2 < c_3 < c'$ the committees need not be consecutive.

[b] As all information used for reconstruction is public, parties are in agreement on which reconstructions fail.

[c] This should be understood as party $P_i^c \in \mathcal{S}^c$ having misbehaved, or the dealer being corrupted.

Protocol $\mathsf{VSS.RecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n) \to s'$

- **Committee** c'**:** For each $i \notin \mathsf{EarlyUnhappy}$, each party inputs shares $\left[f_k^{(i)}\right]$ for all $k \in [0,t]$ to respective instances of $\mathsf{RecFuture}$[a].
- **Committee** $c'' \geq c'$**:** Each party does the following:
  - For all $i \notin \mathsf{EarlyUnhappy}$, obtain the coefficients of $f^{(i)}(x)$ from respective instances of $\mathsf{RecFuture}$. If no output is $\perp$, compute $F'(i,j) = a_{ij} - f^{(i)}(j)$ and $F'(j,i) = b_{ji} - f^{(i)}(j)$ for all $j \in [1,n]$. Check that these unblinded values define polynomials $h^{(i)}(y)$ and $g^{(i)}(x)$ of degree at most t.[b]. Let $\mathsf{Consistent}$ denote the set of indices of parties for which all these checks succeed;
  - For all $i \in \mathsf{EarlyUnhappy}$, consider the polynomials $g^{(i)}(x), h^{(i)}(y)$ identified by the values broadcast by committee c_3;
  - Construct an undirected graph G: vertices are all indices in $\mathsf{Consistent} \cup \mathsf{EarlyUnhappy}$, and there is an edge between i and j if and only if $g^{(i)}(j) = h^{(j)}(i)$ and $g^{(j)}(i) = h^{(i)}(j)$. Let G' be a $(n-t)$-clique in G.
  - Use any $t+1$ polynomials corresponding to indices in G' to interpolate a bivariate polynomial $F'(x,y)$ of degree at most t in each variable. Output $F'(0,0)$.

[a] If we replace $\mathsf{RecFuture}$ with its private version, we obtain a private version of $\mathsf{VSS.RecFuture}$ protocol.

[b] If both the dealer and P_i are honest, then $g^{(i)}(x) = F(x,i)$ and $h^{(i)}(y) = F(i,y)$.

We start by proving some useful lemmas for our perfectly secure VSS protocol.

Lemma 6. *Assume that at most $t < n/3$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If both the dealer P_d^c and party P_i^c are honest, at the end of protocol* $\mathsf{VSS.ShareFuture}$*, then $i \in \mathsf{Happy}$ so that $|\mathsf{Happy}| \geq n-t$.*

Proof. Assume that P_d^c is honest. Then, they compute valid sharings of each coefficient of F and execute protocols $\mathsf{ShareFuture}$ correctly. This means honest party's shares ($n - t \geq 2t+1$) of each coefficient of F lie on a polynomial of degree at most t (in each $\mathcal{S}^k$ for $k\{c', c_1, c_2, c_3\}$). Since P_i^c is honest, they compute valid sharings of each coefficient of $f^{(i)}(x)$ and execute protocols $\mathsf{ShareFuture}$ correctly. This means honest party's shares of each coefficient of $f^{(i)}(x)$ lie on a polynomial of degree at most t in each $\mathcal{S}^k$ for $k \in \{c_1, c_2, c_3, c'\}$. Therefore, each honest party's shares of a_{ij} and b_{ij} lie on polynomials of degree at most t, and also the values a_{ij} and b_{ij} for $j \in [1,n]$ lie on the sum of polynomials of degree at most t, so that party P_i is not marked as *unhappy* by committee c_2. If $a_{ij} - b_{ij} = c_{ij}$, then P_i is not marked as *unhappy* by committee c_3. Assume that $a_{ij} - b_{ij} \neq c_{ij}$. Then, since both P_d^c and P_i^c are honest, all executions of ShareFuture have been executed with the same input and correctly, and it holds that in committee c_3 shares of $f^{(i)}(j)$ broadcast by honest parties lie on polynomial of degree at most t, and also $a_{ij} - f^{(i)}(j) = F(i,j)$ and $b_{ji} - f^{(j)}(i) = F(j,i)$, so that party P_i is not marked as unhappy by committee c_3. To conclude the proof, notice that, since both P_d^c and P_i^c are honest, by the same argument about the correctness of the ShareFuture protocols used above, in committee c' for all $j \in$ EarlyUnhappy it holds that $a_{ij} - f^{(i)}(j) = F(i,j)$ and $b_{ij} - f^{(j)}(i) = F(j,i)$ so that at the end of the VSS.ShareFuture protocol party P_i is not marked as *unhappy*. □

Lemma 7. *Assume that at most $t < n/3$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If the dealer P_d^c is honest, then they are not disqualified at the end of protocol* VSS.ShareFuture.

Proof. Because P_d^c is honest, Lemma 6 guarantees that in committee c' it holds that $|\mathsf{Happy}| \geq n - t$. By inspection of the protocol, the cardinality of Happy can only decrease during the execution, so that at all points in the protocol $|\mathsf{Happy}| \geq n - t$, and the dealer is not disqualified because of the checks on the number of *unhappy* parties by committees c_3 and c'. Again, by inspection of the protocol, the only other possibility for the dealer to be disqualified is if reconstruction of values $F(i,j)$ for some $i, j \in [1, n]$ fails in committees c_3, c'. However, since P_d^c is honest, they compute valid sharings of the coefficients of F and execute protocols ShareFuture correctly, which results in each honest party in committees c_2, c_3 holding shares on polynomials of degree at most t. Therefore, at least $n - t$ of the braodcast shares for each value $F(i,j)$ by committees c_2, c_3 lie on a linear combination of polynomials of degree at most t, and reconstructions by committees c_3, c' succeeds. This concludes the proof. □

Lemma 8 (Commitment of VSS). *Assume that at most $t < n/3$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If the dealer P_d^c is not disqualified and P_i^c and P_j^c are honest, the polynomials associated with them at the end of protocol* VSS.RecFuture *are consistent. In particular, After* VSS.ShareFuture *terminates, the state of honest parties uniquely determines the output of* VSS.RecFuture.

Proof. At the end of protocol VSS.ShareFuture, for each honest P_i^c the index i is in one of the three disjoint sets Happy, EarlyUnhappy, or LateUnhappy $:= [1, n] \setminus (\mathsf{Happy} \cup \mathsf{EarlyUnhappy})$. We distinguish two cases:

1. If $i \in$ EarlyUnhappy and $j \in$ Happy, then this means that

$$h^{(j)}(i) := a_{ji} - f^{(j)}(i) = F(j,i) =: h^{(i)}(j)$$

$$g^{(j)}(i) := b_{ij} - f^{(j)}(i) = F(i,j) =: h^{(i)}(j),$$

   so that they will be associated to consistent polynomial at the end of protocol VSS.RecFuture.

2. If $i, j \in$ Happy $\cup$ LateUnhappy, then

$$g^{(i)}(j) := a_{ij} - f^{(i)}(j) = c_{ij} + b_{ij} - f^{(i)}(j) = b_{ij} - f^{(j)}(i) =: h^{(j)}(i)$$

$$g^{(j)}(i) := a_{ji} - f^{(j)}(i) = c_{ji} + b_{ji} - f^{(j)}(i) = b_{ji} - f^{(i)}(j) =: h^{(i)}(j)$$

   because the consistency checks carried out by committee c_3 succeeded, so that they will be associated to consistent polynomials at the end of protocol VSS.RecFuture.

There are at least $|\mathsf{Happy}| - t \geq t + 1$ indices in Happy at the end of the sharing phase. The associated consistent polynomials define a unique polynomial $F'(x, y)$ of degree at most t in each variable. This proves that the polynomials associated to the indices of honest parties in $\mathcal{S}^c$ all lie on the polynomial $F'(x, y)$. At reconstruction time, G' contains at least $n - 2t \geq t + 1$ indices corresponding to honest parties in $\mathcal{S}^c$, and the associated polynomials uniquely determine $F'(x, y)$, so that the output will be $F'(0, 0) = s'$. □

Lemma 9 (Robustness of VSS). *Assume that at most $t < n/3$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$ in* VSS.ShareFuture. *If the dealer P_d^c is honest and has input s, then the output of protocol* VSS.RecFuture *is s.*

Proof. Follows easily from Lemma 8. □

Lemma 10 (Linearity of VSS). *Assume that the dealer P_d^c is not disqualified in two executions* VSS.ShareFuture(x) *and* VSS.ShareFuture(y) *towards committee $\mathcal{S}^{c'}$. Then, committee $\mathcal{S}^{c'}$ holds sharings of $\mathcal{L}(x, y)$ for all linear functions $\mathcal{L} : \mathbb{F}^2 \to \mathbb{F}$, i.e. for all $\mathcal{L}$ there exists function $\bar{\mathcal{L}}$ (that can be computed efficiently from $\mathcal{L}$) such that* VSS.RecFuture $\left(\bar{\mathcal{L}}(\mathbf{x}_1, \mathbf{y}_1), \ldots, \bar{\mathcal{L}}(\mathbf{x}_n, \mathbf{y}_n)\right) = \mathcal{L}(x, y)$.

Proof. Each $P_i^{c'}$ simply applies $\mathcal{L}$ individually to all public values and all private shares resulting from executions VSS.ShareFuture(x) and VSS.ShareFuture(y). The claim easily follows. □

Lemma 11 (Privacy of VSS.ShareFuture). *If P_d^c is honest, then the view of an unbounded adversary who corrupts up to $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in [c, c']$ is identically distributed until the end of executions* VSS.ShareFuture(s) *and* VSS.ShareFuture(s')*, for any $s \neq s'$.*

Proof. From committee $\mathcal{S}^c$ the adversary learns up to t polynomials $f^{(i)}$ corresponding to corrupted parties. From committee $\mathcal{S}^{c_1}$, thanks to Lemma 5, the adversary only learns t shares of each coefficient of $f^{(i)}(x)$ for all honest parties P_i^c, as well as up to t shares of each coefficient of the polynomial $F(x, y)$. Since these polynomials are of degree at most t, the view of the adversary is independent of s so far. From committee $\mathcal{S}^{c_2}$ the adversary learns fresh shares of all the coefficients of the same polynomials (which are still statistically independent from s by the same argument as above), and also, thanks to the information collected from committee $\mathcal{S}^c$, evaluations $F(i, j)$ for all couples of indices (i, j) such that P_i^c and P_j^c are corrupted, as well as, for all $i \in [1, n]$, evaluations $f^{(i)}(j)$ for all j such that P_j^c is corrupted. Since there are at most t corrupted parties in $\mathcal{S}^c$, the adversary view is still independent of s. All values learned from the adversary from this point on, thanks to Lemma 5, are of two types 1) values already known to the adversary, or 2) fresh sharings of coefficients to polynomials F and $f^{(i)}$ for all $i \in [1, n]$. Indeed, because the dealer is honest, apart from the outputs of protocols ShareFuture, from committee $\mathcal{S}^{c_3}$ the adversary only learns $f^{(i)}(j)$, $f^{(j)}(i)$, and $F(i, j)$ if $a_{ij} - b_{ij} \neq c_{ij}$ if either P_i^c or P_j^c are dishonest. Assume that P_i^c is honest in this scenario. Then the adversary can compute $f^{(i)}(j)$ from already known values as $c_{ij} - f^{(j)}(i)$, and $F(i, j)$ as $b_{ji} - f^{(j)}(i)$. An identical argument applies to values learned by committee $\mathcal{S}^{c_4}$. The claim follows. □

4.4 Same Value, Many Sharings

Our protocol VSS.ShareFuture from the previous section can be used by a dealer $P_d^c \in \mathcal{S}^c$ to VSS a secret towards any committee $\mathcal{S}^{c'}$, as long as $c' \geq c + 4$. However, for our main MPC protocol, we need that the dealer P_d^c should be able to produce different sharings of *the same* value s towards two distinct committees

$\mathcal{S}^{c'}, \mathcal{S}^{c''}$. If P_d^c is honest, this can be achieved easily using two independent executions of $\mathsf{VSS.ShareFuture}$ towards $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$. However, we want to enforce that even if P_d^c is corrupted, committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ agree on whether the sharing succeeded, and the output of the protocol $\mathsf{VSS.RecFuture}$ will be the same regardless of which of the two committees run it.

