MPC with Weighted and Fluid Participation

Ghada Almashaqbeh[1] and Sumedh Ghude[2⋆]

[1] University of Connecticut, ghada@uconn.edu
[2] Northeastern University, sumedh.ghude@gmail.com

Abstract. Most existing multiparty computation (MPC) protocols assume static and equal participation; all computing parties can dedicate resources and stay online for the entire computation, and they have equal influence on computation correctness and security (i.e., they are equally hard to corrupt). Supporting dynamic participation, so parties can join/leave at will, and associating weights to these parties reflecting their trust level, are essential to enable MPC in contemporary emerging applications. Existing solutions addressed these issues separately at varying security levels, and attempting to combine them while addressing malicious security raises several challenges and open questions.
In this paper, we close this gap and develop maliciously-secure MPC protocols that support weighted participation in both static and dynamic (fluid) settings. We extend a prior weighted ramp secret sharing scheme to support verifiability and proactivity, which could be of independent interest. Then, we show how to perform arithmetic operations over weighted shares in a verifiable way, thus enabling maliciously-secure weighted MPC with guaranteed output delivery under static participation. Utilizing the proactivity feature of our secret sharing scheme, we develop a state handoff protocol to support dynamic participation in committee-based MPC under the same security guarantees. Both protocols are synchronous and assume honest majority. To the best of our knowledge, our work is the first to support MPC with weighted and dynamic participation.

1 Introduction

Secure multiparty computation (MPC) allows multiple parties to compute over their private inputs without revealing anything about these inputs beyond what can be inferred from the output [9, 43, 49]. Usually, all computing parties, or servers, are treated equally (in terms of trust and compute resources) and they stay online for the entire protocol execution. Security rests on assuming that an adversary cannot corrupt more than a threshold number of these servers, and that it is equally hard to corrupt any server.

Recent emerging applications and technologies raise the question of whether such a static and indifferent party participation is sufficient. For example, in blockchains, miners can join and leave the system at will—encapsulating a *flexible, non-committing* participation. Also, training complex machine learning models over distributed datasets may take days; it could be the case that not all servers can commit resources or stay online for the entire computation.

In parallel, assuming that all these servers have the same trust level may not capture real life use cases. Instead, a party would have a *weight* reflecting its influence on correctness and security of the private computation. For example, in computing over medical records, a well-established hospital might have a greater weight than small practices. Also, in proof-of-stake systems or collateralized settings, a party's weight would be proportional to its stake and even may depend on a reputation score accounting for how this party behaved so far (as in blockchain oracle networks [16] or off-chain payments [32]). This could be useful in a weighted version of MPC-as-a-service; registered servers have weights based on their collateral and reputation score. Another example is MPC ceremonies for setup generation [1, 10, 14]. These usually indicate participation of high-profile parties, as in [1], as a way of establishing credibility. Thus, a natural extension is reflecting that on the MPC itself by having weighted parties.

These observations have led to new flavors of MPC models. Fluid MPC [12, 19, 46] supports dynamic participation so parties show up when they have enough resources for the computation, and can leave without interrupting the protocol. YOSO (You Speak Only Once) [15, 17, 38] allows unknown parties to dynamically

⋆ Most work done while at the University of Connecticut.

participate in the computation, thus supporting MPC in open systems (Fluid MPC works in closed systems of known parties). Still, all these works assume unweighted servers. Weighted MPC [35] supports servers with weights so the trust assumption is a certain fraction of the total weight is honest. This protocol relies on an efficient (non-linear) weighted ramp secret sharing (WRSS), with an additive overhead in the weights instead of multiplicative as in the naive virtualization approach. However, weighted MPC assumes static participation, and is secure only against semi-honest adversaries—devising a verifiable WRSS scheme to address malicious security was left as an open problem.

One may argue that addressing dynamic and weighted settings can be done by simply combining prior work. This is not the case; several challenges arise, requiring new techniques to resolve them, and this becomes more challenging when aiming for malicious security with guaranteed output delivery (GOD).

In this paper, we focus on closed systems and ask the following question: *Can we realize maliciously-secure weighted MPC with GOD while maintaining an additive overhead? and can we extend that to support dynamic participation?*

1.1 Contributions and Technical Overview

We answer this question in the affirmative by introducing two weighted MPC protocols. The first supports GOD in the malicious setting with static participation, and the second extends this guarantee to the fluid (dynamic participation) setting. As Cleve [20] showed that an honest majority is necessary for MPC with GOD, we assume honest majority. Underlying both protocols, we devise a verifiable and proactive WRSS scheme, which could be of independent interest.

Proactive and verifiable WRSS. The WRSS scheme from [35] cannot be used directly for our purposes; it is secure only against *semi-honest adversaries*, and it *does not support proactivity*. For maliciously-secure MPC, we require a verifiable WRSS scheme that enables performing verifiable arithmetic operations over weighted shares. And for dynamic participation, this scheme must be proactive to allow passing intermediate computation state securely between the parties while accounting for an extra requirement; respecting the parties' weights.

WRSS is a non-linear scheme based on the Chinese Remainder Theorem (CRT). Weight is captured by assigning each party i a prime modulus p_i depending on its weight w_i, such that $2^{w_i}/(1+n^{-1}) \leq p_i < 2^{w_i}$. For a prime p_0 and secret s from field $\mathbb{F}_{p_0}$, sharing proceeds by computing a blinded secret $S = s + p_0 U_L$ (where U_L is a random integer from $\{1, \ldots, L\}$), and shares $s_i = S \bmod p_i$ for $i = \{1, \ldots, n\}$. This produces a share of size $O(w_i)$. Reconstruction uses CRT to solve for S and returns $s = S \bmod p_0$. WRSS works in the ramp setting; it has a reconstruction threshold W_r, i.e., any authorized set contains shares of total weight $\geq W_r$, and a privacy threshold W_p, i.e., any unauthorized set contains shares of total weight $\leq W_p$.

We build a *proactive WRSS* scheme by adding a reshare mechanism that refreshes the shares' randomness, thus producing fresh shares for the same secret while respecting the parties' weights. We do that by adapting the degree reduction protocol from [35] (used to reduce the growth of the blinding factor $U_L p_0$ when performing arithmetic operations over WRSS shares). In particular, the parties generate (using WRSS) two sets of shares for some random r, where each party shares its contribution to r among others and no single party knows r. A party obtains one share from each set denoted as r_i^0 and r_i^1. It uses r_i^0 to produce a share of $s - r$ and broadcasts it, allowing everyone to reconstruct a masked secret $s - r$. Then, a party computes a fresh share for the same secret as $s'_i = s - r + r_i^1 \bmod p_i$. Since each party receives r shares according to its weight, and computing s'_i is done with respect to p_i, s'_i has weight w_i as desired.

To address malicious security, we devise a verifiable WRSS (VWRSS) scheme, thus closing an open problem from [35]. Malicious behaviors cover distributing inconsistent shares (do not correspond to the same S) or shares that violate reconstruction correctness (so $S \notin [0, 2^{W_r})$), or reporting invalid shares during reconstruction. We support verifiability using similar ideas to [30], namely, publishing a commitment to S to ensure consistency, and a range proof π_s attesting that $S \in [0, 2^{W_r})$. The commitment is an efficient discrete-log based scheme with parameters collectively generated by the parties. The dealer computes π and a commitment to S (with randomness R) as $\mathsf{ct} = g^S h^R \bmod Q$, where $Q = \Pi_{i=1}^{n} q_i$ (now each party has a safe prime $q_i = 2p_i + 1$), and g and h are computed using CRT over the parties' contributions g_i and h_i.

For share $([s]_i, r_i)$, a party verifies π_s, then computes a commitment to $([s]_i, r_i)$ as $\mathsf{ct}_i = g_i^{[s]_i} h_i^{r_i} \bmod q_i$ and checks share consistency by verifying that $\mathsf{ct}_i = \mathsf{ct} \bmod q_i$. It accepts $([s]_i, r_i)$ only if both checks pass, and broadcasts a complaint otherwise. Since we deal with malicious adversaries, we add parameter-generation validity checks.

To support maliciously-secure dynamic participation, we build a *proactive VWRSS* by combining the two schemes above. Proactivity now, however, requires computing over *verifiable* shares (randomness shares and the old shares) to produce verifiable outputs. Also, we need a mechanism to prove that the two sets of randomness shares a party generates during resharing are for the same contribution r_i. Specifically, we want to employ commitments to prove that as it is more efficient than using equality zero-knowledge proofs. Enabling that requires additional modifications to our VWRSS scheme, including modifying its common parameters, and how the commitment parameters are generated and used. In fact, these changes are needed to enable MPC using VWRSS. We discuss these changes in Section 3.3, then in Section 3.4, we present a modified VWRSS scheme that implements these changes and show how to make it proactive.

We prove correctness and security of our proactive and verifiable WRSS schemes (definitions and proofs can be found in Appendices A and B, respectively).

Maliciously-secure weighted MPC with GOD and static participation. Here, we utilize our modified VWRSS scheme while resolving two challenges: enabling verifiable computing over shares by producing output shares and commitments to verify output correctness, and managing the blinding factor growth during computing to preserve reconstruction correctness. Under honest majority, MPC over verifiable shares achieves GOD against malicious adversaries.

To compute over VWRSS shares, we utilize the homomorphism of the commitment scheme and VWRSS. For example, addition can be performed locally by adding two shares together, and multiplying their commitments produces a valid commitment for the output. In Section 4.2, we describe how to perform all local operations, and generate valid commitments for their outputs, including negation, addition, scalar addition, and scalar multiplication.

VWRSS's homomorphism increases the blinding factor $U_L p_0$; if the blinded output exceeds 2^{W_r}, reconstruction fails. This increase is slow for local operations but significant for multiplication. Also, multiplying two shares locally and then running a degree reduction protocol to reduce the blinding factor (as done in [35]) will not work for us; the commitment scheme only supports additive homomorphism. We resolve this issue by performing multiplication using Beaver triples [7] and devising a verifiable degree reduction protocol. That is, parties hold verifiable shares of triples $(a, b, c = ab)$, and each multiplication gate consumes a triple. After each multiplication, the parties run the verifiable degree reduction protocol to bring the output share back to the correct range. This protocol employs our proactive VWRSS, specifically, refreshing the randomness of the output shares using the technique described previously, thus preserving parties' weights. The difference is that the first set of r shares has a blinding factor similar to the multiplication output, while the second set has a blinding factor of a fresh share. Also, since we have verifiable Beaver triple shares, multiplication produces valid commitments for the output shares.

Accordingly, in our *maliciously-secure weighted MPC*, servers collectively generate weighted Beaver triples during a preprocessing phase. Then, in the online phase, they perform the computation over VWRSS shares of the clients' inputs. We note that range proofs are only needed for fresh sharing (i.e., when sharing inputs during the input phase, and generating fresh randomness shares within degree reduction and Beaver triples). During the MPC computation, we do not generate range proofs for the output of the arithmetic operations (improving efficiency) since these operations do not violate reconstruction correctness (due to the use of degree reduction when needed), thus improving efficiency. Our protocol supports $W_p = W/3$, where W is the total weight of the servers, and $W_r = 2W_p + 8\lambda$, where λ is the security parameter.

Dynamic participation support. We build a *Weighted Fluid MPC* protocol that allows a dynamic set of weighted servers to perform the computation while supporting malicious security with GOD. In dynamic/fluid settings, as in [19, 38, 46], the computation is divided into rounds, each of which is handled by a server committee. This setting presents new challenges. It requires a handoff protocol to enable a round committee $\mathcal{P}_\ell$ to hand over the intermediate state to the next round committee $\mathcal{P}_{\ell+1}$ in a secure and verifiable way. This

protocol must also respect the weights of $\mathcal{P}_{\ell+1}$ members (which may differ from those of $\mathcal{P}_\ell$). Furthermore, performing multiplication over weighted shares requires care in this paradigm to avoid excessive interaction.