This can be achieved by a minor modification to protocol $\mathsf{VSS.ShareFuture}$. Suppose $P_d^c \in \mathcal{S}^c$ wants to produce such *duplicated* sharings of s towards committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ (we can assume that $c'' > c'$). They invoke a *single* instance of protocol $\mathsf{VSS.ShareFuture}$ with a small modification:

- In the first step when each party P_i^c in committee c samples a blinding polynomial $f^{(i)}(x)$ and uses $\mathsf{ShareFuture}\left(f^{(i)}(x)\right)$ to share it towards committees $\mathcal{S}^k$ for $k \in \{c_1, c_2, c_3, c'\}$, it now also additionally shares it towards committee $S^{c''}$.

- The rest of the protocol proceeds as described in the previous section. Except that in the last step, when committee c' uses the shares of blinding polynomials received from committee c and the publicly broadcast information to locally determine their final shares, committee c'' also does the same. Since P_d^c can only get disqualified based on broadcast information, committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ agree on whether the protocol succeeded or not. And since both these committees received different sharings of the same blinding polynomials, the resulting sharings of the secret will be different. All other properties follow identically to the analysis in Section 4.3.

4.5 2-Level Verifiable Secret Sharing

The sharing state of each party $P_i^{c'}$ after protocol $\mathsf{VSS.ShareFuture}$ consists of many values as well as public information. In this section, we describe protocol $\mathsf{2VSS}$, that has two main goals:

1. *Simplify the sharing state for a value s:* each party $P_i^{c'} \in \mathcal{S}^k$ will hold value $s_i := f_s(i)$ where $f_s(x)$ is a polynomial of degree at most t such that $f_s(0) = s$ (this is essentially to get a simplified sharing from a somewhat ugly and long sharing state at the end of $\mathsf{VSS.ShareFuture}$);

2. *Strengthen the sharing state for a value s:* parties will hold sharings (meaning the state after protocol $\mathsf{VSS.ShareFuture}$) of *each coefficient* of the polynomial $f_s(x)$. By the homomorphic properties of $\mathsf{VSS.ShareFuture}$, this results in each $P_i^{c'}$ knowing a share of $f_s(j)$ for all $j \in [1, n]$.

We formalize these properties in Lemmas 12, 13, 14, 15.

Protocol Overview. The dealer P_d^c *chooses* a polynomial (the degree can be arbitrary, this is important for our use in Section 4.6) such that $f(0) = s$, and duplicates $\mathsf{VSS.ShareFuture}(f_i)$ towards committees $\mathcal{S}^{c_1}$, $\mathcal{S}^{c'}$ for each coefficient f_i of $f(x)$ for $i \in [0, t]$[16]. Then, each party in committee c_1 locally computes a share of $[f(i)]$[17] as $\sum_{k=0}^{t} [f_k] i^k$, and participates in protocol $\mathsf{VSS.RecFuture}(f(i))$ *privately* towards party $P_i^{c'}$. We will refer to the state achieved by this protocol as (t)-$\mathsf{2VSS.ShareFuture}(s)$, where $t = \deg f(x)$.

Similar to the previous section, we need *duplicate* sharings $\mathsf{2VSS.ShareFuture}$ of the *same* value s towards distinct committees $\mathcal{S}^{c'}, \mathcal{S}^{c''}$, such that that even if the dealer P_d^c is dishonest, both committees agree on whether the sharing procedure succeeded. To achieve this, the dealer P_d^c chooses polynomials $f(x)$ and $g(x)$ of the same degree such that $f(0) = s = f'(0)$. Then, they duplicate $\mathsf{VSS.ShareFuture}(f_0 =$

[16] It must holds that at least $c_1 \geq c + 4$.

[17] We overload the notation $[a]$ to also denote the state resulting from protocol $\mathsf{VSS.ShareFuture}(a)$ and $\mathsf{2VSS.ShareFuture}(a)$. The meaning is always clear from the context.

g_0) towards committees $\mathcal{S}^{c_1}, \mathcal{S}^{c'}$, and $\mathcal{S}^{c''}$ as described in Section 4.4. They duplicate sharings of all *non-constant* coefficients of $f(x)$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$ and duplicate sharings of each *non-constant* coefficient of $g(x)$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c''}$. We formally describe the (non-duplicated version of the) protocol below.

Protocol 2VSS.ShareFuture$(s) \to (\mathbf{s}_1, \dots, \mathbf{s}_n)$

- **Committee** c**:** The dealer P_d^c holding input s chooses a polynomial $f(x) = \sum_{k=0}^{\ell} f_k x^k$ such that $f(0) = s$. For all $k \in [0, \ell]$ they duplicate VSS.ShareFuture(f_k) towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$.
- **Committee** c_1**:** If the dealer is disqualified in any invocation of VSS.ShareFuture, disqualify the dealer. Otherwise, each $P_i^{c_1}$ computes the following for all $j \in [1, n]$: $[f(j)] = \sum_{k=0}^{\ell} [f_k] j^k$ and participates in VSS.RecFuture$(f(j))$ towards party $P_j^{c'}$.
- **Committee** c'**:** If the dealer is disqualified in any invocation of VSS.ShareFuture, disqualify the dealer.

Protocol 2VSS.RecFuture$(\mathbf{s_1}, \dots, \mathbf{s_n}) \to s$

Each Party $P_i^{c'}$ participates in VSS.RecFuture(f_0) towards committee $\mathcal{S}^{c''}$.

Lemma 12 (Linearity of 2VSS). *Assume that the dealer P_d^c is not disqualified in two executions* 2VSS.ShareFuture(x) *and* 2VSS.ShareFuture(y) *towards committee $\mathcal{S}^{c'}$. Then, committee $\mathcal{S}^{c'}$ holds sharings of $\mathcal{L}(x, y)$ for all linear functions $\mathcal{L} : \mathbb{F}^2 \to \mathbb{F}$, i.e. for all $\mathcal{L}$ there exists function $\bar{\mathcal{L}}$ (that can be computed efficiently from $\mathcal{L}$) such that* 2VSS.RecFuture $\left(\bar{\mathcal{L}}(\mathbf{x}_1, \mathbf{y}_1), \dots, \bar{\mathcal{L}}(\mathbf{x}_n, \mathbf{y}_n)\right) = \mathcal{L}(x, y)$.

Proof. Follows easily from Lemma 8. □

Lemma 13 (Commitment of 2VSS). *Assume that up to $t < n/3$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If the dealer P_d^c is not disqualified, then after* 2VSS.ShareFuture *terminates there exists a degree-t polynomial $f_s(x)$, such that each honest party $P_i^{c'}$ knows value $f_s(i)$ and also an honest VSS sharing (in the sense of the state resulting from protocol* VSS.ShareFuture*) of each coefficient of the polynomial $f_s(x)$.*

Proof. The dealer simply uses duplicate VSS.ShareFuture to duplicate (see Section 4.4) shares of each of the $t + 1$ coefficients of polynomial $f_s(x)$ to committees c_1 and c'. From commitment property of VSS.ShareFuture (see Lemma 8), it follows that if the dealer is not disqualified then committees c_1 and c' have consistent shares of the same set of $t + 1$ coefficients. Now as long as the honest parties in committee c_1 computes shares of evaluations of the polynomial $f_s(x)$ using these shares of the coefficients, then committee c' will also have Shamir shares of the secret associated with $f_s(x)$. □

Lemma 14 (Robustness of 2VSS). *Assume that up to $t < n/3$ parties are corrupted in each committee in* 2VSS.ShareFuture*. If the dealer P_d^c is honest and has input s, then the output of protocol* 2VSS.RecFuture *is s.*

Proof. Follows easily from Lemma 13. □

Lemma 15 (Privacy of 2VSS.ShareFuture). *Let P_d^c be an honest party. If the degree of the polynomial $f(x)$ chosen is at most ℓ, for some $\ell \geq t$, then the view of an unbounded adversary who corrupts $t < n/3$ parties in each committee, is identically distributed until the end of executions* 2VSS.ShareFuture(s) *and* 2VSS.ShareFuture(s')*, for any $s \neq s'$.*

Proof. Privacy follows from privacy of $\mathsf{VSS.ShareFuture}$ (see Lemma 11) for each coefficient of the polynomial $f(x)$, and because the adversary, from committee $\mathcal{S}^{c'}$ learns at most t shares of $f(x)$, that has degree at most $k \geq t$. □

4.6 Perfectly Secure BGW-Style Maximally-Fluid Multiplication Proof

In this section, we describe a protocol between a (prover) party $P_p^c \in \mathcal{S}^c$, who knows values x, y, z, and committees $\mathcal{S}^{c_1}, \mathcal{S}^{c_2}, \mathcal{S}^{c'}$.

- Committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c_2}$ hold duplicated states (t)-$\mathsf{2VSS.ShareFuture}(x)$ and (t)-$\mathsf{2VSS.ShareFuture}(y)$;
- Committee $\mathcal{S}^{c'}$ holds state (t)-$\mathsf{2VSS.ShareFuture}(z)$.

Let $f_x(x), f_y(x), f_z(x)$ denote the polynomials associated to these states.

- For each polynomial, committees $\mathcal{S}^{c_1}$, $\mathcal{S}^{c_2}$, and $\mathcal{S}^{c'}$ hold duplicated $\mathsf{VSS.ShareFuture}$ sharings of each coefficient;
- Party P_p^c knows $f_x(x)$, $f_y(x)$ and $f_z(x)$.

The goal of the protocol is for party P_p^c is to prove to parties in $\mathcal{S}^{c'}$ that $z = x \cdot y$. We describe the distributed proof below and formalize its properties in Lemmas 16 and 17.

Protocol $\mathsf{MultProof}(x, y, z; f_x(x), f_y(x), f_z(x))$

- **Committee** c**:**
  - Player P_p^c computes polynomial[a] $f_w(x) := f_x(x) \cdot f_y(x) = \sum_{k=0}^{2t} f_{w,k} x^k$.
  - Player P_p^c invokes $(2t)$-$\mathsf{2VSS.ShareFuture}(z)$ towards $\mathcal{S}^{c_1}$ using polynomial $f_w(x)$ and duplicates $\mathsf{VSS.ShareFuture}(f_{w,k})$ for all $k \in [0, 2t]$ towards committee $\mathcal{S}^{c_2}$ *but* because $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c_2}$ already hold state $\mathsf{VSS}(f_{z,0} = z)$ this is taken as state $\mathsf{VSS}(f_{w,0} = f_{z,0} = z)$. This results in parties in $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c_2}$ knowing shares $[f_{w,k}]$ for all $k \in [1, 2t]$, as well as each $P_i^{c_1}$ knowing $w_i := f_w(i)$.
- **Committee** c_1**:** If party P_p^c is disqualified in any invocation of $\mathsf{VSS.ShareFuture}$, the proof *fails.* Otherwise, each player $P_i^{c_1}$ checks if $x_i \cdot y_i = z_i$. If not, they broadcast $\mathsf{Complain}$.
- **Committee** c_2**:** If more than t distinct parties in $\mathcal{S}^{c_1}$ broadcast $\mathsf{Complain}$, the proof *fails.* Otherwise, for each party $P_i^{c_1}$ who broadcast $\mathsf{Complain}$, each party in $\mathcal{S}^{c_2}$ inputs $[x_i], [y_i], [w_i]$ to respective instances of $\mathsf{VSS.RecFuture}$.
- **Committee** c'**:** For each party $P_i^{c_1}$ who broadcast $\mathsf{Complain}$, obtain values x_i, y_i, w_i as outputs of respective instances of $\mathsf{VSS.RecFuture}$. If $x_i \cdot y_i \neq w_i$ for any i the proof *fails.* Otherwise, committee $\mathcal{S}^{c'}$ accepts the proof.

[a] It holds that $\deg f_w(x) \leq 2t$ and $f_{w,0} = f_w(0) = x \cdot y$

Lemma 16 (Soundness and Completeness of $\mathsf{MultProof}$)**.** *Assume that at most $t < n/3$ parties in each committee $\mathcal{S}^k$ for $k \in \{c_1, c_2, c'\}$ are corrupted. If $\mathcal{S}^{c'}$ accepts the proof, they indeed hold state (t)-$\mathsf{2VSS.ShareFuture}(z)$, for $z = x \cdot y$. Furthermore, if the dealer is honest, the proof succeeds.*

Proof. Each party $P_i^{c'}$ holds value $z_i = f_{z'}(i)$ for some polynomial $f_{z'}(x)$ such that $f_{z'}(0) = f_w(0)$. This is because state $\mathsf{VSS.ShareFuture}(f_{z',0})$ is the same as state $\mathsf{VSS.ShareFuture}(f_{w,0})$. We must show that indeed $f_w(x) = f_x(x) \cdot f_y(x)$; from this it follows that $f_{z'}(0) = f_w(0) = f_x(0) \cdot f_y(0) = x \cdot y$. Since the proof succeeds, committee $\mathcal{S}^{c_1}$ broadcasts at most $k \leq t$ times Complain. This means that, for at least $n - k - t$ honest parties $P_i^{c_1}$ that did not complain it holds that $x_i \cdot y_i = w_i$. If the proof succeeds, this means that for each Complaint raised by some $P_i^{c_1}$, parties in $\mathcal{S}^{c_2}$ verify that $x_i \cdot y_i = w_i$. Hence for $(n-k-t)+k = n-t \geq 2t+1$ indices i, it holds that $f_w(i) = w_i = x_i \cdot y_i = f_x(i) \cdot f_w(i)$. Therefore, the polynomials $f_w(x)$ and $f_x(x) \cdot f_y(x)$ coincide in at least $2t+1$ points, but their degree is at most $2t$, which means they are identical. It is straightforward to check that if the dealer is honest, the proof succeeds. □

Lemma 17 (Privacy of MultProof)**.** *Assume that the prover P_p^c is honest and the* 2VSS.ShareFuture *states for x, y and z were generated by to honest dealers. Then, the view of any adversary who corrupts $t < n/3$ parties in each committee $\mathcal{S}^k$ for all $k \in [c, c']$ is identically distributed in executions of* MultProof *with inputs $(x, y, z; f_x(x), f_y(x), f_z(x))$ and $(x', y', z'; f_{x'}(x), f_{y'}(x), f_{z'}(x))$ for $(x, y, z) \neq (x', y', z')$.*

Proof. The only values the adversary learns are in committee $\mathcal{S}^{c_2}$ the values x_i, y_i, and z_i for $i \in [1, n]$ such that party $P_i^{c_1}$ broadcasts Complain. However, since the dealer is honest, no honest player $P_i^{c_1}$ complains, so that these values are already known to the adversary. The claim then follows from this together with Lemma 11 and Lemma 15. □

4.7 Perfectly Secure Maximally-Fluid MPC

With the tools developed in the previous sections, we now present our maximally fluid and perfectly-secure MPC protocol with guaranteed output delivery. Without loss of generality, we can assume that the function to be computed is encoded as a layered arithmetic circuit Circ over a finite field $\mathbb{F}$ with fan-in 2. There are 4 types of gates: input, addition, multiplication, and output gates. For each layer L of Circ, the following invariant is preserved: there is a committee $\mathcal{S}^c$ holding states (t)-2VSS.ShareFuture(x) and (t)-2VSS.ShareFuture(y) for input-wire values x and y of each gate $g \in L$, and a committee $\mathcal{S}^{c'}$ holding state (t)-2VSS.ShareFuture(z) for each output-wire value z of each $g \in L$. This allows for a layer-by-layer computation of the whole circuit. We describe the protocol below.

Protocol CircEval

Input.

This gate has 1 input wire with value x and 1 output wire with value x, and a client associated to it. A client holds input x, and invokes 2VSS.ShareFuture(x) towards a future committee $\mathcal{S}^{c'}$ (all clients towards the same committee);

Output.

This gate has 1 input wire and 1 output wire, and a client associated to it. Let y denote the input wire value. Each honest party P_i^c knows $y_i = f_y(i)$ for some polynomial $f_y(x)$ of degree at most t such that $f_y(0) = y$. Each P_i^c participates in 2VSS.RecFuture(y) towards the client.

Addition.

This gate has 2 input wires with values x, y and 1 output wire with value z. Without loss of generality, we can assume that committee $\mathcal{S}^{c'}$ already holds duplicated states (t)-2VSS.ShareFuture(x) and (t)-2VSS.ShareFuture(y). Committee $\mathcal{S}^{c'}$ computes (t)-2VSS.ShareFuture(z) locally by exploiting the linearity of 2VSS.ShareFuture.

Multiplication.

This gate has 2 input wires with values x, y and 1 output wire with value z. Committee $\mathcal{S}^c$ holds states (t)-2VSS.ShareFuture(x) and (t)-2VSS.ShareFuture(y)[a]. Each party P_i^c does the following:

- Compute $z_i = x_i \cdot y_i$;
- Invoke (t)-2VSS.ShareFuture(x_i) and (t)-2VSS.ShareFuture(y_i) towards committee $\mathcal{S}^{c_1}$, *but* taking the already known states VSS.ShareFuture(x_i) and VSS.ShareFuture(y_i) as sharings of the constant coefficients of the polynomials used, which we denote by $f_{x_i}(x), f_{y_i}(x)$ respectively.
- Duplicate 2VSS.ShareFuture(z_i) towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$. Let $f_{z_i}(x)$ denote the polynomial associated to the state of committee $\mathcal{S}^{c_1}$.
- Run protocol MultProof$(x_i, y_i, z_i; f_{x_i}(x), f_{y_i}(x), f_{z_i}(x))$ towards committee $\mathcal{S}^{c_1}$.[b]

Let Succeed $:= \{i \in [1, n] \mid$ MultProof(x_i, y_i, z_i) succeeds$\}$. Parties in committee $\mathcal{S}^{c_1}$ do the following: for all $i \notin$ Succeed, participate in VSS.RecFuture(x_i) and VSS.RecFuture(y_i).
Each party $P_j^{c'} \in \mathcal{S}^{c'}$ does: for all $i \notin$ Succeed compute a standard (for example, with $t = 0$) state (t)-2VSS.ShareFuture(z_i) and then compute state (t)-2VSS.ShareFuture(z) as

$$\mathcal{L}\Big((t)\text{-2VSS.ShareFuture}(z_1), \ldots, (t)\text{-2VSS.ShareFuture}(z_n)\Big)$$

by exploiting the linearity of 2VSS.ShareFuture.

[a] We can assume that these states are duplicated towards committee $\mathcal{S}^{c_1}$ as well without loss of generality

[b] For the sake of clarity of exposition, we abstract away the duplicated states for the auxiliary committees of the proof.

Theorem 1. *Protocol* CircEval *is a maximally-fluid MPC for all polynomial functions, that achieves* $1/3$-*perfect security and guaranteed output delivery against unbounded adversaries.*

Proof. We describe a simulator Sim for protocol CircEval. Recall that the protocol execution proceeds layer by layer, where at each layer of (computing) gates $g_1, \ldots, g_w$, there will be a designated committee $\mathcal{S}^{\mathsf{in}}$ (that becomes known only when the immediate previous committee is active), holding the share state of an instance of VSS.ShareFuture on each of the input wires to each gate g_i, $1 \leq i \leq w$, and after the gates are processed, there will be another future designated committee $\mathcal{S}^{\mathsf{out}}$ (also known only when the immediate previous committee is active) holding the share state of an instance of VSS.ShareFuture on each of the corresponding output wires. For an input gate, there will be a designated client C with an input wire value associated to it and a committee $\mathcal{S}^{\mathsf{out}}$ will hold the share state of VSS.ShareFuture on this value. For an output gate, a $\mathcal{S}^{\mathsf{in}}$ holds the share state of VSS.ShareFuture on a value y, and a designated client obtains the output y.

We describe the simulator Sim as being processed layer by layer. Let $\mathcal{S}^{\mathsf{in}}$ and $\mathcal{S}^{\mathsf{out}}$ be the corresponding committees holding states of the input wires and that will hold the states of the output wires. We process each gate of the layer independently as follows.

- *Input Gates.* Let C denote the client associated to the input gate.

  The simulator does the following. If C is corrupted, the simulator participates in the protocol VSS.ShareFuture on behalf of all honest parties in the committees that are involved for this protocol. If C is not disqualified at the end of the protocol, by Lemma 13, each honest party P_i^{out} holds a share of a degree-t polynomial $f(\cdot)$. Given that there are at least $t+1$ honest parties in $\mathcal{S}^{\mathsf{out}}$, the simulator can compute $x = f(0)$, which is assigned to be the input of C, and sends it to the trusted party.

If C is honest, the simulator participates in an instance of 2VSS.ShareFuture towards committee $\mathcal{S}^{\mathsf{out}}$, emulating all honest parties in the respective participating committees, and where C has input 0.

- *Addition Gates.* Let x and y be the values of the input wires. We note that addition gates are processed locally in the protocol by committee $\mathcal{S}^{\mathsf{out}}$ (using linearity of VSS.ShareFuture), since we can assume without loss of generality that $\mathcal{S}^{\mathsf{out}}$ is the one holding (t)-2VSS(x) and (t)-2VSS(y). In the simulation, therefore, we also let $\mathcal{S}^{\mathsf{out}}$ perform this locally.
- *Multiplication Gates.* Let x and y be the values of the input wires. The simulator holds, for each honest party P_i^{in} the corresponding share state of (t)-2VSS(x) and (t)-2VSS(y), and can execute exactly the same steps as in the protocol on behalf of the honest parties in this committee and the intermediate committees up to $\mathcal{S}^{\mathsf{out}}$.
- *Output Gates.* Let C denote the client associated to the output gate. Let x denote the value of the input wire. If C is honest, the simulator emulates an instance of the protocol 2VSS.RecFuture, internally emulating C and all honest parties (from $\mathcal{S}^{\mathsf{in}}$ and other auxiliary committees) involved in the protocol. Otherwise, if C is corrupted, the simulator computes the share state of (t)-2VSS(x) that belongs to the corrupted parties in $\mathcal{S}^{\mathsf{in}}$, obtains the corresponding output value y from the trusted party, and computes updated states of the honest parties consistent with (t)-2VSS(y). Note that this is possible because of secrecy of VSS.ShareFuture, so the state of corrupted parties is statistically independent of x. And participates in an instance of 2VSS.RecFuture with this newly computed state on behalf of the honest parties in $\mathcal{S}^{\mathsf{in}}$ and other auxiliary committees.

First, note that by the commitment property of 2VSS, (see Lemma 13), the extracted inputs from corrupted clients correspond to the values that are processed in the real world. Moreover, by robustness and linearity of 2VSS (see Lemmas 12 and 14), and correctness and soundness of multiplication proofs (see Lemma 16), the values of the output wires are correct in the real world, and the final outputs of honest parties correspond to the outputs computed by the trusted party.

In addition, we argue that the view of the adversary is indistinguishable in both worlds. By secrecy of 2VSS and MultProof (see Lemmas 15 and 17), the states of dishonest parties are completely independent of the inputs from honest clients. Moreover, since each honest party succeeds in the multiplication proof (see Lemma 16), up to processing the output gates, the view of the adversary is indistinguishable in both worlds.

Finally, we argue that the view of the adversary is also distributed identically in both worlds when executing each output gate. Let x be the value of the input wire of the output gate in the ideal world, and let y be the value in the real world. In the case of an honest client C, the view of the adversary contains only contains t shares, which are independent of any honest value. Moreover, if the client C is corrupted, the computed states of honest parties are made in such a way that the state (t)-2VSS(y) encodes the value y (in the real world). Therefore, both worlds are distributed identically. □

Extending to Security against R-adaptive **Adversaries.** While in the above proof, for simplicity, we implicitly assume that the adversary is NR-adaptive, we now briefly discuss why we expect our protocol to also be secure against R-adaptive adversaries. Recall that an adversary is allowed to adaptively corrupt a party in any given committee $\mathcal{S}^{\ell}$ with retroactive effect, only if for each committee $\mathcal{S}^{\ell'}$, (where $\ell' < \ell$) in which this party participated in the past, the adversary had not already exhausted its corruption budget, thereby ensuring that the number of corrupt parties in every committee never exceeds $t < n/3$.