We utilize the randomness refreshing idea from above to build a verifiable handoff protocol. Recall that fresh shares of s are computed from the publicly revealed shares of $s - r$. Anyone, including $\mathcal{P}_{\ell+1}$, can reconstruct $s - r$. We utilize this observation to enable handoff; for some intermediate state value s shared among $\mathcal{P}_\ell$, $\mathcal{P}_\ell$ generates (using VWRSS) two sets of shares for the fresh randomness r. The first set has a blinding factor similar to s and is held by $\mathcal{P}_\ell$, while the second set has a smaller range, like a fresh share, and is held by $\mathcal{P}_{\ell+1}$. Parties in $\mathcal{P}_\ell$ mask their shares of s using their shares of r and reveal the shares of $s - r$. Parties in $\mathcal{P}_{\ell+1}$ reconstruct $s - r$ and use their shares of r to generate fresh shares of s as before. Since a member i in $\mathcal{P}_{\ell+1}$ holds a share of r with weight $w_{\ell+1,i}$, and the computation is done with respect to $p_{\ell+1,i}$ in a verifiable way, each member in $\mathcal{P}_{\ell+1}$ obtains a refreshed share of s based on its weight and a correct commitment. Furthermore, as the second set of r shares has a fresh blinding range, handoff achieves a simultaneous degree reduction.

For multiplication, indeed it can be done as before; $\mathcal{P}_\ell$ does that using a Beaver triple, and hands off the resultant shares to $\mathcal{P}_{\ell+1}$. However, this incurs additional interaction; parties in $\mathcal{P}_\ell$ must interact to generate the Beaver triple/use it for multiplication, and then interact with $\mathcal{P}_{\ell+1}$ during handoff. To reduce this excessive interaction, we augment multiplication within the handoff protocol, thus involving both committees, to achieve three goals at once; multiplication, degree reduction, and handoff. At a high level, both committees hold shares of the Beaver triple. $\mathcal{P}_\ell$ partially computes the multiplication, then hands over the state to $\mathcal{P}_{\ell+1}$ who finishes multiplication to obtain output shares in the desired range. We note that preprocessing in the fluid setting is divided among the rounds, on per-committee bases, rather than in one offline phase as before.

Overhead and security of our MPC protocols. We analyze computation and communication overhead of our protocols, and compare them to prior work. We also prove their security assuming honest majority and synchronous communication. These can be found in Sections 4 and 5, and Appendix D and E. We note that to enable constructing the simulator in our proofs, we add an additional layer to the circuit representing the computation in which the output shares are multiplied by shares of 1. Also, for the fluid setting, we consider the retroactive-adaptive adversary model [19], where corrupting a party in round ℓ means it is actively corrupt in round ℓ and passively corrupt in all prior rounds it participated in (so the adversary obtains the party state for these prior rounds).

1.2 Related Work

Weighted secret sharing (WSS). A trivial approach for WSS is virtualization; a party with weight w_i obtains w_i independent shares. Although it enables using any linear secret sharing scheme to perform MPC, this approach incurs a multiplicative cost; the share size is $O(w_i\lambda)$, and a party has to perform the computation over each share separately, thus penalizing parties of large weights.

Beimel et al. [8] instantiate WSS by constructing a circuit realizing a general access structure that is converted using a generic compiler to a circuit realizing a weighted access structure. They show an information theoretic scheme (with share size $n^{\log n}$, where n is the number of parties) and a computational one (with share size polynomial in n). Applebaum et al. [4] employ a slicing technique that transforms a general access structure into a weighted one with share size 1.5^n. Benhamouda et al. [11] instantiate WSS in the ramped setting using rounding techniques (with a share size depends on n) and wiretap channels (share size depends on the used wiretap channel techniques). Applebaum et al. [3] introduce succinct computational WSS, utilizing projective PRGs to generate shares of size $polylog(N)$, where $N = 2^n$. More recently, Farras et al. [34] show another computational WSS scheme with $polylog(n)$ share size and $O(n^2 \log n \log \sigma)$ public information (σ is the weight threshold) assuming subexponentially secure one-way functions. All these schemes are either complex or they break the algebraic structure of the secret, which limit their use in MPC.

We build on the WRSS scheme from [35] that has an additive overhead in the weight—the share size for a party of weight w_i is $O(w_i)$. WRSS does not support malicious security or proactivity. Shehata et al. [47] address verifiability of WRSS using a proof of modulus based on range proofs. However, these proofs are

not homomorphic, and so valid proofs during MPC computation cannot be generated by combining prior proofs of input shares. It also means that this scheme is not proactive. Our VWRSS scheme, which employs homomorphic commitments, supports verifiability and proactivity in an efficient way.

Other works, like Tonkikh et al. [48], achieve efficient WSS by employing virtualization but with reducing the magnitude of party weights while preserving the weighted access structure. Farras et al. [33] reduce the problem of weight reduction to the problem of finding the Chow parameters, and use an efficient solver to obtain smaller weights that preserve the access structure. These methods can be used in conjunction with our proactive VWRSS to improve efficiency; reducing the weights will reduce the share size and so the computation cost.

MPC in dynamic settings. YOSO [38] and Fluid MPC [19] pioneered this direction. YOSO MPC assumes stateless servers that are not known in advance. Followup work avoided the trusted setup of YOSO MPC [2, 15, 42], achieved constant-round computation [15], improved scalability via secret sharing packing [31], built a modular framework for general composability [17], and showed full stack protocol design in the adaptive asynchronous setting [24]. Fluid MPC allows servers (that are known at least before the round in which they want to participate begins) to dynamically join and leave based on their available resources. It works in the honest majority setting, as YOSO, but only supports security with abort. This was addressed in [28] achieving GOD at the expense of increased round complexity. Rachuri et al. [46] show a fluid MPC protocol with dishonest majority, and Bienstock et al. [12] devise protocols with linear communication cost per gate. David et al. [26] present a variation for computations represented as layered graphs, and a more recent work [27] relax this layered model, by allowing intra-committee interaction, to improve overhead. All these protocols assume unweighted servers.

MPC in weighted settings. MPC with weights [35] is the first work to support weighted configuration in a more efficient way than virtualization. As mentioned before, this protocol only offers semi-honest security under static participation.[3] A recent work [39] achieves weighted MPC with malicious security and GOD. It uses succinct computational secret sharing (SCSS) with linear homomorphism to compile a monotone Boolean circuit for weighted threshold access structures from [8]. This yields a share complexity polynomial in the number of parties n and independent of their weights. Although theoretically appealing, this construction inherits a limitation from [8]. Specifically, employing shallow monotone sorting networks with excessively large hidden constants that propagate into the resulting SCSS, rendering the scheme infeasible for a concrete instantiation. Furthermore, this work deals only with static participation and does not discuss whether proactivity can be supported. Additionally, malicious security of the developed MPC protocol is proven in the (f_{Rand}, f_{Beaver})-hybrid model, but the discussed instantiations for these functionalities are semi-honestly secure. Our work achieves malicious security with GOD with share complexity that is additive in the party weights (but concretely efficient), offers maliciously-secure instantiations of the randomness and Beaver triple generation subprotocols, and additionally supports proactivity and dynamic participation.

2 Preliminaries

We recall the WRSS scheme from [35]. Overview of secret sharing, commitments and non-interactive zero-knowledge proofs can be found in Appendix A.

Notation. We denote the natural numbers by $\mathbb{N}$, integers by $\mathbb{Z}$, integers modulo some q by $\mathbb{Z}_q$, and a field by $\mathbb{F}$. We use $|\cdot|$ for the size of a set, $[n]$ for $\{1, 2, \ldots, n\}$, $[s]_i$ for the ith share of a secret s, λ for the security parameter, pp for the public parameters, $\mathsf{negl}(\lambda)$ for a function negligible in λ, PPT for probabilistic polynomial time, and $\leftarrow\!\!\$$ for randomly choosing an element from a set.

[3] The work [36] employs WRSS [35] to build an efficient unweighted large-scale MPC protocol with semi-honest security (and static participation). The work [13] extends [36] to achieve malicious security with abort (so no GOD) using error detection, statistical bound checks, and triple sacrifice. They detect misbehavior existence but do not identify it (i.e., they do not develop a verifiable WRSS scheme).

Weighted ramp secret sharing (WRSS). Let p_0 be a prime of length λ bits, $\mathbb{F}_{p_0}$ be a field, n be the number of parties, $\mathcal{B}$ be the set of all authorized subsets, and $\bar{\mathcal{B}}$ be the set of all unauthorized subsets (for simplicity, we let a subset contain share indices). WRSS works in the ramp setting. It defines two thresholds (with a gap between them): a reconstruction threshold T, so any authorized subset $B \in \mathcal{B}$ contains shares with a total weight $\geq T$; and a privacy threshold t, so any unauthorized subset $\bar{B} \in \bar{\mathcal{B}}$ contains shares with a total weight $\leq t$. There is a trade-off between the threshold gap and the share size; when $T - t = \Theta(\lambda)$, the share size of a party i with weight w_i is $O(w_i)$.

WRSS chooses a set of prime moduli $p_1, \ldots, p_n$ from ranges that depend on the parties' weights. Specifically, a party with weight w_i has a prime modulus p_i such that $\frac{2^{w_i}}{(1+n^{-1})} \leq p_i < 2^{w_i}$.[4] This moduli set defines two quantities P_{max} and P_{min}, and it is required that $P_{max} < P_{min}$, where:

$$P_{max} = \max_{\bar{B} \in \bar{\mathcal{B}}} \left(\prod_{i \in \bar{B}} p_i \right) \leq 2^t \quad \text{and} \quad P_{min} = \min_{B \in \mathcal{B}} \left(\prod_{i \in B} p_i \right) \geq 2^{T - O(1)}$$

Since $t < T$, we have $P_{max} < P_{min}$. With an integer L chosen such that $(L+1)p_0 < P_{min}$, the algorithms (Share, Reconstruct) proceed as follows:

- Share: For a secret $s \in \mathbb{F}_{p_0}$, compute a blinded secret $S = s + p_0 U_L$, where U_L denotes an integer chosen uniformly at random from $[L]$. Then, compute the shares as $[s]_i = S \bmod p_i$ for $i \in [n]$.
- Reconstruct: Given an authorized share subset $B \in \mathcal{B}$, use CRT to find the Lagrange coefficients $\{\lambda_i\}_{i \in B}$ and compute $S = \sum_{i \in B} \lambda_i [s]_i \bmod P$, where $P = \prod_{i \in B} p_i$, then return $s = S \bmod p_0$.

Each λ_i is an integer that satisfies $\lambda_i \bmod p_i = 1$, and $\forall j \neq i$, $\lambda_i \bmod p_j = 0$. It can be computed as $\lambda_i = P_i P_i^{-1}$, where $P_i = \prod_{j \in B, j \neq i} P_j$, and P_i^{-1} is the inverse of P_i modulo p_i. WRSS is $2^{-\lambda}$-private when $L = 2^{t+\lambda}$, and perfectly correct when $T \geq t + 2\lambda + \Theta(1)$. We refer the reader to [35] for parameter configuration and correctness/security analysis.

3 Proactive and Verifiable WRSS

Proactive secret sharing [41,44] allows refreshing the shares of a secret such that new shares are not relevant to the old ones. That is, an adversary who compromises an unauthorized subset of the older shares, cannot combine them with any unauthorized subset of the new shares to learn any information about the secret. This notion defends against mobile adversaries who can corrupt and uncorrupt parties during protocol execution, or corrupt parties across different committees in dynamic settings. Verifiable secret sharing, on the other hand, defends against malicious adversaries who may not only control a subset of the share holders but also the dealer. A malicious dealer may distribute invalid shares or inconsistent ones that do not correspond to the same secret, and malicious share holders may report incorrect shares during reconstruction.

We require proactivity and verifiability for our weighted MPC protocols. Proactivity is a basic building block for degree reduction (needed after performing arithmetic operations over shares to preserve reconstruction correctness) and the handoff protocol (needed in the fluid setting to securely pass the computation state between committees). While verifiability enables building maliciously-secure weighted MPC protocols, which under honest majority achieve GOD. We introduce a *proactive and verifiable WRSS* scheme; we do that in stages, where we start with proactivity, then verifiability, and lastly combining these together.