Such adversaries can easily be handled by our simulation. For each committee assigned to handle a computation gate (addition or multiplication), we can do as follows: Whenever a party $P_i^c \in \mathcal{S}^c$ from a past committee is corrupted, the simulator can simply reveal to the adversary the emulated state on behalf of P_i^c that it has so far, in an execution where the honest clients have input 0. Since the adversary holds only up to t states from each committee, secrecy of our VSS scheme and MultProof (see Lemmas 15 and 17), ensures that the view of the adversary is independent of any values from honest parties. If the committee is executing an output gate, the simulator needs to instead output the stated that was patched consistently with the output of that gate.

5 Computationally Secure Maximally-Fluid MPC with Guaranteed Output Delivery

In this section we present our computationally-secure maximally fluid MPC with guaranteed output delivery, secure up to $t < n/2$ corruptions in each committee. The protocol makes use of non-interactive equivocal linearly-homomorphic commitments 4 (see Definition 4).

5.1 (Computationally Secure) Channels To Future Committees

We present a maximally fluid protocol CompSendFuture that allows a sender party $P_s^c \in \mathcal{S}^c$ to send a message to a recipient party $P_r^{c'} \in \mathcal{S}^{c'}$, in such a way that a (computationally bounded) adversary $\mathcal{A}$ controlling at most $t < n/2$ servers in each committee $\mathcal{S}^k$, for $k \in [c, c']$ cannot tamper with, or learn any information about the message m. One can think of this protocol as constructing a one-directional secure channel between P_s^c and $P_r^{c'}$ from one-directional secure channels between all parties in consecutive committees.

The intuition is as follows: if $c' = c + 1$, then P_s^c can simply send the message to $P_r^{c'}$. If instead the sender wants to send a message *further* into the future, i.e. if $c' > c+1$, then P_r^c samples a polynomial $f(x)$ such that $\deg f(x) \leq t$ and $f(0) = m$, they broadcast commitments to the coefficients of $f(x)$, and they send $m_i = f(i)$ together with the respective opening information to party P_i^{c+1}. This results in publicly known commitments to each m_i. If $c' = c + 2$, now each party P_i^{c+1} opens the commitment to value m_i to $P_r^{c'}$. If the dealer is honest, $P_r^{c'}$ receives at least $n - t \geq t + 1$ valid openings and reconstruct m using Lagrange interpolation. If $c' > c + 2$, each party P_i^{c+1} acts as the sender in the protocol we just described, with input m_i *but* using the already publicly known commitment to m_i as the commitment to the constant term of the polynomial they use to share m_i. In the end, party $P_i^{c'-1}$ holds a $(c'-c-1)$-dimensional tensor of values and the respective opening information; they open all these values to $P_r^{c'}$. Then, $P_r^{c'}$ reconstructs valid openings using Lagrange interpolation layer-by-layer. If P_s^c is honest, in each reconstruction layer there will be at least $t + 1$ valid openings and the last step of this process outputs m. We summarize and prove the properties of this protocol in Lemmas 18 and 19 below.

Protocol CompSendFuture(m)

Party P_s^c is the sender, with input m, and party $P_r^{c'}$, for some $c' > c$ is the receiver. For all $i \in [1, n]$ party P_i^c initializes $\mathbf{m}_i := \perp$ and $\mathbf{r}_i := \perp^a$.

- **Committee** c**:** The sender P_s^c samples a uniform random polynomial $f_m(x) = \sum_{k=0}^{t} f_{m,k} x^k$ such that $f_m(0) = f_{m,0} = m$ and uniform random $r_{m,0}, \ldots, r_{m,t}$, and compute $\mathsf{com}_{m,k} := \mathsf{Commit}(f_{m,k}, r_{m,k})$ for all

$k \in [0,t]$. Broadcast $\mathsf{com}_{m,k}$ for all $k \in [0,t]$ and send

$$\left(f_m(i), \sum_{k=0}^{t} r_{m,k} \cdot i^k\right)$$

to party P_i^{c+1}.

- **Committees** $k \in [c+1, c'-1]$**:** Each party P_j^k sets $\mathbf{m}_i := (\mathbf{m}_{1i}, \ldots, \mathbf{m}_{ni})$ and $\mathbf{r}_i := (\mathbf{r}_{1i}, \ldots, \mathbf{r}_{ni})$, where $\mathbf{m}_{ji}$ and $\mathbf{r}_{ji}$ denote the first and second component of message received from P_j^{k-1}. Then
  - If $k = c'-1$, send $(\mathbf{m}_i, \mathbf{r}_i)$ to $P_r^{c'}$;
  - If $k \neq c'-1$, let Index denote the set of indices of $\mathbf{m}_i$. For each $\ell \in \mathsf{Index}$ do:
    - Sample a uniform random polynomial $f_\ell(x)$ such that $\deg f_\ell(x) \leq t$ and $f_\ell(0) = (\mathbf{m}_i)_\ell$;
    - Sample uniform random values $r_{\ell,1}, \ldots, r_{\ell,t}$ and compute commitments $\mathsf{com}_{\ell,k} := \mathsf{Commit}(f_{\ell,k}, r_{\ell,k})$ for all $k \in [1,t]$;
    - Broadcast $\mathsf{com}_{\ell,k}$ for $k \in [1,t]$ and send

$$\left(f_\ell(j), (\mathbf{r}_i)_\ell + \sum_{k=1}^{t} r_{\ell,k}\right)_{\ell \in \mathsf{Index}}$$

    to P_j^{k+1} for all $j \in [1,n]$.

- **Committee** c'**:** Upon receiving message $(\mathbf{m}_i, \mathbf{r}_i)$ from $P_i^{c'-1}$ for all $i \in [1,n]$ party $P_r^{c'}$ does:
  - Let Index denote the set of indices of the tensor $\mathbf{m}_i$. For all $\ell \in \mathsf{Index}$ check against the (appropriate linear combination of the) broadcast commitments that $(\mathbf{r}_i)_\ell$ is a valid opening for $(\mathbf{m}_i)_\ell$. Denote by Valid the subset of Index for which this check succeeds;
  - Use Lagrange interpolation layer-by-layer on values in $(\mathbf{m}_i)_{\ell \in \mathsf{Valid}}$ to reconstruct a value m_i, where $m_i = \perp$ if the last interpolation fails.

Repeat the procedure on vector $(m_1, \ldots, m_n)$ to recover a value m', where $m' = \perp$ if the last interpolation fails.

[a] Intuitively, $\mathbf{m}_i$ contains values, and $\mathbf{r}_i$ the respective opening information.

Lemma 18 (Robustness of $\mathsf{CompSendFuture}$)**.** *Assume P_s^c is honest with input m. For any PPT adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c'-1]$ in* $\mathsf{CompSendFuture}$ *the output of $P_r^{c'}$ is $m' = m$.*

Proof. We prove in the statement for $c' = c+2$. It is easy to generalize the argument to the case of more intermediate committees via a simple recursive argument. Since the sender P_s^c is honest, they use a degree t polynomial $f(x)$ such that $f(0) = m$ to secret share m towards committee $\mathcal{S}^{c+1}$. Furthermore, they publish valid commitments to each coefficient f_k of $f(x)$, and send the relative opening information to each party in committee $\mathcal{S}^{c+1}$. Consider now the receiver $P_r^{c'}$, and assume that $P_r^{c'}$ adds index i to set Valid. If (m_i, r_i) denotes the message they received from party P_i^{c+1}, then $\mathsf{Open}(\sum_{k=0}^{t} \mathsf{com}_k i^k, m_i, r_i) = 1$. For each set of $t+1$ indices in Valid, by linearity of the commitments, by using Lagrange interpolation it is possible to compute a valid opening to com_0. Furthermore, notice that for each honest P_i^{c+1}, we have $i \in \mathsf{Valid}$, so that $|\mathsf{Valid}| \geq n - t \geq t+1$, since $t < n/2$. Therefore, for $i \in \mathsf{Valid}$ values m_i lie on the same polynomial $f'(x)$ of degree at most t. Since the values m_i sent by honest parties lie on $f(x)$, no matter what subset of Valid the receiver uses to interpolate $f'(x)$, it holds that $f'(x) = f(x)$ so that the output of $P_r^{c'}$ is m. □

Lemma 19 (Privacy of $\mathsf{CompSendFuture}$)**.** *Let P_s^c be an honest party. Then the view of a PPT adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$ is identically distributed during executions* $\mathsf{CompSendFuture}(s)$ *and* $\mathsf{CompSendFuture}(s')$ *for any $s \neq s'$.*

Proof. All commitments are perfectly hiding. The polynomial $f(x)$ sampled by the sender is of degree at most t, and $\mathcal{A}$ only receives t evaluations of this polynomial, so that their view after committee $\mathcal{S}^{c+1}$ is independent of m. Moreover, their view in each subsequent committee is independent of the joint state of honest parties in each committee. The claim easily follows.

□

5.2 (Computationally Secure) Shamir Sharing to Future Committees

Building on $\mathsf{CompSendFuture}$ we describe protocols $\mathsf{CompShareFuture}$ and $\mathsf{CompRecFuture}$, that allow a dealer P_d^c to commit to a value s towards a committee $\mathcal{S}^{c'}$ for $c' > c$. It is guaranteed that even if P_d^c is corrupted, after protocol $\mathsf{CompShareFuture}$ terminates there is a unique $s' \in \mathbb{F} \cup \{\perp\}$ such that the output of $\mathsf{CompRecFuture}$ is either s' or $\perp$ (this is determined by $\mathcal{A}$ at opening time).

The idea is as follows: the dealer P_d^c samples a uniform random polynomial $f(x)$ with $\deg(f) \leq t$ and $f(0) = s$. For $i \in [0, t]$ let $\mathsf{com}_i := \mathsf{Commit}(f_i, r_i)$, where f_i denotes the i-th coefficient of $f(x)$. By the homomorphic properties of the commitment scheme, this results in commitments to any linear combination of the coefficients. Then P_d^c broadcasts com_i for all $i \in [0, t]$ and invokes $\mathsf{CompSendFuture}\left(s_i := f(i)\right)$ and $\mathsf{CompSendFuture}\left(r(i)\right)$ towards party $P_i^{c'}$, where $r(i) := \sum_{k=0}^{t} r_i \cdot i^k$

Protocol $\mathsf{CompRecFuture}$ then works as follows: each $P_i^{c'}$ in committee $\mathcal{S}^{c'}$ broadcasts $\left(f(i), r(i)\right)$. Let

$$\mathsf{Valid} := \left\{ i \in [1, n] \;\middle|\; \mathsf{Open}\left(\sum_{k=0}^{t} \mathsf{com}_k i^k, f(i), r(i)\right) = 1 \right\}$$

denote the set of indices of parties that broadcast valid openings of their shares. If $|\mathsf{Valid}| \geq t+1$, each party $P_i^{c''}$ interpolates $f(x)$ and $r(x)$ using any $t+1$ shares $f(i)$ and $r(i)$ with $i \in \mathsf{Valid}$ and outputs $\left(f(0), r(0)\right)$. Otherwise, the output is $\perp$.

It will be useful that P_d^c can commit *to the same value* towards separate committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$, so that even if P_d^c is corrupted there exists a unique value $s' \in \mathbb{F} \cup \{\perp\}$ such that $\mathsf{CompRecFuture}$ produces output s' or $\perp$ (determined by $\mathcal{A}$ at reconstruction time) *regardless* of which committee executes it. To achieve this, we require that P_d^c executes $\mathsf{CompShareFuture}$ towards committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{c''}$ with polynomials $f(x)$, $r(x)$ and $f'(x)$, $r'(x)$ respectively, such that $f(0) = s = f'(0)$ and $r(0) = r'(0)$, *but* with respect to the *same* commitment $\mathsf{com}_0 := \mathsf{Commit}(f_0, r_0)$. We provide a formal description of the (non-duplicated version of the) protocol below. The properties of the protocol are summarized and proven in Lemmas 20, 21, and 22.

Protocol $\mathsf{CompShareFuture}(s) \rightarrow (\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathsf{com}_0, \ldots, \mathsf{com}_t)$

- **Committee** c**:** The dealer P_d^c does:

- Sample a uniform random polynomials

$$f(x) := \sum_{k=0}^{t} f_k x^k, \quad r(x) := \sum_{k=0}^{t} r_k x^k$$

such that $f(0) = s$.

- For $i \in [0, t]$ let $\mathsf{com}_i := \mathsf{Commit}(f_i, r_i)$;
- Broadcast com_i for all $i \in [0, t]$;
- Invoke $\mathsf{CompSendFuture}\,(s_i := f(i))$, $\mathsf{CompSendFuture}\,(r(i))$ towards party $P_i^{c'}$ for all $i \in [1, n]$.

- **Committee** c'**:** Party $P_i^{c'}$ sets their output $\mathbf{s}_i$ to the outputs $f'(i)$ and $r'(i)$ of the respective invocations of CompSendFuture. The public outputs are com_k for $k \in [0, t]$.

Protocol $\mathsf{CompRecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathsf{com}_0, \ldots, \mathsf{com}_t) \to (s)$

- **Committee** c'**:** Each party $P_i^{c'}$ broadcasts[a] $\mathbf{s}_i$.
- **Committee** c''**:** Each $P_i^{c'}$ does
  - Let

$$\mathsf{Valid} := \left\{ i \in [1, n] \;\middle|\; \mathsf{Open}\left(\sum_{k=0}^{t} \mathsf{com}_k i^k, f'(i), r'(i) \right) = 1 \right\}.$$

  - If $|\mathsf{Valid}| > t$ interpolate a polynomial $f'(x)$ of degree at most t using any $t+1$ shares $f'(i)$ with $i \in \mathsf{Valid}$ and output $f'(0)$;
  - If $|\mathsf{Valid}| \le t$ output $\perp$.

[a] By replacing broadcast with CompSendFuture we obtain a private version of protocol CompRecFuture towards any committee $\mathcal{S}^k$ for $k \ge c''$.

Lemma 20 (Privacy of CompShareFuture)**.** *If P_d^c is an honest party, then the view of an adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$ is identically distributed during executions* $\mathsf{CompShareFuture}(s)$ *and* $\mathsf{CompShareFuture}(s')$*, for any $s \ne s'$.*

Proof. All commitments broadcast from P_d^c are perfectly hiding. Since the adversary corrupts at most $t < n/2$ in each committee $\mathcal{S}^k$ for all $k \in [c+1, c'-1]$, by Lemma 19, the view of the adversary in each execution of CompSendFuture is independent of the input by P_d^c. From committee $\mathcal{S}^{c'}$, the adversary learns t evaluations on the polynomial $f(x)$ of degree at most t, so that their view remains independent of $m = f(0)$. □

Lemma 21 (Honest Extractability/Robustness of CompShareFuture)**.** *Assume that at most $t < n/2$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in \{c, c_1, c'\}$. If P_d^c is honest with input s, then after* CompShareFuture *terminates, it is possible to efficiently compute values s and r from public information and the states of honest parties. Furthermore* $\mathsf{Open}\,(\mathsf{com}_0, s, r) = 1$ *and*

$$s := \mathsf{CompRecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathsf{com}_0, \ldots, \mathsf{com}_t).$$

Proof. Let $\mathcal{L}$ denote the linear Lagrange interpolation function corresponding to a subset $\mathcal{J} \subseteq \mathsf{Valid}$ s.t. $|\mathcal{J}| = t+1$. Notice that we have have $\mathcal{L}((\mathsf{com}_j)_{j\in\mathcal{J}}) = \mathsf{com}_0$. Then by linearity of commitments and their binding property we have $\mathsf{com}_0 = \mathsf{Commit}(s', r')$. This means that any subset of size $t+1$ of values corresponding to indices Valid lie on the same polynomial. If the dealer P_d^c is honest, then obviously $s' = s$, and for all $i \in [1, n]$ such that $P_i^{c'}$ is honest, we have $i \in \mathsf{Valid}$ so that $|\mathsf{Valid}| \geq n - t \geq t+1$. □

Lemma 22 (Linearity of CompShareFuture)**.** *Consider states*

$$(\mathbf{x}_1, \ldots, \mathbf{x}_n; \mathsf{com}_{x,0}, \ldots, \mathsf{com}_{x,t})$$

$$(\mathbf{y}_1, \ldots, \mathbf{y}_n; \mathsf{com}_{y,0}, \ldots, \mathsf{com}_{y,t}).$$

For any linear function $\mathcal{L}$, if $\perp \neq (z, \rho)$ is

$$\mathsf{CompRecFuture}(\mathcal{L}(\mathbf{x}_1, \mathbf{y}_1), \ldots, \mathcal{L}(\mathbf{x}_n, \mathbf{y}_n); \mathcal{L}(\mathsf{com}_{x,0}, \mathsf{com}_{y,0}), \ldots, \mathcal{L}(\mathsf{com}_{x,t}, \mathsf{com}_{y,t}))$$

it holds that[18] $\mathsf{Open}(\mathcal{L}(\mathsf{com}_{x,0}, \mathsf{com}_{y,0}), z, \rho) = 1$[19].

Proof. Follows from the linearity of the commitments, their binding property, and Lemma 21. □

5.3 (Computationally Secure) Verifiable Secret Sharing

If the dealer P_d^c is corrupted, protocol CompShareFuture does not guarantee that an output will be produced during CompRecFuture. To overcome this, we design maximally fluid protocols CompVSS.ShareFuture and CompVSS.RecFuture that allow a dealer $P_d^c \in \mathcal{S}^c$ to secret share a value towards a committee $\mathcal{S}^{c'}$ for some $c' \geq c+2$ and ensure that, if the dealer is not disqualified after protocol CompVSS.ShareFuture terminates, reconstruction towards committee $\mathcal{S}^{c''}$ will produce an output $s' \neq \perp$. All the properties that one expects from a verifiable secret sharing scheme hold: the value is fixed after the sharing phase, and if the dealer is honest this value is the dealer's input.

Protocol $\mathsf{VSS} - \mathsf{Share}$ works as follows: the dealer P_d^c, who holds input s, samples a uniform random polynomial $f(x)$ such that $\deg f(x) \leq t$ and $f(0) = s$. Let f_k denote the k-the coefficient of $f(x)$. The dealer *duplicates* $\mathsf{CompShareFuture}(f_k)$ for all $k \in [0, t]$ towards committees $\mathcal{S}^{c_1}, \mathcal{S}^{c'}$; the public outputs of these $t+1$ invocations of CompShareFuture result in a matrix $\mathbf{D}$ of public commitments

$$\mathbf{D} := \begin{pmatrix} \mathsf{com}_{0,0} & \ldots & \mathsf{com}_{0,t} \\ \vdots & \ddots & \vdots \\ \mathsf{com}_{t,0} & \ldots & \mathsf{com}_{t,t} \end{pmatrix}$$

where the k-th column denotes the commitments to the polynomial used to share f_k, as well as a matrix $\mathbf{D}'$ containing the commitments for the duplicate executions towards committee $\mathcal{S}^{c'}$. Notice that the matrices $\mathbf{D}$ and $\mathbf{D}'$ share the first row, as the dealer broacasts these commitments only once.

Furthermore, by linearity of CompShareFuture, committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$ hold states $\mathsf{CompShareFuture}_d(s_i = f(i))$[20] for all $i \in [1, n]$. Then P_d^c sends value s_i and the corresponding opening information[21] $\rho_i = \sum_{k=0}^{t} r_k i^k$ to $P_i^{c_1}$ by invoking CompSendFuture. If player $P_i^{c_1}$ does not receive satisfactory information from the dealer, that is, if they receive $s_i = \perp$ or $\mathsf{Open}\left(\left(\sum_{k=0}^{t} \mathsf{com}_{0,k} i^k\right), s_i, \rho_i\right) = 0$,

[18] The expression $\mathcal{L}(\mathbf{x}_i, \mathbf{y}_i)$ is to be understood as the function applied component-wise.

[19] This statement can be generalized to functions $\mathcal{L}$ with n inputs with a purely notional overhead.

[20] The subscript d denotes states resulting from an execution of CompShareFuture where P_d^c acted as the dealer.

[21] We denote by r_k the randomness from $\mathsf{com}_{0,k} = \mathsf{Commit}(f_k, r_k)$

they broadcast a complaint, otherwise, they run protocol $\mathsf{CompShareFuture}_i(s_i)$ towards committee $\mathcal{S}^{c'}$. To ensure that party $P_i^{c'}$ commits to the value they received from P_c^d, the commitment to the constant term of the polynomial used for $\mathsf{CompShareFuture}_i$ is taken as $\sum_{k=0}^t \mathsf{com}_{0,k} i^k$. This results in another $(t+1) \times n$ matrix of public commitments

$$\mathbf{P} := \begin{pmatrix} \left(\sum_{k=0}^t \mathsf{com}_{0,k} 1^k\right) & \dots & \left(\sum_{k=0}^t \mathsf{com}_{0,k} n^k\right) \\ \vdots & \ddots & \vdots \\ \mathsf{com}_t^1 & \dots & \mathsf{com}_t^n \end{pmatrix}$$

where the j-th column denotes the commitments output by $\mathsf{CompShareFuture}_j$. If a complaint is raised by player $P_i^{c_1}$, players in committee $\mathcal{S}^{c'}$ reconstruct s_i by running protocol

$$\mathsf{CompRecFuture}_d\left(\mathbf{s}_{i1}, \dots, \mathbf{s}_{in}; \sum_{k=0}^t \mathsf{com}_{0,k} 1^k, \dots, \sum_{k=0}^t \mathsf{com}_{0,k} n^k\right).$$

If the output is $\perp$, the dealer is disqualified.

In $\mathsf{CompVSS.RecFuture}$, to reconstruct s_i players in committee $\mathcal{S}^{c'}$ run protocol

$$\mathsf{CompRecFuture}_i\left(\mathbf{s}_{i1}, \dots, \mathbf{s}_{in}; \left(\sum_{k=0}^t \mathsf{com}_{0,k} i^k\right), \mathsf{com}_1^i, \dots, \mathsf{com}_t^i\right)$$

for all indices $i \in [1, n]$ corresponding to parties of committee $\mathcal{S}^{c_1}$ that did *not* complain. Then, committee $\mathcal{S}^{c''}$ uses any $t+1$ values s_i (either broadcast during $\mathsf{CompVSS.ShareFuture}$ or valid outputs of $\mathsf{CompRecFuture}_i$) to reconstruct s'.

As a last remark, observe that by having both P_d^c and players in committee $\mathcal{S}^{c_1}$ duplicate $\mathsf{CompShareFuture}_d(s_i)$ and $\mathsf{CompShareFuture}_i(s_i)$ for all $i \in [1, n]$ towards committees $\mathcal{S}^{c'}$ and $\mathcal{S}^{\tilde{c}}$ for some $\tilde{c} \geq c'$, one guarantees that if P_d^c is not disqualified, then both $\mathcal{S}^{c'}$ and $\mathcal{S}^{\tilde{c}}$ hold sharings of the *same* value.

In the following protocol description, we use the notation developed above, with the addition of the symbol $*$, denoting (an appropriate linear combination of) publicly known commitments, where the inclusion of the explicit expression would be excessively cumbersome. The properties of the protocol are summaried and proven in Lemmas 23, 24, and 25 below.

Protocol $\mathsf{CompVSS.ShareFuture}(s) \to (\mathbf{s}_1, \dots, \mathbf{s}_n; \mathbf{D}, \mathbf{P})$

- **Committee c:** The dealer P_d^c who holds input s does:
  - Sample a uniform random polynomial
  $$f(x) = \sum_{k=0}^t f_k x^k$$
  such that $f(0) = s$;
  - Duplicate $\mathsf{CompShareFuture}_d(f_k)$ for all $k \in [0, t]$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$;
  - Let r_k denote the constant term of the random polynomial used in the execution $\mathsf{CompShareFuture}(f_k)$. Let $s_i := f(i)$ and $\rho_i := \sum_{k=0}^t r_k i^k$. Do $\mathsf{CompSendFuture}(s_i)$ and $\mathsf{CompSendFuture}(\rho_i)$ towards $P_i^{c_1}$.