3.1 Proactive WRSS

Generally, proactivity is achieved via resharing; parties holding shares of a secret interact to generate new shares of the same secret. Basically, they refresh the randomness underlying the shares. As we have weighted parties, resharing must also *respect the weights* of the share holders during resharing.

[4] As noted in [35], this range contains $2^{w_i}/(1+n)$ integers, among which asymptotically $2^{w_i}/((1+n)w_i)$ integers are primes. Thus, for large enough w_i (say $polylog(\lambda)$), there are enough primes within this range to choose an appropriate prime p_i.

Let λ be a security parameter, $P_1, \ldots, P_n$ be n parties with weights $\{w_i\}_{i\in[n]}$, W_p and $W_r = W_p + 4\lambda + \Theta(1)$ be the privacy and reconstruction thresholds for WRSS, respectively. Each P_i has a prime modulus p_i such that $2^{w_i}/(1+n^{-1}) \leq p_i < 2^{w_i}$.

Input: P_i holds a WRSS share $[s]_i$ of some secret $s \in \mathbb{F}_{p_0}$ with $L = 2^{W_p+\lambda}$.

Reshare: The parties execute the Reshare protocol as follows:

- Each P_i samples $r_i \leftarrow\$ \mathbb{F}_{p_0}$, shares r_i twice among all parties using WRSS.Share with $L = 2^{W_p+2\lambda}$ and $L = 2^{W_p+\lambda}$, respectively. Let the shares held by party P_j be $[r_i]^0_j$ and $[r_i]^1_j$, respectively (0 and 1 are used to distinguish the shares).
- Each P_i computes its shares of the (unknown) value r as $[r]^0_i = \sum_{j\in[n]}[r_j]^0_i$ and $[r]^1_i = \sum_{j\in[n]}[r_j]^1_i$.
- Each P_i negates $[r]^0_i$ by computing $[-r]^0_i = (2^{W_p+2\lambda}+1)p_0 - [r]^0_i \bmod p_i$, and uses that to mask $[s]_i$ by computing $[s-r]_i = [s]_i + [-r]^0_i \bmod p_i$. P_i broadcasts the share $[s-r]_i$.
- Each P_i reconstructs $s-r$, by invoking WRSS.Reconstruct over the revealed shares, and computes a fresh share as $[s]'_i = (s-r) + [r]^1_i \bmod p_i$.

Output: Each party has a fresh share $[s]'_i$ of the secret s.

Fig. 1. The proactive WRSS scheme.

The scheme. The idea behind proactive WRSS is refreshing the blinding factor, which represents the randomness of the shares. This is done by having the parties generate a random mask and use it to refresh their shares. We observe that an adaptation of the degree reduction protocol from [35], which they use to reduce the blinding factor range of the multiplication output to preserve correctness, achieves proactivity for WRSS. The difference is that the shares we want to reshare have smaller blinding factor range than [35] as no operations have been performed over them. This protocol utilizes the fact that despite its non-linearity, WRSS supports additive homomorphism.

Our scheme is shown in Figure 1, which assumes semi-honest adversaries (later we introduce verifiable WRSS that handles malicious adversaries). We use W_r and W_p to denote the reconstruction and privacy thresholds, respectively, where, in terms of WRSS notation, $W_r = T - O(1)$ and $W_p = t$.[5] Under semi-honest security, we have $W \geq W_r$, where W is the parties' total weight. The parties collectively generate a fresh sharing of an (unknown) random r, where each party contributes randomness r_i and shares it among all parties twice. Since at least one honest party contributes an honestly generated randomness r_i, the resulting sharing of r is statistically close to a uniform r (as we show in our security proof). This produces two sets of r shares with blinding ranges $2^{W_p+3\lambda}$ and $2^{W_p+2\lambda}$ (so L is $2^{W_p+2\lambda}$ and $2^{W_p+\lambda}$, respectively). They use the first set to mask the secret s as $s-r$, where they locally compute the shares $[s-r]_i$ and broadcast these shares so each party can locally reconstruct $s-r$. Then, they use the second set of r shares to obtain a refreshed share of s denoted as $[s]'_i$. Since each party receives randomness shares according to its weight, and the computation of $[s]'_i$ is done with respect to p_i, $[s]'_i$ will have weight w_i.

For the blinding range, as noted the first set of r shares has a blinding range $2^{W_p+3\lambda}$ instead of a fresh share range $2^{W_p+2\lambda}$. This is needed to ensure that the shares of $s-r$ are $2^{-\lambda}$ close to fresh sharing of a random value, and so they do not reveal any information about s (for more details, refer to the security proof). To ensure that the parties can safely reconstruct $s-r$ (whose blinding range is $2^{W_p+3\lambda}$), we choose $W_r = W_p + 4\lambda + \Theta(1)$. [6] Based on this choice of W_r, we have $W - W_p = \Theta(\lambda)$. At the same time, since the second set of r shares has an almost fresh-share blinding range of $2^{W_p+2\lambda}$, each $[s]'_i$ has an almost fresh-share

[5] We set $W_r = T - O(1)$, which ensures that every authorized set B, whose $\Pi_{i\in B}p_i \geq P_{min} \geq 2^{T-O(1)}$, can always reconstruct the secret. We require this modified bound for malicious security of VWRSS (see Appendix B.2). It is not required for proactive WRSS, as it is semi-honest, but is used throughout for notation consistency.

[6] As a party's share of r is the sum of the shares received from others, the blinding factors of the two shares of r are actually $n2^{W_p+3\lambda} = 2^{W_p+3\lambda+\log(n)}$ and $n2^{W_p+2\lambda} = 2^{W_p+2\lambda+\log(n)}$, respectively. For $W_r = W_p + 4\lambda + \Theta(1)$, reconstruction correctness is preserved as long as $\log n < \lambda$. Thus, for simplicity, we drop $\log(n)$ in the discussion.

blinding factor, thus preserving correctness. This also means that resharing can be applied over reshared shares without violating correctness.

Security and correctness. In Appendix B.1 we prove the following theorem. Specifically, we prove correctness and security of our proactive WRSS scheme (based on Definition 2 from Appendix A.2).

Theorem 1. *For $W_r \geq W_p + 4\lambda + \Theta(1)$ and $W - W_p = \Theta(\lambda)$, the proactive WRSS scheme is perfectly correct and statistically private (cf. Definition 2).*

3.2 Verifiable WRSS

Verifiable secret sharing addresses malicious adversaries. These include a dealer who distributes inconsistent shares (do not correspond to the same S) or shares that violate correctness (by having $S \notin [0, 2^{W_r})$), or parties who report invalid shares during reconstruction. In other words, and even if the dealer is malicious, any execution of the sharing phase determines a unique value that will be recovered during the reconstruction phase. We show a construction that achieves verifiability for WRSS, closing an open problem in [35].

The scheme. To achieve verifiability, we adapt ideas from [30] to the WRSS setting ([30] tackles verifiability for *unweighted* CRT-based secret sharing, the Asmuth-Bloom scheme [6]—no MPC). In particular, publishing a commitment to S to be used for share consistency checking, with commitment parameters and randomness are generated collectively by the parties, and a range proof attesting that $S \in [0, 2^{W_r})$. Moreover, to address rushing adversaries who may attempt to bias parameter generation, we employ knowledge of the discrete logarithm proofs as explained below.

Our VWRSS scheme is shown in Figure 2—we split the share phase across Share and Verify as we will need verify to be invoked separately in subsequent protocols. The dealer computes a commitment ct for the blinded secret S that will be used by the parties to check the consistency of their shares. To prevent the dealer from distributing valid/consistent shares for an $S \notin [0, 2^{W_r})$, the dealer also generates a range proof π attesting that the commitment is for an $S \in [0, 2^{W_r})$.[7] A party P_i will accept its share only if both checks pass; π is valid and the share is consistent based on ct. Similarly, during reconstruction, only shares that pass these checks are included in an authorized set B.

The commitment to S (with randomness R) is computed using parameters generated collectively by the parties (we refer to this commitment as aggregated commitment). Each party has a safe prime $q_i = 2p_i + 1$ and a generator g_i for the group G_{p_i} which is the subgroup of $\mathbb{Z}_{q_i}^*$ of order p_i.[8] It contributes to the generation of the random element h in the commitment so that no single party (including the dealer) knows the discrete log of h (with respect to g). This is needed to prevent the dealer from breaking the binding property of the commitment, and so making honest parties accept inconsistent shares. In detail, each party P_i samples $x_{ji} \leftarrow\$ \mathbb{Z}_{p_j}$, computes $g_j^{x_{ji}}$ along with a proof $\pi_{x_{ji}}$ attesting that it knows the exponent x_{ji}, and broadcasts these values. This proof prevents a rushing adversary who may select its $g^{x_{ij}}$ after seeing honest parties' contributions in a way that allows producing an h_i for which this adversary knows the discrete log (we explain how a rushing adversary can do that in Remark 1).

Once P_i receives contributions from all other parties, it computes $h_i = \prod_{j\in[n]} g_i^{x_{ij}} \bmod q_i$ by including only those for which $\pi_{x_{ji}}$ verifies correctly. The dealer computes all h_i in the same way, and samples randomness $R \leftarrow\$ \mathbb{Z}_P$ (where $P = \prod_{i\in[n]} p_i$), then extracts $r_i = R \bmod p_i$ for all parties. It uses CRT to compute (g, h), and computes a commitment to S as $\mathsf{ct} = g^S h^R \bmod Q$. Having g and h generated based on the individual parties' moduli allows these parties to extract commitments for their individual shares from ct. The dealer sends each party P_i its share $[s]_i$ and corresponding randomness r_i, and broadcasts (ct, π).

[7] One might argue that a dealer must also prove correct blinding of s, i.e., it used a U_L with $L = 2^{W_p+\lambda}$. Using a smaller L, although it satisfies $S \in [0, 2^{W_r})$, risks the privacy of s as shown in [35, 40]. Thus, a dealer is motivated to respect this value.

[8] In the range $\left[2^{w_i}/(1+n^{-1}), 2^{w_i}\right)$, which contains $2^{w_i}/(1+n)$ integers, it is conjectured that there are asymptotically $\frac{2C_2(2^{w_i})}{(1+n)(w_i)^2}$ Sophie-Germain primes (i.e., a prime p_i such that $q_i = 2p_i + 1$ is also a prime), where $C_2 \approx 0.66016$ is the twin prime constant. Again, for large enough w_i, there are enough Sophie-Germain primes in this range to choose p_i.

Same parameters as in Figure 1 but now each party P_i has $q_i = 2p_i + 1$. Also, let G_{p_i} be the subgroup of $\mathbb{Z}_{q_i}^*$ of order p_i with a generator g_i.

Share: To share a secret $s \in \mathbb{F}_{p_0}$ among $P_1, \ldots, P_n$, each P_i does the following:

- For $j \in [n]$, sample $x_{ji} \leftarrow\$ \mathbb{Z}_{p_j}$, compute $g_j^{x_{ji}} \bmod q_j$ and a proof $\pi_{j,x_{ji}}$ attesting that P_i knows the exponent x_{ji} of $g_j^{x_{ji}}$, then broadcast $(g_j^{x_{ji}}, \pi_{j,x_{ji}})$.
- Compute $h_i = \prod_{j \in [n']} g_i^{x_{ij}} \bmod q_i$, where $[n']$ is $[n]$ excluding all j values for which $\pi_{i,x_{ij}}$ does not verify.