- **Committee c_1:** For all $i \in [1, n]$ party $P_i^{c_1}$ does:

- If the output of $\mathsf{CompSendFuture}$ is $s'_i = \perp$ or if $\mathsf{Open}(*, s'_i, \rho'_i) = 0$ broadcast $\mathsf{Complain}$;
- Invoke $\mathsf{CompShareFuture}_i(s_i)$ towards committee $\mathcal{S}^{c'}$, *but* with $\sum_{k=0}^{t} \mathsf{com}_{0,k} i^k$ as commitment to the constant term of the polynomial used;

- **Committee c':** Each party does:
  - For all $i \in [1, n]$ such that $P_i^{c_1}$ has broadcast $\mathsf{Complain}$, participate in protocol $\mathsf{CompRecFuture}_d(\mathbf{s}_{i1}, \ldots, \mathbf{s}_{in}; *)$;
  - If any output is $\perp$ disqualify the dealer.
  - Otherwise, party P_i's output is the vector of the outputs of each execution of $\mathsf{CompShareFuture}(f_k)$ and the values s_i and ρ_i.

Protocol $\mathsf{CompVSS.RecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathbf{D}, \mathbf{P}) \to s$

- **Committee c':** If $P_i^{c_1}$ did *not* broadcast $\mathsf{Complain}$, each party participates in $\mathsf{CompRecFuture}_i(\mathbf{s}_{i1}, \ldots, \mathbf{s}_{in}; *)$.[a]
- **Committee c''_1:** Each party does:
  - Let $\mathsf{Valid} := \{i \in [1, n] \mid \mathsf{CompRecFuture}_i(\mathbf{s}_{i1}, \ldots, \mathbf{s}_{in}; *) \neq \perp\}$.
  - Use $t + 1$ values s_i such that either $i \in \mathsf{Valid}$ or $P_i^{c_1}$ broadcast $\mathsf{Complain}$ to interpolate a polynomial $f'(x)$ of degree at most t. Output $f'(0)$.

[a] By invoking here the private version of $\mathsf{CompRecFuture}$ we obtain a private version of $\mathsf{CompVSS.RecFuture}$.

Lemma 23 (Privacy of $\mathsf{CompVSS.ShareFuture}$)**.** *Assume that P_d^c is honest. Then the view of a PPT adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c + 1, c']$ is identically distributed during executions* $\mathsf{CompVSS.ShareFuture}(s)$ *and* $\mathsf{CompVSS.ShareFuture}(s')$, *for any $s \neq s'$.*

Proof. By assumption, all commitments broadcast by the dealer are perfectly hiding. By Lemma 20, from committees $\mathcal{S}^k$ for $k \in [c + 1, c_1]$ the adversary only obtains t evaluations on the polynomial $f(x)$ of degree (at most) t, as well as t shares of each coefficient f_k of $f(x)$ as output from $\mathsf{CompShareFuture}$, so that until this point their view remains independent from s. From committees $\mathcal{S}^k$ for $k \in [c_1 + 1, c']$ the adversary only obtains t (independent form the previous) shares of each coefficient f_k of $f(x)$, as well as evaluation $f(i) = s_i$ for all $i \in [1, n]$ such that party $P_i^{c_1}$ broadcasts $\mathsf{Complain}$. However, since P_d^c is honest, only parties that are corrupted in $\mathcal{S}^{c'}$ broadcast complaints, so that overall the adversary does not learn more than t evaluations of $f(x)$, and their view remains independent of s until the of the protocol $\mathsf{CompVSS.ShareFuture}$. □

Lemma 24 (Extractability/Robustness of $\mathsf{CompVSS.ShareFuture}$)**.** *Assume that at most $t < n/2$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$. If P_d^c is not disqualified after* $\mathsf{CompVSS.ShareFuture}$, *then it is possible to efficiently compute values s' as well as randomness r' from public information and the states of honest parties such that* $\mathsf{Open}\left(\mathsf{com}_{0,0}, s', r'\right) = 1$ *and*

$$s' := \mathsf{CompVSS.RecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathbf{D}, \mathbf{P}).$$

Furthermore, if the dealer is honest $s' = s$.

Proof. To compute s', consider all values s_i relative to honest parties (either extracted via Lemma 21, or reconstructed via CompRecFuture if $P_i^{c_1}$ complains). Since P_d^c is not disqualified, there are at least $n - t \geq t + 1$ of such values, and by Lemma 22 together with the binding property the commitments we conclude that any subset of size $t + 1$ of them must lie on the same polynomial $f'(x)$ of degree at most t. Let $s' := f'(0)$. It is easy to show that s' computed in this way has all the required properties. If the dealer is honest, the shares of honest parties lie on $f(x)$, so that $s' = s$. □

Lemma 25 (Linearity of CompVSS.ShareFuture). *Consider the following states resulting from executions of* CompVSS.ShareFuture *in which at most $t < n/2$ parties are corrupted by a PPT adversary in each committee:*[22]

$$(\mathbf{x}_1, \ldots, \mathbf{x}_n; \mathbf{D}_x, \mathbf{P}_x),$$

$$(\mathbf{y}_1, \ldots, \mathbf{y}_n; \mathbf{D}_y, \mathbf{P}_y).$$

For all linear functions $\mathcal{L}$ it is possible to efficiently compute values z' and ρ' from the honest parties states of

$$\mathsf{State}_{\mathcal{L}} := (\mathcal{L}(\mathbf{x}_1, \mathbf{y}_1), \ldots, \mathcal{L}(\mathbf{x}_n, \mathbf{y}_n); \mathcal{L}(\mathbf{D}_x, \mathbf{D}_y), \mathcal{L}(\mathbf{P}_x, \mathbf{P}_y))$$

such that

1. $z' := \mathsf{CompVSS.RecFuture}(\mathsf{State}_{\mathcal{L}})$
2. $\mathsf{Open}(\mathcal{L}(\mathbf{D}_{00}^x, \mathbf{D}_{00}^y), z, \rho) = 1.$

Proof. Follows from the linearity of commitments, their computational binding property, and Lemmas 22 and 24. □

5.4 (Computationally Secure) 2-Level Verifiable Secret Sharing

To perform the multiplication proof in our MPC protocol, we need a simpler sharing state than that achieved by CompVSS.ShareFuture. In particular, we want each party in the committee $\mathcal{S}^c$ tasked with evaluating a certain arithmetic gate to hold consistent degree t-Shamir sharings of the input values to the gates. Building on CompVSS.ShareFuture, we explain how to achieve this in a simple way: the dealer P_d^c duplicates CompVSS.ShareFuture(f_k) for all $k \in [0, t]$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$, where $f(x)$ is a uniform random polynomial of degree at most t such that $f(0) = s$. Then, each party in $\mathcal{S}^{c_1}$ participates in protocol CompVSS.RecFuture (privately) towards player $P_i^{c'}$, to deliver to them the value $s_i = f(i)$. Notice that committee $\mathcal{S}^{c'}$ also holds sharings of all values $f(j)$ for $j \in [1, n]$ thanks to linearity of CompVSS.ShareFuture. We denote the state achieved by this 2-level sharing as Comp2VSS.ShareFuture(s). Observe that we do not describe a corresponding reconstruction procedure, as protocol CompVSS.RecFuture can be invoked for the secret s as well as for each share s_i individually.

To *duplicate* a state Comp2VSS.ShareFuture(s) towards many committees, the dealer P_d^c can sample 2 polynomials $f(x)$ and $f'(x)$ of degree at most t such that $f(0) = s = f'(0)$ and run respective executions of Comp2VSS.ShareFuture towards the the different committees, *but* duplicating the sharing of the constant coefficient CompVSS.ShareFuture($f_0 = s = f'_0$) to all committees instead of running independent instances. This ensures that all committees hold (independent) Shamir-sharings of the same value without violating privacy. Observe that simply duplicating all states CompVSS.ShareFuture(f_k) for $k \in [0, t]$ for a single polynomial $f(x)$ would indeed violate privacy, as $\mathcal{A}$ learns t evaluations of $f(x)$ from each committee.

[22] We will often denote such a state as CompVSS.ShareFuture(s)

We describe the non-duplicated version of the protocol below. We denote by $\mathbf{\Delta}$ and $\mathbf{\Pi}$ the vectors of matrices of public commitments $(\mathbf{D}_0, \ldots, \mathbf{D}_t)$ and $(\mathbf{P}_0, \ldots, \mathbf{P}_k)$ respectively, resulting from $t+1$ states $\mathsf{CompVSS.ShareFuture}(f_k)$ for all f_k with $k \in [0, t]$. The properties of the protocol are formalized and proven in Lemmas 27, 26, and 28 below.

Protocol $\mathsf{Comp2VSS.ShareFuture}(s) \to (\mathbf{S}_1, \ldots, \mathbf{S}_n; \mathbf{\Delta}, \mathbf{\Pi})$

- **Committee** c**:** The dealer P_d^c who holds input s does:
  - Sample a uniform random polynomial
  $$f(x) = \sum_{k=0}^{t} f_k x^k$$
  such that $f(0) = s$;
  - Duplicate $\mathsf{CompVSS.ShareFuture}(f_k)$ for all $k \in [0, t]$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$;
- **Committee** c_1**:** Disqualify the dealer if the dealer is disqualified in any execution of $\mathsf{CompVSS.ShareFuture}$. Otherwise, each player participates in protocol $\mathsf{CompVSS.RecFuture}$ to reconstruct (privately) value $s_i = f(i)$ towards party $P_i^{c'}$.
- **Committee** c'**:** Disqualify the dealer if the dealer is disqualified in any execution of $\mathsf{CompVSS.ShareFuture}$.

Lemma 26 (Privacy of $\mathsf{Comp2VSS.ShareFuture}$). *Assume that P_d^c is honest. Then the view of a PPT adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c+1, c']$ is identically distributed during executions* $\mathsf{Comp2VSS.ShareFuture}(s)$ *and* $\mathsf{Comp2VSS.ShareFuture}(s')$*, for any $s \neq s'$.*

Proof. By assumption, all commitments broadcast by the dealer are perfectly hiding. From committees $\mathcal{S}^k$ for $k \in [c+1, c_1]$, by Lemma 23, the adversary only learns t shares of each coefficient f_k of $f(x)$, for $k \in [0, t]$. Furthermore, from committees $\mathcal{S}^k$ for $k \in [c_1 + 1, c']$, again by Lemma 23 and because the dealer duplicates $\mathsf{CompVSS.ShareFuture}$ honestly, the adversary learns only 1) t (fresh) sharings of each coefficient f_k for $k \in [0, t]$ and 2) t evaluations of the polynomial $f(x)$ (one from each corrupted party in committee $\mathcal{S}^{c'}$). Therefore, their view is statistically independent from m. □

Lemma 27 (Extractability/Robustness of $\mathsf{Comp2VSS.ShareFuture}$). *Assume that at most $t < n/2$ parties are corrupted in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If P_d^c is not disqualified after* $\mathsf{Comp2VSS.ShareFuture}$*, then it is possible to efficiently compute values s' as well as randomness ρ' from public information and the states of honest parties such that* $\mathsf{Open}\left(\mathbf{\Delta}_{000}, s', r'\right) = 1$ *and*[23]

$$s' := \mathsf{CompVSS.RecFuture}(\mathbf{s}_1, \ldots, \mathbf{s}_n; \mathbf{\Delta}, \mathbf{\Pi}).$$

Furthermore, if the dealer is honest $s' = s$.

Proof. If the dealer P_d^c does not get disqualified after protocol $\mathsf{Comp2VSS.ShareFuture}$ terminates, then the dealer is not disqualified in *any* execution of $\mathsf{CompVSS.ShareFuture}$. Let s_i' and r_i' denote the values extracted from each state of $\mathsf{CompVSS.ShareFuture}(s_i)$ for all $i \in [1, n]$ using Lemma 24. Then $\mathsf{Open}(\sum_{k=0}^{t} \mathbf{\Delta}_{00k} i^k, s_i', \rho') = 1$, and from any $t+1$ such openings, via Lagrange interpolation and linearity of the commitments, one can efficiently compute an opening for $\mathbf{\Delta}_{000}$. By the binding property of

[23] The commitment $\mathbf{\Delta}_{000}$ denotes $(\mathbf{D_0})_{00}$.

the commitments, all these openings must be equal, so that any subset of $t+1$ values s'_i lies on the same polynomial $f'(x)$ of degree at most t. Since there are at least $n-t \geq t+1$ honest parties in $\mathcal{S}^{c'}$, from their states one can efficiently compute $s' := f(0)$, and $\rho' := r'(0)$, where $r(x)$ denotes the polynomial interpolated from values r'_i of honest parties. It is easy to verify that these values have the required properties. Robustness follows easily from that of protocol CompVSS.ShareFuture (see Lemma 24). □

Lemma 28 (Linearity of Comp2VSS.ShareFuture)**.** *Consider the following states resulting from executions of* Comp2VSS.ShareFuture *in which at most $t < n/2$ parties are corrupted by a PPT adversary in each committee:*

$$(\mathbf{X}_1, \ldots, \mathbf{X}_n; \mathbf{\Delta}^x, \mathbf{\Pi}^x),$$
$$(\mathbf{Y}_1, \ldots, \mathbf{Y}_n; \mathbf{\Delta}^y, \mathbf{\Pi}^y).$$

For all linear functions $\mathcal{L}$ it is possible to efficiently compute values z' and ρ' from the honest parties states of

$$\mathsf{State}_{\mathcal{L}} := (\mathcal{L}(\mathbf{X}_1, \mathbf{Y}_1), \ldots, \mathcal{L}(\mathbf{X}_n, \mathbf{Y}_n); \mathcal{L}(\mathbf{\Delta}^x, \mathbf{\Delta}^y), \mathcal{L}(\mathbf{\Pi}^x, \mathbf{\Pi}^y)).$$

such that

1. $\mathsf{Open}(\mathcal{L}(\mathbf{\Delta}^x_{000}, \mathbf{\Delta}^y_{000}), z', \rho') = 1$*;*
2. $z' = \mathsf{Comp2VSS.ShareFuture}(\mathsf{State}_{\mathcal{L}})$*.*

Proof. Follows from the linearity of commitments, their computational binding property, and Lemmas 25 and 24. □

5.5 (Computationally Secure) Maximally-Fluid Multiplication Proof

In this section, we present a protocol that allows a prover knowing openings for some appropriate states CompVSS.ShareFuture to prove towards a future committee, holding duplicates of these states, that a certain *non linear* relation between them holds. We follow the idea of [CDD+99].

Originally designed in the statistical security setting, this approach requires the prover and the parties holding the states in question to engage in a distributed zero-knowledge proof. To generate the random challenge for the prover, we ask each party in one committee to execute CompVSS.ShareFuture(r_i) for a uniform random value, and reconstruct the sum of all executions that succeed. The privacy properties of protocol CompVSS.ShareFuture (see Lemma 23) guarantee that the challenge is unpredictable. Adapting this approach to the fluid model comes with the usual challenges of dealing with the number of interaction rounds required, while maintaining the necessary state across committees but without violating secrecy. The full protocol works as follows: committees $\mathcal{S}^{c_1}, \mathcal{S}^{c_2}, \mathcal{S}^{c_3}, \mathcal{S}^{c'}$ hold duplicate states CompVSS.ShareFuture of values x, y, z. Intuitively, a prover $P^c_p \in \mathcal{S}^c$, who knows values x, y, z and the corresponding opening information, must convince parties in $\mathcal{S}^{c'}$ that $x \cdot y = z$. To this end, the prover P^c_p samples a uniform random value β, computes $\gamma = x\beta$, and duplicates CompVSS.ShareFuture(β) and CompVSS.ShareFuture(γ) towards committees $\mathcal{S}^{c_1}, \mathcal{S}^{c_2}$, and $\mathcal{S}^{c_3}$. In the meantime, each player P^c_i samples a uniform random value r_i and invokes protocol CompVSS.ShareFuture(r_i) towards $\mathcal{S}^{c_1}$ (these sharings will be added and reconstructed to produce a random challenge for the prover). If any execution of CompVSS.ShareFuture where the prover acts as the dealer fails, the proof fails. Otherwise, committee $\mathcal{S}^{c_1}$ runs the CompVSS.RecFuture protocol for value $r = \sum_{i \in \mathsf{Alive}} r_i$ for the set Alive of parties that were not disqualified as dealers of CompVSS.ShareFuture(r_i). Then, in succession, committees $\mathcal{S}^{c_2}$ and $\mathcal{S}^{c_3}$ run the CompVSS.RecFuture protocol for values $r' = ry + \beta$ and $r'' = r'x - rz - \gamma$ respectively. If $r'' \neq 0$, the proof fails, otherwise the proof is accepted. Intuitively, if $xy \neq z$, once β and γ are fixed, the equations hold for a unique r, and because the prover does not know r when committing to β and γ, they can only cheat with probability $1/|\mathbb{F}|$. The properties of this protocol are summarized and proven in Lemmas 29 and 30 below.

Protocol $\mathsf{CompMultProof}(x, y, z)$

- **Committee** c**:** The prover P_p^c does:
  - Sample uniform random β, and compute $\gamma := x\beta$;
  - Duplicate $\mathsf{CompVSS.ShareFuture}(\beta)$ and $\mathsf{CompVSS.ShareFuture}(\gamma)$ towards committees $\mathcal{S}^{c_1}, \mathcal{S}^{c_2}$, and $\mathcal{S}^{c_3}$.

  Each party P_i^c does:
  - Sample a uniform random r_i;
  - Invoke $\mathsf{CompVSS.ShareFuture}(r_i)$ towards committee $\mathcal{S}^{c_1}$.
- **Committee** c_1**:** Each party does:
  - If P_p^c is disqualified during either execution of $\mathsf{CompVSS.ShareFuture}$ the proof fails;
  - Let
    $$\mathsf{Alive} := \{i \in [1, n] \mid P_i^c \text{ not disqualified in } \mathsf{CompVSS.ShareFuture}(r_i)\};$$
  - Let $r := \sum_{i \in \mathsf{Alive}} r_i$;
  - Participate in $\mathsf{CompVSS.RecFuture}$ for r;
- **Committee** c_2**:** Each party does:
  - Participate in $\mathsf{CompVSS.RecFuture}$ for value $r' = ry + \beta$;
- **Committee** c_3**:** Each party does:
  - Participate in $\mathsf{CompVSS.RecFuture}$ for value $r'' = r'x - rz - \gamma$;
- **Committee** c'**:** If the output of $\mathsf{CompVSS.RecFuture}(r'')$ is not 0, the proof fails. Otherwise, the proof is accepted.

Lemma 29 (Soundness and Completeness of $\mathsf{CompMultProof}$)**.** *Consider an execution of protocol* $\mathsf{CompMultProof}(x, y, z)$[24] *in which a PPT adversary corrupts at most $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c, c']$. If the proof is accepted, and x', y', z' denote values extracted from the corresponding states* $\mathsf{VSSCrypto} - \mathsf{ShareFuture}(x)$, $\mathsf{VSSCrypto} - \mathsf{ShareFuture}(y)$, *and* $\mathsf{VSSCrypto} - \mathsf{ShareFuture}(z)$ *via* Lemma 24*, then with probability* $1 - 1/|\mathbb{F}|$ *it holds that $x'y' = z'$. Furthermore, if the prover is honest, the proof succeeds.*

Proof. Follows by Lemma 25 and the observation that since $r'' = r'x' - rz' - \gamma = 0$ then $r(x'y' - z') + \beta x' - \gamma = 0$. Assuming $x'y' \neq z'$, then for any fixed β and γ, there is a unique r such that the equation holds. Since β and γ are chosen independently from r (this is guaranteed by Lemma 23), and r is uniform

[24] A full notation should include, as inputs to the protocol, three states resulting from $\mathsf{CompVSS.ShareFuture}$ as well as private inputs for the prover P_d^c. We adopt this more compact informal notation to improve readability.

random (for each honest party P_i^c, $i \in \mathsf{Alive}$), then the probability that $x'y' \neq z'$ and the proof succeeds is $1/\mathbb{F}$. It is straightforward to check that, if the dealer is honest, the proof succeeds. □

Lemma 30 (Privacy of CompMultProof)**.** *Assume that P_d^c is honest. Then the view of an adversary who corrupts $t < n/2$ parties in each committee $\mathcal{S}^k$ for $k \in [c, c']$ is identically distributed during executions of* $\mathsf{CompMultProof}(x, y, x)$ *and* $\mathsf{CompMultProof}(x', y', z')$ *for $(x, y, z) \neq (x', y', z')$.*

Proof. Follows immediately from Lemma 23 and the observation that all reconstructed values in committees $\mathcal{S}^{c_2}$, $\mathcal{S}^{c_3}$, and $\mathcal{S}^{c'}$ are uniform random. □

5.6 Maximally-Fluid Computationally Secure MPC

We are ready to present our full MPC protocol. The function to be computed is encoded as a layered arithmetic circuit over $\mathbb{F}$ with fan-in 2, denoted by Circ.[25] For all gates g in layer L of Circ there is a committee $\mathcal{S}^c$ that holds states Comp2VSS.ShareFuture of the input values x, y to g, as well as one committee $\mathcal{S}^{c'}$ that holds the state Comp2VSS.ShareFuture for the output value z. This invariant allows to carry out the full computation.[26]

Input Gates. Each client invokes protocol Comp2VSS.ShareFuture(x) towards committee $\mathcal{S}^{c'}$.

Addition Gates. The computation can be performed locally exploiting the linearity of Comp2VSS.ShareFuture. The only problem that arises in the fluid model lies in transferring the state to committee $\mathcal{S}^{c'}$. However, since $c' \geq c$ we can assume without loss of generality that committee $\mathcal{S}^{c'}$ already holds duplicated states Comp2VSS.ShareFuture of values x and y (by simply requiring that whoever it responsible for producing these states duplicates them to $\mathcal{S}^{c'}$ at sharing time).

Multiplication Gates. We can assume that states Comp2VSS.ShareFuture of x and y are duplicated onto all necessary committees (by the same reasoning as above). Then each party P_i^c, who knows shares x_i and y_i, locally computes $z_i := x_i \cdot y_i$ and invokes Comp2VSS.ShareFuture(z_i) towards an auxiliary committee $\mathcal{S}^{c_1}$. Furthermore, they prove to $\mathcal{S}^{c_1}$ that indeed $z_i = x_i \cdot y_i$. If the proof fails, parties in $\mathcal{S}^{c_1}$ (who hold sharings of x_i and y_i for all $i \in [1, n]$) reconstruct these values and take z_i to be $x_i \cdot y_i$ towards $\mathcal{S}^{c'}$. Using the values x_i for which the multiplication proofs succeeded, each party computes state CompVSS.ShareFuture(z_i) exploiting the linear properties of CompVSS.ShareFuture.

Output Gates. Parties in $\mathcal{S}^c$ participate in protocol CompVSS.RecFuture to (privately) reconstruct y towards the client.

We provide a formal description of the protocol and prove its security in Theorem 2 below.

Protocol CryptoCircEval

Input.

This gate has 1 input wire with value x and 1 output wire with value x, and a client associated to it. A client holds input x, and invokes

$$\mathsf{Comp2VSS.ShareFuture}(x)$$

[25] This is without loss of generality, see for example [CGG⁺21]

[26] This is only to make the exposition more clear. Indeed, the committees holding sharing states to the inputs of each gate need not be the same.

towards a future committee $\mathcal{S}^{c'}$ (all clients towards the same committee); If the client is disqualified, take 0 as input.

Output.

This gate has 1 input wire and 1 output wire, and a client associated to it. Let y denote the input wire value. Each party P_i^c participates in protocol

$$\mathsf{CompVSS.RecFuture}(\mathbf{y}_1, \ldots, \mathbf{y}_n; \mathbf{D}_y, \mathbf{P}_y)$$

towards the client.

Addition.

This gate has 2 input wires with values x, y and 1 output wire with value z. As $c' > c$ without loss of generality we can assume that $\mathcal{S}^{c'}$ holds duplicated states $\mathsf{Comp2VSS.ShareFuture}(x)$ and $\mathsf{Comp2VSS.ShareFuture}(y)$. Committee $\mathcal{S}^{c'}$ computes $\mathsf{Comp2VSS.ShareFuture}(z)$ locally as

$$\mathsf{Comp2VSS.ShareFuture}(x) + \mathsf{Comp2VSS.ShareFuture}(y)$$

by exploiting the linearity of Comp2VSS.ShareFuture.