Then, the dealer does the following:

- Sample $R \leftarrow\$ \mathbb{Z}_P$, where $P = \prod_{i \in [n]} p_i$.
- Compute each h_i as the parties did above.
- Use CRT to compute g and h by solving the systems of congruences: $g \equiv g_i \bmod q_i$ and $h \equiv h_i \bmod q_i$ for $i \in [n]$. Lastly, set $Q = \prod_{i \in [n]} q_i$.
- Share s among the parties by invoking WRSS.Share with $L = 2^{W_p + \lambda}$. Also, privately send each party P_i its corresponding $r_i = R \bmod p_i$.
- Compute a commitment to $S = s + U_L p_0$ as $\mathsf{ct} = g^S h^R \bmod Q$, then generate a range proof π attesting that $S \in [0, 2^{W_r})$, and broadcast (ct, π).

Verify: To verify share $([s]_i, r_i)$, party P_i does the following:

- Verify π and check that $g_i^{[s]_i} h_i^{r_i} \equiv \mathsf{ct} \bmod q_i$. If both checks pass, then accept the share. Otherwise, reject and broadcast $([s]_i, r_i)$ as a complaint (anyone can verify π, and if $([s]_i, r_i)$ is an inconsistent share, anyone can verify that using the same consistency check that P_i used since ct, g_i, and h_i are public).

Honest parties discard their shares if any valid complaint is issued.

Reconstruct: For an authorized set of shares B (where each share is accompanied with r_i), do the following:

- Verify π. If invalid, output $\perp$.
- For each $[s]_i \in B$, if $g_i^{[s]_i} h_i^{r_i} \not\equiv \mathsf{ct} \bmod q_i$, exclude $[s]_i$ from B.
- If B is still an authorized set (its total weight is $\geq W_r$), reconstruct s by invoking WRSS.Reconstruct. Otherwise, output $\perp$.

Fig. 2. The VWRSS scheme.

Verifying share validity proceeds by having each party verify π and then compute a commitment to its share $([s]_i, r_i)$ as $\mathsf{ct}_i = g_i^{[s]_i} h_i^{r_i} \bmod q_i$. A share is valid if the proof verifies and that ct_i is equal to the commitment extracted from ct, i.e., $\mathsf{ct}_i = \mathsf{ct} \bmod q_i$. Any party that receives an incorrect share will advertise that to notify other parties.[9] The same checks are performed over the shares received during reconstruction (where each share is accompanied by r_i generated by the dealer). A malicious party may send a different r_i or $[s]_i$ than the ones it received. These will be detected since the consistency check will fail (a party does not know the discrete log of h_i since h_i is generated collectively by the parties, so it cannot break the binding property of the commitment).

Lastly, since we deal with malicious adversaries, we discuss verifying that the parties have generated their own parameters correctly. We assume a PKI so all parties are known to each other using certified public keys, and thus, saying P_i implicitly means knowing the public key of that party. Also, we assume the weights of the parties are known (one can imagine that the public key certificate also includes the weight). Thus, everyone is able to compute W_r and W_p and can verify that the modulus p_i a party broadcasts is a prime in the required range. Same for each $g_i^{x_{ij}}$ and h_i, anyone is able to check that these indeed are elements in

[9] As we assume a PKI, a dealer signs all messages (shares and the commitment/range proof) it sends. Thus, anyone can verify that indeed the share was sent by the dealer.

G_{p_i}. As a result, a malicious party who broadcasts any incorrect parameter will be excluded, and parameter computation (and thresholds as well) will be over the party set that contributed correct parameters.

Security and correctness. Security and correctness of VWRSS follow in a similar way to the proofs in [30], which are for the unweighted version. For completeness, in Appendix B.2, we prove the following theorem. Note that since VWRSS does not include resharing (which we add it later), W_r and W_p are the same as in WRSS, namely $W_r \geq W_p + 2\lambda + \Theta(1)$. However, since VWRSS handles malicious adversaries, we have $W \geq W_r + W_p$, and so $W - 2W_p = \Theta(\lambda)$.

Theorem 2. *For $W_r \geq W_p + 2\lambda + \Theta(1)$ and $W - 2W_p = \Theta(\lambda)$, VWRSS is perfectly correct, statistically private, and computationally complete (cf. Definition 3 in Appendix A.3).*

We note that our security proof of the aggregated commitment, correctness of commitment extraction, and the completeness property proof that captures the verifiability aspect, are more detailed/formal than [30]. Notably, [30] does not address the rushing adversary case when generating h_i, and so does not employ knowledge of the discrete log proofs.

Remark 1 (The rushing adversary case). To see the rushing adversary case, suppose the parties only broadcast their contributions $g_i^{x_{ij}}$. A rushing adversary, who aims to make some honest party P_i accept an inconsistent share, can wait until all parties (except one corrupt party, say P_k) broadcast their $g_i^{x_{ij}}$. Then, it computes $h_i^* = \prod_{j\in[n]-\{k\}} g_i^{x_{ij}}$, chooses $x \in \mathbb{Z}_{p_i}$ and instructs P_k to send $g_i^x (h_i^*)^{-1}$ to P_i (while behaving honestly with the rest of the parties). This will cause P_i to compute $h_i = h_i^* g_i^x (h_i^*)^{-1} = g_i^x$. Since the adversary knows x, it can open the commitment to different blinded secrets; suppose the adversary generates a blinded secret S (and randomness R) from which the shares $\{[s]_j, r_j\}_{j\in[n]}$ are generated. The adversary can find a different $[s']_i$ (corresponding to a different blinded secret S') and r'_i (corresponding to a different randomness R') such that $R' \equiv R \bmod q_k$ for $k \in [n] - \{i\}$ and $([s]_i - [s']_i) = x(r'_i - r_i) \bmod p_i$, and so $g_i^{[s]_i} h_i^{r_i} \equiv g_i^{[s']_i} h_i^{r'_i} \bmod q_i$. The adversary can then send the inconsistent share $([s']_i, r'_i)$ to P_i, which P_i would accept as $\mathsf{ct}_i = g_i^{[s']_i} h_i^{r'_i} = g_i^{[s]_i} h_i^{r_i}$.

Remark 2 (The computational and statistical security parameters). Note that λ is the statistical security parameter inherited from WRSS [35], where p_0 is around 2^λ, $\mathbb{F}_{p_0}$ is the field of secrets, and $W_r - W_p = \Theta(\lambda)$. We have a separate computational security parameter κ and we require each $w_i \geq \kappa$ (and so each p_i and q_i, whose sizes depend on w_i, are at least around 2^κ). This is needed for the security of the commitment scheme. Accordingly, asymptotic complexity in terms of weights reflects the computational security parameter κ (this is why we do not mention κ separately). We note that the threshold signature and encryption applications of (semi-honest) WRSS in [35] also require large enough weights and it was noted that weights can be multiplied by a large enough factor to achieve that. Note that λ could be chosen to be equal to kappa (as in [35]), and it could be the case that $\lambda > \kappa$ since λ represents the input length in our MPC protocols (these private inputs are from $\mathbb{F}_{p_0}$).

3.3 On the Use of VWRSS in our MPC Protocols

Although it is of independent interest, our goal of developing VWRSS is to build maliciously-secure weighted MPC. Beside verifying input shares, this also requires verifying correctness of arithmetic operations performed over these shares during the computation. That is, an arithmetic operation will not only produce output shares but also commitments to verify output correctness. We observe that the aggregated commitment $\mathsf{ct} = g^S h^R \bmod Q$ is additively homomorphic. This allows us to compute valid aggregated commitments for the output of (linear) operations from input aggregated commitments. Also, by employing Beaver triples for multiplication, we can also utilize this additive homomorphism to compute valid commitments for multiplication using linear operations.

Furthermore, to support maliciously-secure weighted MPC in the fluid setting, we require a proactive VWRSS. To improve efficiency, we want to employ commitments to achieve verifiable proactivity. Specifically, use commitments to check that both sets of the random mask shares in Reshare are for the same r, thus

addressing malicious adversaries who may distribute inconsistent sets (as we describe in the next section). This is more efficient than employing zero-knowledge equality proofs. Enabling that requires an additional set of parameters to be used with commitments, along with a slightly different approach for sampling the randomness R used for the aggregated commitments. In particular, we will have an additional safe prime $q_0 = 2p_0 + 1$. So, the commitment parameters will include an additional tuple $\{g_0, h_0, q_0\}$, where $g_0 \in \mathbb{Z}^*_{q_0}$ is a generator for subgroup G_{p_0}, and h_0 is generated in a similar way as h_i but with $h_0 \in \mathbb{Z}^*_{q_0}$. For the randomness R, the dealer first samples a random $r \leftarrow\$\, \mathbb{F}_{p_0}$, computes a blinded r as $R \leftarrow\$\, r + U_{2^{W+\lambda}} p_0$, and then computes $r_i \equiv R \bmod p_i$ for $i \in [n] \cup \{0\}$. Choosing R in this way results in commitments that are $2^{-\lambda}$ statistically hiding (instead of perfectly hiding as before).[10]

Lastly, we note that range proof generation and verification are only needed for fresh sharing, i.e., during the input phase when distributing shares of the private inputs, when sharing the random mask during reshare, or when generating Beaver triples as we discuss later. These are not needed during the MPC computation since we will show that all arithmetic operations performed over the shares will not violate the reconstruction correctness bound.

In the next section, we modify VWRSS to reflect these changes, and then show how to make it proactive.

3.4 Proactive VWRSS

We devise a proactive VWRSS scheme by combining our proactive WRSS and VWRSS while adding the changes discussed in the previous section. This scheme must: ensure that parties generate valid shares for the random mask x (we use x instead of r, and $r_{s,i}$ in the share instead of r_i to avoida notation overloading); ensure that the two sets of shares are for the same x; and enable the parties to compute a valid commitment for the refreshed blinded secret S' without knowing the older blinded secret S. Note that there is no need to generate a range proof for S' since the random masks are within the valid range, leading to an S' within this range. We show a proactive VWRSS scheme that achieves these requirements by modifying VWRSS and then presenting a verifiable Reshare protocol.

The scheme. It is shown in Figure 3, which highlights the modifications in the setup phase (with respect to VWRSS from Figure 2) in red. Now, p_0 is associated with a safe prime q_0, and a generator g_0 for G_{p_0}. Also, all parameters are generated in one setup phase and will be reused for all sharing instances.

We discuss how the reshare protocol in the figure resolves the issues outlined earlier. First, to ensure valid sharing of the mask x, it is not enough to have valid shares of each x_i individually. The parties add these shares together (rather than invoking pVWRSS.Reconstruct), thus they must add shares coming from the same party subset to produce consistent random masks. To achieve that, the parties monitor all complaints and populate a set V that includes only those who distributed valid shares to *everyone* for *both* sharing sets of x. Honest parties, whose total weight is $\geq W_r$, will distribute valid shares (and cannot be framed of doing otherwise) and will be in V allowing for successful reconstruction of $s - x$ (and so generating fresh shares).

Second, we employ a check over the commitments to the blinded x_i, second condition in the AND statement under the second step in the figure, to ensure that the sharing instances $\{[x_i]^0_j\}_{j\in[n]}$ and $\{[x_i]^1_j\}_{j\in[n]}$ are for the same x_i. This check correctness can be found in Appendix B.3. Briefly, when the two share sets are for the same x_i, the blinded secrets X^0_i and X^1_i have a difference that is a multiple of p_0 (recall that $X^0_i = x_i + U_L p_0$ and $X^1_i = x_i + U'_L p_0$). The dealer also samples $R^0_{x_i}, R^1_{x_i} \leftarrow\$\, r_{x_i} + U_{2^{W+\lambda}} p_0$ with a difference $R^0_{x_i} - R^1_{x_i} = (r_{x_i} - r_{x_i}) + (U_{2^{W+\lambda}} - U'_{2^{W+\lambda}})p_0$ that is also a multiple of p_0. (This is inspired by [50] which uses shares of 0 as the randomness for Pedersen commitments for their asynchronous complete secret sharing protocol; $R^0_{x_i}, R^1_{x_i} \leftarrow\$\, r_{x_i} + U_{2^{W+\lambda}} p_0$ are effectively blinded secrets of r_{x_i} with $L = 2^{W+\lambda}$). The ratio of these commitments produces $g^{X^0_i - X^1_i} h^{R^0_{x_i} - R^1_{x_i}} = g^{kp_0} h^{k'p_0}$, for some integers k and k', only if the same x_i using the same r_{x_i} has been shared. By introducing (g_0, h_0, q_0) associated with p_0 into the commitment, we can perform this difference check for X^0_i and X^1_i in the exponent of g_0 and for $R^0_{x_i}$ and $R^1_{x_i}$ in the exponent of

[10] Intuitively, this is because for any adversary controlling a subset B with total weight $\leq W_p$, the distribution of the randomness modulo $P_B = \prod_{i\in B} p_i$ is $2^{-\lambda}$-close to uniform rather than perfectly uniform. See Appendix B.3 for more details.