Multiplication.

This gate has 2 input wires with values x, y and 1 output wire with value z. Committee $\mathcal{S}^c$ holds states $\mathsf{Comp2VSS.ShareFuture}(x)$ and $\mathsf{Comp2VSS.ShareFuture}(y)^a$.

- **Committee** c : Each party P_i^c does:
  - Compute $z_i = x_i \cdot y_i$;
  - Invoke $\mathsf{Comp2VSS.ShareFuture}(x_i)$ towards committee $\mathcal{S}^{c_1}$, *but* using the already known state $\mathsf{CompVSS.ShareFuture}(x_i)$ as a sharing to the constant coefficient of the polynomial used;
  - Invoke $\mathsf{Comp2VSS.ShareFuture}(y_i)$ towards committee $\mathcal{S}^{c_1}$, *but* using the already known state $\mathsf{CompVSS.ShareFuture}(y_i)$ as a sharing to the constant coefficient of the polynomial used;
  - Invoke $\mathsf{Comp2VSS.ShareFuture}(z_i)$ towards committees $\mathcal{S}^{c_1}$ and $\mathcal{S}^{c'}$;
  - Run protocol $\mathsf{CompMultProof}(x_i, y_i, z_i)$ towards committee $\mathcal{S}^{c_1}$.

- **Committee** c_1 : Each party does:
  - Let $\mathsf{Succeed} := \{i \in [1, n] \mid \mathsf{CompMultProof}(x_i, y_i, z_i) \text{ succeeds}\}$;
  - For all $i \notin \mathsf{Succeed}$, participate in

$$\mathsf{CompVSS.RecFuture}(\mathbf{x}_1, \ldots, \mathbf{x}_n; \mathbf{D}_x, \mathbf{P}_x)$$

$$\mathsf{CompVSS.RecFuture}(\mathbf{y}_1, \ldots, \mathbf{y}_n; \mathbf{D}_y, \mathbf{P}_y).$$

- **Committee** c' : Each party $P_j^{c'} \in \mathcal{S}^{c'}$ does:

- For all $i \notin$ Succeed take a standard state $\mathsf{CompVSS.ShareFuture}(z_i)$ and then compute $\mathsf{Comp2VSS.ShareFuture}(z)$ locally as

$$\mathcal{L}\Big(\mathsf{Comp2VSS.ShareFuture}(z_1), \ldots, \mathsf{Comp2VSS.ShareFuture}(z_n)\Big)$$

by exploiting the linearity of Comp2VSS.ShareFuture.

[a] We can assume that these states are duplicated towards committee $\mathcal{S}^{c1}$ without loss of generality

Theorem 2. *Assuming non-interactive linearly-homomorphic equivocal commitments, protocol* CryptoCircEval *is a maximally-fluid MPC for all polynomial functions, that achieves* $1/2$*-computational security and guaranteed output delivery.*

Proof. First, we describe a simulator Sim for protocol CryptoCircEval.

- *Input Gates.* For each dishonest client C, the simulator Sim executes Comp2VSS.ShareFuture on behalf of all the honest parties in committees $\mathcal{S}^k$ for $k \in [c, c']$. If at the end of Comp2VSS.ShareFuture the client C is not disqualified, then Sim uses the *exctractability* of protocol Comp2VSS.ShareFuture (see Lemma 27) to obtain the client's input x, and inputs this to the TTP. Otherwise, input 0 to the TTP.

  For each honest client, the simulator executes an instance of Comp2VSS.ShareFuture with input 0.

- *Addition/Multiplication Gates.* The simulator Sim executes all steps as in the real world on behalf of all honest parties in all committees $\mathcal{S}^k$ for $k \in [c, c']$.

- *Output Gates.* For each output gate associated with a corrupted client client C, the simulator obtains the output y of the gate from the TTP. The simulator computes states for all honest party in $\mathcal{S}^c$ consistent with the state of honest parties in $\mathcal{S}^c$ *and* the value y, by exploiting the equivocal property of the commitments. Then Sim emulates instances of the corresponding instance of $\mathsf{CompVSS.RecFuture}(y)$ towards C on behalf of all honest parties in committees $\mathcal{S}^k$ for $k \in [c, c'-1]$ based on these computed states.

We prove the claim by a hybrid argument. Consider the following hybrid worlds.

- *Hybrid* 0. This is the *real world,* where Sim has access to all inputs of honest clients and simply executes CryptoCircEval on behalf of all of honest parties in each committee.

- *Hybrid* 1. This world is identical to Hybrid 0, *until* the last message sent from honest parties to the clients when processing output gates. These messages are computed by Sim as in the ideal world (see above).

- *Hybrid* 2. This world is identical to Hybrid 1, except that in each multiplication gate, the simulator does the following on behalf of each honest party in each invocation of CompMultProof by an honest parties acting as the prover:

  - Sample r' uniform at random, and based on r' and the state of the dishonest parties use the equivocal property of the commitments to compute valid openings for $\mathsf{CompVSS.RecFuture}(r')$.
  - Set $r'' = 0$, and based on the states of dishonest parties, use the equivocal property of the commitments to compute valid openings for $\mathsf{CompVSS.RecFuture}(r'')$.

- *Hybrid* 3. This is the ideal world: the simulator does not have access to any of the inputs of the honest clients, but instead sim emulates each execution of Comp2VSS.ShareFuture with input 0 on behalf of honest clients, and then proceeds as explained above.

Claim 1. *Hybrids* 0 *and* 1 *are indistinguishable.*

<u>Proof:</u> All messages are the same up to the output gates. In the output gate, the simulator outputs honest parties messages that are consistent with the output given by the trusted party y and the states from corrupted parties.

In the real world, the honest messages are computed according to the protocol CompVSS.RecFuture. First note that the output obtained in the real world is the same output as in Hybrid 1, since Comp2VSS.ShareFuture is robust and linear (see Lemmas 27 and 28).

Furthermore, by privacy of Comp2VSS.ShareFuture and the multiplication proof (see Lemma 26 and 30), the states of corrupted parties are completely independent of the inputs from honest clients and distributed identically in both worlds. By linearity of CompVSS.ShareFuture (see Lemma 25) and equivocation of commitments (see Definition 4), the computed states from honest parties (along with the states of corrupted parties) are consistent with the correct output y in Hybrid 1, which is the same as in the real world. Both hybrids are therefore identically distributed.

■

Claim 2. *Hybrids* 1 *and* 2 *are indistinguishable.*

<u>Proof:</u> Consider an honest prover P^c that attempts to perform a multiplication proof CompMultProof prove that $z = x \cdot y$ (each corrupted prover is handled the same way in both worlds.) First, note that in Hybrid 1, the reconstructed value r' is blinded by β, which is uniformly random and unknown to the adversary. Moreover, since the prover is honest, Lemma 16 guarantees that the proof succeeds (which means that $r'' = 0$).

In Hybrid 2, the simulator computes r' (directly) as a uniform random value, and computes opening information (as in the proof of the previous claim), so that the value r' is reconstructed. The same is done so that the value reconstructed r'' is 0.

Since the values r' and r'' are distributed identically in both hybrids (and also the messages sent during CompVSS.RecFuture on behalf of the honest parties follow the same distribution in both worlds), the hybrids are indistinguishable. ■

Claim 3. *Hybrids* 2 *and* 3 *are indistinguishable.*

<u>Proof:</u> The only difference is that in Hybrid 2, the sharing state of protocol Comp2VSS.ShareFuture with respect to the internal wire values of Circ are computed based on the real inputs from honest clients, while in Hybrid 3, they are computed based on the honest clients inputs set to 0. By secrecy of Comp2VSS.ShareFuture (see Lemma 26), and the fact that the multiplication proofs from honest parties do not fail (see Lemma 29), the view of the adversary is the same in both hybrids. In addition, by Lemma 27 and the binding property of the commitments (see Definition 4), the extracted input value from each corrupted client in Hybrid 3 is consistent with the shared state held by the corresponding committee after the input gate is processed in Hybrid 2. ■

□

Extending to Security against R-adaptive **Adversaries.** Similar to our perfectly secure protocol, for simplicity, we implicitly assume that the adversary is NR-adaptive in the above proof as well. We now briefly discuss why we expect our protocol to also be secure against R-adaptive adversaries.

Such adversaries can be handled by our simulation roughly as follows. For each committee assigned to handle a computation gate (addition or multiplication), whenever a party $P_i^c \in \mathcal{S}^c$ from a past committee is corrupted, the simulator can simply reveal to the adversary the emulated state on behalf of P_i^c that it has so far, in an execution where the honest clients have input 0. Since the adversary holds only up to t states from each committee, secrecy of our sub-protocols (see Lemmas 26 and 30), ensures that the view of the adversary is independent of any values from honest parties. If the committee is executing an output gate, the simulator needs to instead output the state that was patched consistently with the output of that gate according to the simulation of the output gate.

Acknowledgements

We would like to thank Martin Hirt for helpful discussions on perfectly secure MPC.

References

[AHKP22] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES: MPC with small clients and larger ephemeral servers. Cryptology ePrint Archive, Report 2022/751, 2022. `https://eprint.iacr.org/2022/751`.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, January 2017.

[BELO14] Joshua Baron, Karim El Defrawy, Joshua Lampkins, and Rafail Ostrovsky. How to withstand mobile virus attacks, revisited. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 293–302. ACM, July 2014.

[BGG$^+$20] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, November 2020.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BHKL18] Assi Barak, Martin Hirt, Lior Koskas, and Yehuda Lindell. An end-to-end system for large scale P2P MPC-as-a-service and low-bandwidth MPC for weak participants. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 695–712. ACM Press, October 2018.

[BKLZL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Heidelberg, November 2020.

[BTH06] Zuzana Beerliová-Trubíniová and Martin Hirt. Efficient multi-party computation with dispute control. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 305–328. Springer, Heidelberg, March 2006.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Heidelberg, August 1988.

[CDD$^+$99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, Heidelberg, May 1999.

[CDGK22] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. Cryptology ePrint Archive, Report 2022/242, 2022. `https://eprint.iacr.org/2022/242`.

[CDK$^+$23] Matteo Campanelli, Bernardo David, Hamidreza Khoshakhlagh, Anders Konring, and Jesper Buus Nielsen. Encryption to the future: a paradigm for sending secret messages to future (anonymous) committees. In *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part III*, pages 151–180. Springer, 2023.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

[CGG$^+$21] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, August 2021. Springer, Heidelberg.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th FOCS*, pages 383–395. IEEE Computer Society Press, October 1985.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[DEP21] Ivan Damgård, Daniel Escudero, and Antigoni Polychroniadou. Phoenix: Secure computation in an unstable network with dropouts and comebacks. Cryptology ePrint Archive, Report 2021/1376, 2021. `https://eprint.iacr.org/2021/1376`.

[DKI$^+$23] Bernardo David, Anders Konring, Yuval Ishai, Eyal Kushilevitz, and Varun Narayanan. Perfect mpc over layered graphs. Cryptology ePrint Archive, Paper 2023/330, 2023. `https://eprint.iacr.org/2023/330`.

[GHK$^+$21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, August 2021. Springer, Heidelberg.

[GIKR01] Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *33rd ACM STOC*, pages 580–589. ACM Press, July 2001.

[GKM$^+$20] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. Cryptology ePrint Archive, Report 2020/504, 2020. `https://eprint.iacr.org/2020/504`.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

[HMP00] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 143–161. Springer, Heidelberg, December 2000.

[KRY22] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Towards efficient YOSO MPC without setup. Cryptology ePrint Archive, Report 2022/187, 2022. `https://eprint.iacr.org/2022/187`.

[Mic17] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

[NRO22] Jesper Buus Nielsen, João L. Ribeiro, and Maciej Obremski. Public randomness extraction with ephemeral roles and worst-case corruptions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 127–147. Springer, Heidelberg, August 2022.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[PCRR09] Arpita Patra, Ashish Choudhary, Tal Rabin, and C. Pandu Rangan. The round complexity of verifiable secret sharing revisited. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 487–504. Springer, Heidelberg, August 2009.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[RS22] Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 719–749. Springer, Heidelberg, August 2022.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.