Same parameters as in Figure 2, with an additional prime p_0, and so $q_0 = 2p_0 + 1$, with G_{p_0} be the subgroup of $\mathbb{Z}^*_{q_0}$ of order p_0 and a generator g_0.

Setup: Each party P_i does the following:

- Generate x_{ji}, $g_j^{x_{ji}}$, and compute h_i as before while considering $j \in [n] \cup \{0\}$.
- Compute $h_0 = \prod_{j\in[n]} g_0^{x_{0j}} \bmod q_0$.
- (The dealer also does this step.) Use CRT to compute (g, h, Q) by solving the following system of congruences: $g \equiv g_j \bmod q_j$, $h \equiv h_j \bmod q_j$, for $j \in [n] \cup \{0\}$, and set $Q = q_0 \prod_{j\in[n]} q_j$.

Share, Verify, and Reconstruct are same as in VWRSS.

Reshare: Each party P_i holds a verified share $([s]_i, r_{s,i})$, where $L = 2^{W_p+\lambda}$, and a commitment ct_s to the blinded secret S. Each party P_i does the following:

- Sample $x_i \leftarrow\$\ \mathbb{F}_{p_0}$ and $r_{x_i} \leftarrow\$\ \mathbb{F}_{p_0}$ and share x_i twice (using pVWRSS.Share) with $L = 2^{W_p+2\lambda}$ and $L = 2^{W_p+\lambda}$ respectively, using randomnesses $R^0_{x_i}$ and $R^1_{x_i}$ sampled from $r_{x_i} + U_{2W+\lambda}p_0$, and parameters $\{g, h, Q\}$. Let these shares be $([x_i]^0_j, r^0_{x_i,j})$ and $([x_i]^1_j, r^1_{x_i,j})$, for $j \in [n]$, their respective commitments be $\mathsf{ct}^0_{x_i}$ and $\mathsf{ct}^1_{x_i}$, and their respective range proofs be $\pi^0_{x_i}$ and $\pi^1_{x_i}$ (as before, 0 and 1 are used to distinguish the share sets).
- Invoke pVWRSS.Verify on all shares received from all parties, and construct the set V that include all parties P_j for which:
$$\left(\{[x_i]^0_j\}_{j\in[n]}, \{[x_i]^1_j\}_{j\in[n]} \text{ are valid } \right) \wedge \left(\mathsf{ct}^0_{x_i}(\mathsf{ct}^1_{x_i})^{-1} \bmod Q \equiv 1 \bmod q_0\right)$$
- For $b \in \{0, 1\}$, compute the shares $[x]^b_i = \sum_{j\in V}[x_j]^b_i \bmod p_i$ and $r^b_{x,i} = \sum_{j\in V} r^b_{x_j,i} \bmod p_i$, and the respective commitments $\mathsf{ct}^b_x = \prod_{j\in V} \mathsf{ct}^b_{x_j} \bmod Q$.
- Compute the share negation and its respective commitment as follows:
$$\left([-x]^0_i, r^0_{-x,i}\right) = \left((2^{W_p+2\lambda}+1)p_0 - [x]^0_i \bmod p_i, (2^{W+\lambda}+1)p_0 - r^0_{x,i} \bmod p_i\right)$$
$$\mathsf{ct}^0_{-x} = g^{(2^{W_p+2\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} (\mathsf{ct}^0_x)^{-1} \bmod Q$$
- Mask $([s]_i, r_{s,i})$ with $([-x]^0_i, r^0_{-x,i})$ by computing $[s-x]^0_i = [s]_i + [-x]^0_i \bmod p_i$, and $r_{s-x,i} = r_{s,i} + r^0_{-x,i} \bmod p_i$. Then, compute the commitment $\mathsf{ct}^0_{s-x} = \mathsf{ct}_s\mathsf{ct}^0_{-x} \bmod Q$. Lastly, broadcast $([s-x]^0_i, r_{s-x,i})$ and ct^0_{s-x}.
- Reconstruct $s - x$ by invoking pVWRSS.Reconstruct (no range proof checking) over the revealed shares.
- Generate a refreshed share of s as $([s]'_i, r'_{s,i}) = \left((s-x) + [x]^1_i \bmod p_i, r^1_{x,i}\right)$, and compute the corresponding commitment as $\mathsf{ct}'_s = g^{s-x}\mathsf{ct}^1_x \bmod Q$.

Fig. 3. The proactive VWRSS scheme (pVWRSS).

h_0. Specifically, checking that the exponents of g and h are multiples of p_0 can be done by checking that $g^{kp_0}h^{k'p_0} = 1 \bmod q_0$ (follows using a similar argument from the proof of Lemma 4 in Appendix B.2).

Third, the parties can compute a valid commitment for the refreshed blinded secret S' using the commitments for the random mask shares based on the set V. Thus, S' has a commitment with respect to the parameters (g, h, Q).

Security and correctness. The security of our proactive VWRSS scheme follows by the security of proactive WRSS and VWRSS. The difference from VWRSS is that the modified aggregated commitment scheme achieves statistical hiding. In Appendix B.3, we prove the following theorem.

Theorem 3. *For $W_r \geq W_p + 4\lambda + \Theta(1)$ and $W - 2W_p = \Theta(\lambda)$, proactive VWRSS is perfectly correct, statistically private, and computationally complete.*

4 Maliciously-secure Weighted MPC

The weighted MPC protocol from [35] is secure against semi-honest adversaries; achieving malicious security was left as an open question. In this section, we close this gap and show a maliciously-secure weighted MPC

protocol with GOD by utilizing our proactive VWRSS scheme. This requires resolving two challenges: (1) enabling verifiable computing over shares by producing valid commitments for the output shares without accessing the secrets, and (2) devising a verifiable degree reduction protocol to reduce the blinding factor growth during the computation to preserve correctness. We address these challenges below starting with degree reduction. Our MPC protocol achieves malicious security for an adversary that controls a party subset with a total weight $< W/3$, where W is the servers' total weight, thus we have $W - 3W_p = \Theta(\lambda)$.

Additional notation. The blinding factor maximum size is denoted by α, i.e., $U_L p_0 \leq 2^\alpha$ and so $L = 2^{\alpha-\lambda}$ (p_0 is of size λ bits). S_α denotes a blinded secret with respect to the blinding range it uses, where $S_\alpha = s + U_{2^{\alpha-\lambda}} p_0$. Party P_i's share is viewed as a tuple $[s]_i^\alpha = (\mathsf{sh}, \mathsf{r}, \mathsf{ct}, \pi)$, where $\mathsf{sh} = s + U_{2^{\alpha-\lambda}} p_0 \bmod p_i$, $\mathsf{r} = r_{s,i}$, $\mathsf{ct} = g^{S_\alpha} h^{R_s} \bmod Q$, π is the range proof for S_α, and the dot notation is used to refer to its components, e.g., $[s]_i^\alpha.\mathsf{sh}$.

4.1 Degree Reduction

WRSS has homomorphic properties that allow adding and multiplying shares to produce correct shares of the result. However, this comes at the expense of increasing the blinding range of the output shares; if the blinded secret exceeds 2^{W_r}, reconstruction fails. Garg et al. [35] showed a degree reduction protocol to bring the output share back to the correct range. We utilize our pVWRSS scheme to convert this protocol into a verifiable one, which we refer to as WMPC.DegreeReduction. This protocol is the same as pVWRSS.Reshare in Figure 3, but uses different blinding ranges. In particular, the shares $[x_i]_j^0$ use 2^λ times the blinding range of the output shares (this is needed for security which we establish based on the integer masking lemma). While the shares $[x_i]_j^1$ use a blinding range of a fresh sharing, i.e., $\alpha = W_p + 2\lambda$.

At a high level, suppose the parties have shares $\{[s]_j^{\alpha'}\}_{j\in[n]}$ with $\alpha' > W_p + 2\lambda$. To bring these shares back to the range of $2^\alpha = 2^{W_p+2\lambda}$, the parties generate shares of the random mask x using two different blinding factor ranges, namely, $\{[x]_j^{0,\alpha'+\lambda}\}_{j\in[n]}$ and $\{[x]_j^{1,\alpha}\}_{j\in[n]}$. They use $\{[x]_j^{0,\alpha'+\lambda}\}_{j\in[n]}$ to mask $\{[s]_j^{\alpha'}\}_{j\in[n]}$, reconstruct $s - x$ in the clear, and locally add it to $\{[x]_j^{1,\alpha}\}_{j\in[n]}$ to obtain a share of s in the desired range. The underlying techniques of our pVWRSS scheme address malicious adversaries who may misbehave during degree reduction.

We reuse various parts of pVWRSS.Reshare as building blocks in our subsequent protocols. Specifically, we label its first four steps that generate two share sets of unknown random element x as WMPC.ShareRandDouble. We also denote generating one share set of unknown random x as WMPC.ShareRand. In summary, we have the following:

- WMPC.DegreeReduction $\left(\{[s]_i^{\alpha_1}\}_{i\in[n]}, \alpha_1, \alpha_2\right) \to \{[s]_i^{\alpha_2}\}_{i\in[n]}$: For input shares $\{[s]_i^{\alpha_1}\}_{i\in[n]}$ whose blinding range is α_1, it outputs refreshed shares $\{[s]_i^{\alpha_2}\}_{i\in[n]}$ for the same secret but with a blinding range α_2, such that $\alpha_2 < \alpha_1$.
- WMPC.ShareRandDouble$(\alpha_1, \alpha_2) \to (\{[x]_i^{\alpha_1}\}_{i\in[n]}, \{[x]_i^{\alpha_2}\}_{i\in[n]})$: Takes as input two blinding ranges α_1 and α_2, and outputs two share sets $\{[x]_i^{\alpha_1}\}_{i\in[n]}$ and $\{[x]_i^{\alpha_2}\}_{i\in[n]}$ for the same unknown random x.
- WMPC.ShareRand$(\alpha) \to \{[x]_i^\alpha\}_{i\in[n]}$: Takes as input a blinding range α, and outputs a set of shares $\{[x]_i^\alpha\}_{i\in[n]}$ for some unknown random x, using randomness $R_x \leftarrow\$\, \mathbb{Z}_P$ where $P = \prod_{i\in[n]} p_i$.

4.2 Computing Over Verifiable Shares

Now we show how to perform arithmetic operations over pVWRSS shares.

Negate, add, scalar add, and scalar multiply. We utilize the additive homomorphism of pVWRSS and commitments to perform these operations, see Figure 4. Each operation produces a verifiable output share with a commitment to the blinded result (detailed correctness analysis can be found in Appendix C.1). No range proofs are needed; starting with shares in the correct range, all these local operations (combined with degree reduction when needed) preserve the range of the blinding factor. That is, WMPC.scalarMul produces output shares with a blinding range that is 2^λ times larger than that of the input shares. Thus, in our

WMPC.Neg: To negate a share $[s]_i^\alpha$, locally compute $[-s]_i^\alpha.\mathsf{sh} = (2^{\alpha-\lambda}+1)p_0 - [s]_i^\alpha.\mathsf{sh} \bmod p_i$, $[-s]_i^\alpha.\mathsf{r} = (2^W + 1)p_0 - [s]_i^\alpha.\mathsf{r} \bmod p_i$, and $[-s]_i^\alpha.\mathsf{ct} = g^{(2^{\alpha-\lambda}+1)p_0} h^{(2^W+1)p_0} ([s]_i^\alpha.\mathsf{ct})^{-1} \bmod Q$.

WMPC.Add: To add two shares $[s_1]_i^\alpha$ and $[s_2]_i^\alpha$, locally compute $[s_1+s_2]_i^{\alpha+1}.\mathsf{sh} = [s_1]_i^\alpha.\mathsf{sh} + [s_2]_i^\alpha.\mathsf{sh} \bmod p_i$, $[s_1+s_2]_i^{\alpha+1}.\mathsf{r} = [s_1]_i^\alpha.\mathsf{r} + [s_2]_i^\alpha.\mathsf{r} \bmod p_i$, and $[s_1+s_2]_i^{\alpha+1}.\mathsf{ct} = ([s_1]_i^\alpha.\mathsf{ct})([s_2]_i^\alpha.\mathsf{ct}) \bmod Q$.

WMPC.scalarAdd: To add a scalar c to a share $[s]_i^\alpha$, locally compute $[c+s]_i^\alpha.\mathsf{sh} = c + [s]_i^\alpha.\mathsf{sh} \bmod p_i$, $[c+s]_i^\alpha.\mathsf{r} = [s]_i^\alpha.\mathsf{r} \bmod p_i$, and $[c+s]_i^\alpha.\mathsf{ct} = g^c [s]_i^\alpha.\mathsf{ct} \bmod Q$.

WMPC.scalarMul: To multiply a share $[s]_i^\alpha$ by a scalar c, locally compute $[cs]_i^{\alpha+\lambda}.\mathsf{sh} = c[s]_i^\alpha.\mathsf{sh} \bmod p_i$, $[cs]_i^{\alpha+\lambda}.\mathsf{r} = c[s]_i^\alpha.\mathsf{r} \bmod p_i$, and $[cs]_i^{\alpha+\lambda}.\mathsf{ct} = ([s]_i^\alpha.\mathsf{ct})^c \bmod Q$.

Fig. 4. Performing arithmetic operations over pVWRSS shares (no range proof is needed for all results, i.e., $\pi = \perp$ in the share tuple).

WMPC protocol, every WMPC.scalarMul operation is followed by a WMPC.DegreeReduction to reduce the blinding range. Furthermore, WMPC.Add produces output shares with a blinding range that is 2 times that of the input shares; such a slight increase does not violate correctness. In Remark 3, we discuss handling accumulated growth across multiple add gates.

Multiplication. While locally multiplying individual shares of s_1 and s_2 produces a share of $s_1 s_2$ (with a degree reduction performed over the resultant share), this will not work for us as we operate on verifiable shares. The commitment scheme supports only additive homomorphism, so we cannot generate valid commitments for the output. We resolve this issue by employing Beaver triples [7] that transform multiplication into summation, allowing us to compute a valid commitment for the multiplication result from the input commitments.

Generating verifiable Beaver triples. A Beaver triple is a tuple of the form $(a, b, c = ab)$ shared among all parties (no party knows the triple itself). Triples are usually generated during a preprocessing phase, independent of the inputs, and used during the online phase of the protocol where each multiplication gate consumes a triple. To multiply shares of two secrets s_1 and s_2, the parties mask them with the shares of a and b, respectively, then reveal these masked shares, allowing everyone to reconstruct $s_1 - a$ and $s_2 - b$. Then, party P_i computes the ith share of $s_1 s_2$ as $[s_1 s_2]_i = (s_1 - a)[s_2]_i + (s_2 - b)[a]_i + [ab]_i$.

Figure 5 shows our verifiable Beaver triple generation protocol. The parties execute n instances of WMPC.ShareRand to generate verifiable shares of random elements $\{a_k\}_{k\in[n]}$. The parties add their shares of $\{a_k\}_{k\in[n]}$ to obtain a share of $a = \sum_{k\in[n]} a_k$. The parties also execute an instance of WMPC.ShareRand to generate verifiable shares of random element b. Generating verifiable shares of $c = ab$ is the tricky part. Recall that $b = \sum_{i\in V} b_i$, where the set V (identified within pVWRSS.Verify) consists of parties P_i who distributed valid shares of b_i. Each P_i broadcasts *only* the commitment of the sharing of $a_k b_i$ for all $k \in [n]$ (where a_k is the random secret associated with a previous invocation of WMPC.ShareRand and b_i is P_i's contribution to b generated by WMPC.ShareRand). Since P_i knows the blinded secret used in sharing b_i, these commitments are computed as in scalar multiplication (where $[a_k]_i^{\alpha+\lambda}.\mathsf{ct} = g^{(A_k)_{\alpha+\lambda}} h^{R_{a_k}} \bmod Q$ and $(B_i)_{\alpha+\lambda}$ is the blinded secret for b_i) as we show in Appendix C.2:

$$\mathsf{ct}_{a_k b_i} \equiv \left([a_k]_i^{\alpha+\lambda}.\mathsf{ct}\right)^{(B_i)_{\alpha+\lambda}} \equiv g^{(A_k B_i)_{2\alpha+2\lambda}} h^{(R_{a_k}(B_i)_{\alpha+\lambda})} \bmod Q$$

Note that this step can be performed in parallel with the second round of communication of WMPC.ShareRand and WMPC.ShareRandDouble invocations to reduce round complexity. For clarity, we keep them separate in Figure 5.

P_i verifies each of these commitments as shown in step 3 in the figure. If a particular commitment $\mathsf{ct}_{a_k b_j}$ fails this check, P_i broadcasts $[b_j]_i^{\alpha+\lambda}.\mathsf{sh}$ and $[a_k]_i^{\alpha+\lambda}.\mathsf{ct}$ as a complaint so all parties remove P_j from the set V. To ensure that the $\mathsf{ct}_{a_k b_i}$s can be combined into a commitment for the sharing of ab, each P_i recomputes

WMPC.GenBeaver:

1. The parties invoke the following in parallel, where $\alpha = W_p + 2\lambda$:
   - For each $k \in [n]$, $\mathsf{WMPC.ShareRand}(\alpha + \lambda)$ to obtain $\{[a_k]_j^{\alpha+\lambda}\}_{j\in[n]}$.
   - $\mathsf{WMPC.ShareRand}(\alpha + \lambda)$ to obtain $\{[b]_j^{\alpha+\lambda}\}_{j\in[n]}$. Each P_i stores $(B_i)_{\alpha+\lambda}$ used to generate the shares $\{[b_i]_j^{\alpha+\lambda}.\mathsf{sh} = (B_i)_{\alpha+\lambda} \bmod p_j\}_{j\in[n]}$. Let the set of valid contributions for this random sharing be V.
   - $\mathsf{WMPC.ShareRandDouble}(2\alpha + 3\lambda, \alpha + 2\lambda)$ to obtain $\{[x]_j^{2\alpha+3\lambda}\}_{j\in[n]}$ and $\{[x]_j^{\alpha+2\lambda}\}_{j\in[n]}$.
2. Each party P_i, for $i \in [n]$ computes its share $[a]_i^{\alpha+\lambda}$ as $[a]_i^{\alpha+\lambda}.\mathsf{sh} = \sum_{k\in[n]}[a_k]_i^{\alpha+\lambda}.\mathsf{sh} \bmod p_i$, $[a]_i^{\alpha+\lambda}.\mathsf{r} = \sum_{k\in[n]}[a_k]_i^{\alpha+\lambda}.\mathsf{r} \bmod p_i$, and $[a]_i^{\alpha+\lambda}.\mathsf{ct} = \prod_{k\in[n]}[a_k]_i^{\alpha+\lambda}.\mathsf{ct} \bmod Q$.
3. Each party $P_i \in V$ computes and broadcasts the following commitments:
$$\mathsf{ct}_{a_k b_i} = \left([a_k]_i^{\alpha+\lambda}.\mathsf{ct}\right)^{(B_i)_{\alpha+\lambda}} \bmod Q, \qquad \forall\ k \in [n]$$
4. Each party P_i, for $i \in [n]$, checks whether:
$$\left(g_i^{[a_k]_i^{\alpha+\lambda}.\mathsf{sh}} h_i^{[a_k]_i^{\alpha+\lambda}.\mathsf{r}}\right)^{[b_j]_i^{\alpha+\lambda}.\mathsf{sh}} \stackrel{?}{\equiv} \mathsf{ct}_{a_k b_j} \bmod q_i \quad \forall k \in [n], \forall j \in V$$
   If this check fails for some $\mathsf{ct}_{a_k b_j}$, P_i broadcasts $([a_k]_i^{\alpha+\lambda}.\mathsf{ct}, [b_j]_i^{\alpha+\lambda}.\mathsf{sh})$ as a complaint and all parties remove j from V.
5. Each party P_i, for $i \in [n]$, recomputes its share $[b]_i^{\alpha+\lambda}$ based on the updated set V, and computes its share $[ab]_i^{2\alpha+2\lambda}$ as:
$$[ab]_i^{2\alpha+2\lambda}.\mathsf{sh} = \sum_{k\in[n]} ([a_k]_i^{\alpha+\lambda}.\mathsf{sh})([b]_i^{\alpha+\lambda}.\mathsf{sh}) \bmod p_i$$
$$[ab]_i^{2\alpha+2\lambda}.\mathsf{r} = \sum_{k\in[n]} ([a_k]_i^{\alpha+\lambda}.\mathsf{r})([b]_i^{\alpha+\lambda}.\mathsf{sh}) \bmod p_i$$
$$[ab]_i^{2\alpha+2\lambda}.\mathsf{ct} = \prod_{k\in[n]}\prod_{j\in V} \mathsf{ct}_{a_k b_j} \bmod Q$$
6. Each party P_i then invokes $\mathsf{WMPC.DegreeReduction}$ over $[ab]_i^{2\alpha+2\lambda}$ using its mask shares $([x]_i^{2\alpha+3\lambda}, [x]_i^{\alpha+2\lambda})$ to obtain $[ab]_i^{\alpha+2\lambda}$ with a blinding range $2^{\alpha+2\lambda}$.

Fig. 5. Generating verifiable Beaver triples.

its share of b based on the updated set V. P_i uses the individual commitments $\mathsf{ct}_{a_k b_j}$ and the individual shares $[a_k]_i^{\alpha+\lambda}$ and $[b]_i^{\alpha+\lambda}$ to compute $[ab]_i^{2\alpha+2\lambda}.\mathsf{ct}$ and $[ab]_i^{2\alpha+2\lambda}.\mathsf{sh/r}$ respectively, as shown in the figure.

The parties then invoke $\mathsf{WMPC.DegreeReduction}$ over $\{[ab]_j^{2\alpha+2\lambda}\}_{j\in[n]}$ to reduce the blinding range to $\alpha + 2\lambda = W_p + 4\lambda$, resulting in a Beaver triple $\{[a]_j^{\alpha+\lambda}, [b]_j^{\alpha+\lambda}, [ab]_j^{\alpha+2\lambda}\}_{j\in[n]}$. As the blinding range of the pre-reduced shares is $2\alpha + 2\lambda = 2W_p + 6\lambda$, the shared random double must have as inputs the blinding range $\alpha_1 = 2W_p + 7\lambda$ and $\alpha_2 = W_p + 2\lambda$. To ensure that any authorized subset can successfully reconstruct the masked value, we require $W_r > 2W_p + 7\lambda$, which results in having $W_r = 2W_p + 8\lambda$. The requirement $W - 3W_p = \Theta(\lambda)$ results from supporting GOD in the malicious setting. GOD requires that honest parties form an authorized set, i.e., $W - W_p \leq W_r$. Combining this with $W_r = 2W_p + 8\lambda$ gives $W \leq 3W_p + 8\lambda$, so a corruption threshold of $W/3$.

We note the following about the generation of the shares of a. If we use one $\mathsf{WMPC.ShareRand}$ instance, this means that a party fully controls its contribution to a, denoted as a_k. This will enable an attacker to utilize the commitment $\mathsf{ct}_{a_k b_i}$ to learn g^{b_i} by choosing its contribution in a way that makes $\mathsf{ct}_{a_k} = g$ (i.e $A_k = 1$ and $R_{a_k} = 0$), which would enable this adversary to reveal information about the multiplications inputs when a Beaver triple is used for multiplication. Thus, and as shown in the figure, we require all parties to contribute to every single a_k by having them execute n invocations of $\mathsf{WMPC.ShareRand}$ instead of one.

WMPC.Mul: Given two sets of shares $\{[s_1]_j^\alpha\}_{j\in[n]}$ and $\{[s_2]_j^\alpha\}_{j\in[n]}$, and Beaver triple $\left(\{[a]_j^{\alpha+\lambda}\}_{j\in[n]}, \{[b]_j^{\alpha+\lambda}\}_{j\in[n]}, \{[ab]_j^{\alpha+2\lambda}\}_{j\in[n]}\right)$, the parties compute $\{[s_1 s_2]_j^{\alpha+2\lambda}\}_{j\in[n]}$ as follows. Each P_i invokes:

1. WMPC.Neg($[a]_i^{\alpha+\lambda}$) and WMPC.Neg($[b]_i^{\alpha+\lambda}$) to obtain $[-a]_i^{\alpha+\lambda}$ and $[-b]_i^{\alpha+\lambda}$.
2. WMPC.Add($[s_1]_i^\alpha, [-a]_i^{\alpha+\lambda}$) and WMPC.Add($[s_2]_i^\alpha, [-b]_i^{\alpha+\lambda}$) to obtain $[s_1-a]_i^{\alpha+\lambda}$ and $[s_2-b]_i^{\alpha+\lambda}$ and broadcast these shares.
3. pVWRSS.Reconstruct $\left(\{[s_1-a]_j^{\alpha+\lambda}\}_{j\in[n]}\right)$ to obtain s_1-a, and pVWRSS.Reconstruct $\left(\{[s_2-b]_j^{\alpha+\lambda}\}_{j\in[n]}\right)$ to obtain s_2-b.
4. WMPC.scalarMul $\left([a]_i^{\alpha+\lambda}, s_2-b\right)$ to obtain $[as_2-ab]_i^{\alpha+2\lambda}$ and WMPC.scalarMul $\left([s_2]_i^\alpha, s_1-a\right)$ to obtain $[s_1s_2-as_2]_i^{\alpha+\lambda}$.
5. WMPC.Add $\left([s_1s_2-as_2]_i^{\alpha+\lambda}, [as_2-ab]_i^{\alpha+2\lambda}\right)$ to obtain $[s_1s_2-ab]_i^{\alpha+2\lambda}$.
6. WMPC.Add $\left([s_1s_2-ab]_i^{\alpha+2\lambda}, [ab]_i^{\alpha+2\lambda}\right)$ to obtain $[s_1s_2]_i^{\alpha+2\lambda}$.

Fig. 6. Verifiable multiplication using Beaver triples (recall that a share $[\cdot]$ is a tuple including the share value and the commitment. Again, no range proofs are need, so $\pi=\perp$ for the output shares).

This guarantees that the individual commitments $\mathsf{ct}_{a_k b_i}$ hide the honest contributions (as we show in the security proof in Appendix D.2).

Multiplication during the online phase. Figure 6 shows the WMPC.Mul protocol. Party P_i with shares $[s_1]_i^\alpha$ and $[s_2]_i^\alpha$,[11] along with verifiable Beaver triple shares $\{[a]_i^{\alpha+\lambda}, [b]_i^{\alpha+\lambda}, [ab]_i^{\alpha+2\lambda}\}$ computes $[s_1-a]_i^{\alpha+\lambda}$ and $[s_2-b]_i^{\alpha+\lambda}$ and broadcasts them. This allows all parties to reconstruct s_1-a and s_2-b. Each P_i then computes its share $[s_1s_2]_i^{\alpha+2\lambda}$ using WMPC.Add and WMPC.scalarMul as follows (this is why we use $\alpha+2\lambda$ when generating the shares of ab):

$$[s_1s_2]_i^{\alpha+2\lambda} = (s_1-a)[s_2]_i^\alpha + (s_2-b)[a]_i^{\alpha+\lambda} + [ab]_i^{\alpha+2\lambda} \bmod p_i$$

This process also generates a correct commitment for $\{[s_1s_2]_j^{\alpha+2\lambda}\}_{j\in[n]}$ (details are in Appendix C.1). As shown, multiplication produces shares with a blinding range within $2^{\alpha+2\lambda} = 2^{W_p+4\lambda}$. Thus, a degree reduction must be performed after each multiplication gate to reduce that, and so preserve correctness.

4.3 Putting it All Together

We now present our maliciously-secure weighted MPC protocol with GOD. A set of unweighted clients $\mathcal{C}$ possess private inputs for a circuit C representing the function f to be computed. They share these inputs among a set of servers who will evaluate C over these shares. Similar to prior work [5,21–23], for any corrupt client that distribute invalid shares of its input, this will be detected by the servers (by verifying broadcast complaints). Servers then replace this client's input with some predetermined default value and continue the protocol.

Both clients and servers run pVWRSS.Setup to obtain the required parameters; servers do that before the preprocessing phase, while $\mathcal{C}$ can do that anytime before the output phase. Let the privacy and reconstruction thresholds for the clients be $W_p^{\mathcal{C}}$ and $W_r^{\mathcal{C}}$, respectively. We note that while unweighted clients means having unit weights, for the security of our VWRSS, a unit weight means ha weight of λ. This is needed to preserve commitment security as those clients handle the output phase in which they receive VWRSS shares of the output. Nonetheless, efficiency-wise, a weight of λ means the size of the share each client obtains is λ bits. One can easily extend this to support weighted clients by generating their pVWRSS parameters according to their weights (similar to the servers, the minimum weight is λ as discussed before).

[11] The blinding ranges of the shares of s_1 and s_2 may be slightly different (by $2^{O(log(n))}$) due to being the output of *prior* WMPC.DegreeReduction invocations with random double sharings generated via different V sets. This difference is small compared to their blinding ranges. Hence, we can assume that they have the same α.

Preprocessing Phase: The servers generate Beaver triples needed for all multiplication gates using WMPC.GenBeaver. They also generate all random mask sharing sets needed for all degree reductions by invoking:

- WMPC.ShareRandDouble($\alpha_1 = W_p + 5\lambda, \alpha_2 = W_p + 2\lambda$) for a number of times equals to the total number of multiplication gates.
- WMPC.ShareRandDouble($\alpha_1 = W_p + 4\lambda, \alpha_2 = W_p + 2\lambda$) for a number of times equals to the total number of scalar multiplication gates.

Online Phase: Clients share their inputs among the servers:

- Each client invokes pVWRSS.Share with $L = 2^{W_p+\lambda}$ (so $\alpha = W_p + 2\lambda$) to share its input among the servers.
- Each server verifies every share it receives by invoking pVWRSS.Verify. For incorrect shares (for which complaints are found to be valid), servers replace the corrupt client input with some pre-determined default value.

The servers compute C over these shares:

- Addition: Each server locally invokes WMPC.Add.
- Negation: Each server locally invokes WMPC.Neg.
- Scalar addition: Each server locally invokes WMPC.scalarAdd.
- Scalar multiplication: Each server locally invokes WMPC.scalarMul, then invokes WMPC.DegreeReduction (consuming a random double sharing) to reduce the output blinding range from $\alpha' = W_p + 3\lambda$ to $\alpha = W_p + 2\lambda$.
- Multiplication: Servers run WMPC.Mul (consuming a Beaver triple), then WMPC.DegreeReduction (consuming a random double sharing) to reduce the output blinding range from $\alpha' = W_p + 4\lambda$ to $\alpha = W_p + 2\lambda$.

Output Reconstruction Phase: Once circuit evaluation is complete, servers execute WFMPC.FluidHandoff (which will be explained in the next section) to hand over their internal state securely to the clients $\mathcal{C}$. $\mathcal{C}$ then jointly generate a sharing of 1 and a fresh Beaver triple. $\mathcal{C}$ then multiply their output shares with the shares of 1 using this Beaver triple by invoking WMPC.Mul. (This multiplication by 1 is needed for the simulator construction in the security proof.[12]) Finally, $\mathcal{C}$ broadcast their resultant shares, allowing all clients to recover the final output by invoking pVWRSS.Reconstruct.

Remark 3 (Degree reduction after addition). In [35], it is argued that degree reduction is not needed after addition due to the slow growth of the blinding range. This, however, assumes a polynomial number of additions of *fresh* shares, which might not be always the case. For example, a large number of sequential additions would lead to accumulative increase in the blinding range, producing out-of-range shares even before performing a single multiplication. Performing degree reduction after each addition avoids such a scenario. To reduce overhead, parties can track the blinding range growth after additions and perform degree reductions when necessary. As we detail in Appendix C.3, for any gate, if the input wire share's blinding range is $< 2^{W_p+2\lambda+\lambda/4}$, the output wire share's blinding range (after degree reduction, if needed) is also $< 2^{W_p+2\lambda+\lambda/4}$. Intuitively, this holds because of our choice of $W_r = 2W_p + 8\lambda$, and the extra $\lambda/4$ slack in the bound absorbs any small increases due to additions or mask generation. This bound ensures that no share goes out-of-range and that any wire share can be safely fed into any arithmetic gate. Furthermore, since the circuit to be computed is known ahead of time, the parties can determine which additions would require a degree reduction, and so preprocess additional random double sharings for these degree reductions.

Security. In Appendix D, we prove the following theorem (a detailed overhead analysis of our protocol and a comparison to [35] are found in Appendix D.1).

Theorem 4. *Let C be a circuit over field $\mathbb{F}_{p_0}$ with a depth d. For m unit-weight clients $\mathcal{C}$ holding inputs $\{x_1, \ldots, x_m\}$ and n servers with weights $\{w_i\}_{i\in[n]}$ and total weight $W = \sum_{i\in[n]} w_i$, the protocol above realizes a maliciously-secure weighted MPC for computing C with GOD, such that:*

[12] $\mathcal{C}$ generates shares of 1 by invoking WMPC.ShareRandDouble($W_p^{\mathcal{C}} + 2\lambda, W_p^{\mathcal{C}} + 3\lambda$), subtracting the resulting shares of a random x to get shares of 0, and then locally adding 1 to obtain shares of 1.

– *In the preprocessing phase, the communication overhead of each server per gate is $O(n^2W)$ for multiplication gates and $O(nW)$ for addition and scalar multiplication gates, and the round complexity is $O(1)$.*
– *In the online phase, the communication overhead for a client to share its inputs is $O(W)$, the communication overhead per gate for a server with weight w_i is $O(w_i)$ and the round complexity is $2d$.*
– *In the output phase, servers and clients communication overheads are $O(nW + m^2\lambda)$ and $O(m^3\lambda)$ respectively, and both have $O(1)$ round complexity.*
– *For any malicious adversary who adaptively corrupts servers with a total weight $\leq W_p$ and statically corrupts $< m/3$ clients, this protocol is computationally secure given $W - 3W_p = \Theta(\lambda)$.*

As noted, to perform the output phase securely, we require static corruption of the clients. Servers are still adaptively corrupted across the computation. If the protocol above is used in the peer-model (where the servers are the input owners), it will limit the adversary to static corruption of servers. We leave supporting adaptive security in the peer-model as a future work direction.

5 Weighted Fluid MPC

Fluid MPC [19] allows parties to come and go during protocol execution. This model divides the computation into rounds; servers register for each round they want to participate in, then a committee is elected to handle the round. Once a round concludes, the current committee transfers the state to the next committee using a handoff protocol. In particular, fluid MPC employs linear secret sharing in the BGW framework. Handoff is resharing the shares of the intermediate state, which also performs degree reduction needed for multiplication. This process continues until the last round that produces the computation output.

Fluid MPC introduces the notion of fluidity, quantifying the number of contiguous rounds a server must participate in, with maximal fluidity meaning that a server can leave after one round. This is in addition to the notions of retroactive (R) and non-retroactive (NR) adaptive malicious adversaries. Such adversaries can statically corrupt a minority of clients, and adaptively corrupt a minority of servers in each round ℓ with/without a retroactive effect. Retroactive means that the adversary obtains a server's state in all prior rounds this server participated in (so active corruption in round ℓ and passive corruption in prior rounds).

Fluid MPC, both [19] and [46], supports *security with abort*. The works [26, 28] present maximally fluid MPC with GOD at the expense of large communication complexity. All these protocols, however, are in the *unweighted setting*.

We aim to build a *Weighted Fluid MPC* protocol that allows a dynamic set of weighted servers to perform the computation while addressing malicious adversaries and offering GOD. We employ our pVWRSS scheme, and many of the techniques from Section 4, to achieve that. The fluid setting presents new challenges. First of all, we require a *verifiable* handoff protocol, to handle malicious adversaries, that must *respect the weights of the involved committees* as these committees might have different weights. Furthermore, *performing multiplication* over verifiable weighted shares requires care in this paradigm. In this section, we discuss how we address these challenges.

Additional notation and setting. We denote the committee of round ℓ as $\mathcal{P}_\ell$ consisting of n_ℓ parties such that $P_{\ell,i}$ denotes the ith party in $\mathcal{P}_\ell$. Each committee has its own pVWRSS parameters, denoted as pp_ℓ, based on the weights of its members, where $\mathsf{pp}_\ell = \{p_{\ell,i}, q_{\ell,i}, g_{\ell,i}, h_{\ell,i}\}_{i\in n_\ell}$ (for clarity, our subprotocols take these parameters as an explicit input rather than implicit as before). However, the same $\{p_0, g_0, h_0, q_0\}$ are used throughout protocol execution (the computation is over the field $\mathbb{F}_{p_0}$). Furthermore, since each committee might have different total weight, denoted as W^ℓ, the privacy and reconstruction thresholds for each committee might be different, denoted as W_p^ℓ and W_r^ℓ, respectively. As before, we have $W^\ell - 3W_p^\ell = \Theta(\lambda)$, though we set $W_r^\ell = 2W_p^\ell + 10\lambda$ (to make multiplication in the fluid setting more efficient, as we explain later). Lastly, and similar to [19], we assume that round committees are provided to the protocol using an external functionality $\mathcal{F}_{wCommittee}$. In our setting, this functionality also outputs the thresholds of the elected committee.

We note that due to space limitation, we place the figures of the handoff and multiplication protocols in Appendix **??**.

FluidHandoff: Takes as input the public parameters of $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$, namely, pp_ℓ and $\mathsf{pp}_{\ell+1}$, and the shares $\{[s]^{\alpha_\ell}_{\ell,j}\}_{j\in[n_\ell]}$ held by $\mathcal{P}_\ell$. Outputs fresh shares $\{[s]^{\alpha_{\ell+1}}_{\ell+1,j}\}_{j\in[n_{\ell+1}]}$ for $\mathcal{P}_{\ell+1}$ where $\alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda$, and $\alpha_\ell = W_p^\ell + 3\lambda$ (if the handed-off value is the output of addition/negation/scalar addition) or $\alpha_\ell = W_p^\ell + 4\lambda$ (if it is the output of scalar multiplication).

- Each $P_{\ell,i} \in \mathcal{P}_\ell$ samples $x_i, r_i \leftarrow\$\ \mathbb{F}_{p_0}$. It shares x_i, using $\mathsf{pVWRSS.Share}$, among $\mathcal{P}_\ell$ with $L = 2^{\alpha_\ell}$, $R \leftarrow\$\ r_i + U_{2^{W^\ell+\lambda}} p_0$, and parameters pp_ℓ, and among $\mathcal{P}_{\ell+1}$ with $L = 2^{\alpha_{\ell+1}-\lambda}$, $R \leftarrow\$\ r_i + U_{2^{W^{\ell+1}+\lambda}} p_0$, and parameters $\mathsf{pp}_{\ell+1}$, generating $\{[x_i]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]}$ and $\{[x_i]^{\alpha_{\ell+1}}_{\ell+1,j}\}_{j\in[n_{\ell+1}]}$, respectively.
- Parties in both $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ verify the received shares using $\mathsf{pVWRSS.Verify}$. Each party then individually generates a set V, where:

$$V = \{i \mid \{[x_i]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]} \text{ and } \{[x_i]^{\alpha_{\ell+1}}_{\ell+1,j}\}_{j\in[n_{\ell+1}]} \text{ are valid shares}$$
$$\wedge\ \left([x_i]^{\alpha_\ell+\lambda}_{\ell}.\mathsf{ct} \bmod q_0\right)\left([x_i]^{\alpha_{\ell+1}}_{\ell+1}.\mathsf{ct} \bmod q_0\right)^{-1} \equiv 1 \bmod q_0\}$$

- Each party $P_{\ell,j} \in \mathcal{P}_\ell$ computes its share $[x]^{\alpha_\ell+\lambda}_{\ell,j}$ as (also each $P_{\ell+1,j} \in \mathcal{P}_{\ell+1}$ computes $[x]^{\alpha_{\ell+1}}_{\ell+1,j}$ using same formulas with $\ell+1$ instead of ℓ):

$$[x]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh/r} = \sum_{i\in V} [x_i]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh/r} \bmod p_{\ell,j}$$

$$[x]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{ct} = \prod_{i\in V} [x_i]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{ct} \bmod Q_\ell$$

- The parties in $\mathcal{P}_\ell$ invoke:
  - $\mathsf{pVWRSS.Neg}(\mathsf{pp}_\ell, \{[x]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]})$, generating $\{[-x]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]}$.
  - $\mathsf{pVWRSS.Add}(\mathsf{pp}_\ell, \{[s]^{\alpha_\ell}_{\ell,j}\}_{j\in[n_\ell]}, \{[-x]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]})$, generating $\{[s-x]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]}$. Each party $P_{\ell,j}$ publicly reveals its share $[s-x]^{\alpha_\ell+\lambda}_{\ell,j}$.
- Each party $P_{\ell+1,j} \in \mathcal{P}_{\ell+1}$ invokes:
  - $\mathsf{pVWRSS.Reconstruct}(\mathsf{pp}_\ell, \{[s-x]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in[n_\ell]})$ to obtain $(s-x)$.
  - $\mathsf{pVWRSS.scalarAdd}(\mathsf{pp}_{\ell+1}, [x]^{W_p^{\ell+1}+3\lambda}_{\ell+1,j}, s-x)$ to obtain $[s]^{\alpha_{\ell+1}}_{\ell+1,j}$.

Fig. 7. Weighted and verifiable handoff protocol.

5.1 Weighted and Verifiable Handoff Protocol

The handoff protocol is used to hand over the intermediate state representing outputs of add, scalar add, scalar multiply, and negate gates (we discuss multiplication in the next section). This protocol also performs simultaneous degree reduction. We adapt our WMPC.DegreeReduction protocol to build a weighted and verifiable handoff FluidHandoff, as shown in Figure 7.

Recall that in WMPC.DegreeReduction, fresh shares of s (that have fresh blinding factor) are generated from the publicly revealed shares of $s-x$ with a mask x. Anyone, including the next round committee, can reconstruct $s-x$. We utilize this observation to enable handoff. That is, committee $\mathcal{P}_\ell$ generates a double sharing of x among themselves and the next committee $\mathcal{P}_{\ell+1}$. Parties in $\mathcal{P}_\ell$ then mask their shares of s using the first set of x shares and reveal the shares of $s-x$. Parties in $\mathcal{P}_{\ell+1}$ reconstruct $s-x$ and use the second set of x shares to generate fresh shares of s, which represent the handed over state.

We highlight a few points. Since $\mathcal{P}_\ell$ members do not know the second set of x shares, they will not know the fresh shares of s held by $\mathcal{P}_{\ell+1}$. Furthermore, since the shares of x held by $\mathcal{P}_{\ell+1}$ have a blinding range similar to a fresh share, this handoff achieves simultaneous degree reduction. In particular, we set the blinding range of $\mathcal{P}_{\ell+1}$'s shares of x to be $\alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda$ (meaning that $L = 2^{W_p^{\ell+1}+2\lambda}$); as we will see later, this is needed to make our multiplication protocol more efficient. Accordingly, $\mathcal{P}_\ell$ should receive shares of blinding range $\alpha_\ell = W_p^\ell + 3\lambda$ from WFMPC.FluidHandoff involving $\mathcal{P}_{\ell-1}$ and $\mathcal{P}_\ell$. This also means that the blinding range α_ℓ for a share $[x]^{\alpha_\ell}_{\ell,j}$ is $\alpha_\ell = W_p^\ell + 3\lambda$ if it is the output of WMPC.Add, WMPC.scalarAdd, or WMPC.Neg,

and $\alpha_\ell = W_p^\ell + 4\lambda$ if it is the output of WMPC.scalarMul (as again, WMPC.scalarMul increases the blinding range by a factor of 2^λ). Lastly, and as before, since the second set of x shares and fresh share computation use the parameters of $\mathcal{P}_{\ell+1}$, the fresh shares of s will have the correct weights of $\mathcal{P}_{\ell+1}$'s members.

As shown in the figure, a consistency check is needed to ensure that the same mask x has been shared correctly across $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$. We extend the one from the previous section to work across consecutive committees. Suppose $\mathcal{P}_\ell$ wants to handoff $\{[s]_{\ell,j}^{\alpha_\ell}\}_{j\in[n_\ell]}$ to $\mathcal{P}_{\ell+1}$. Each party $P_{\ell,i} \in \mathcal{P}_\ell$ randomly samples x_i, and generates two share sets; $\{[x_i]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in[n_\ell]}$ for $\mathcal{P}_\ell$, using parameters (g_ℓ, h_ℓ, Q_ℓ), and $\{[x_i]_{\ell+1,j}^{\alpha_{\ell+1}}\}_{j\in[n_{\ell+1}]}$ for $\mathcal{P}_{\ell+1}$, using parameters $(g_{\ell+1}, h_{\ell+1}, Q_{\ell+1})$. Then, each party in both committees verifies validity of the received shares, each using its respective committee parameters, and verifies that (which follows by Lemma 3):

$$
\begin{aligned}
&\left([x_i]_{\ell,j}^{\alpha_\ell+\lambda}.\mathsf{ct} \bmod q_0\right)\left([x_i]_{\ell+1,j}^{\alpha_{\ell+1}}.\mathsf{ct} \bmod q_0\right)^{-1} \\
&\equiv \left(g_\ell^{X_{i,\alpha_\ell+\lambda}} h_\ell^{R_{x_i,\alpha_\ell+\lambda}} \bmod q_0\right)\left(g_{\ell+1}^{X_{i,\alpha_{\ell+1}}} h_{\ell+1}^{R_{x_i,\alpha_{\ell+1}}} \bmod q_0\right)^{-1} \bmod q_0 \\
&\equiv \left(g_0^{X_{i,\alpha_\ell+\lambda} \bmod 2p_0} h_0^{R_{x_i,\alpha_\ell+\lambda} \bmod 2p_0}\right)\left(g_0^{X_{i,\alpha_{\ell+1}} \bmod 2p_0} h_0^{R_{x_i,\alpha_{\ell+1}} \bmod 2p_0}\right)^{-1} \bmod q_0 \\
&\equiv g_0^{x_i} h_0^{r_{x_i}} g_0^{-x_i} h_0^{-r_{x_i}} \bmod q_0 \\
&\equiv 1 \bmod q_0
\end{aligned}
$$

Note that here, $R_{x_i,\alpha_\ell+\lambda} = r_{x_i} + U_2 w_{\ell+\lambda} p_0$ and $R_{x_i,\alpha_{\ell+1}} = r_{x_i} + U_2 w_{\ell+1+\lambda} p_0$ refers to the randomnesses used by $P_{\ell,i}$ to generate the commitments $[x_i]_{\ell,j}^{\alpha_\ell+\lambda}.\mathsf{ct}$ and $[x_i]_{\ell+1,j}^{\alpha_{\ell+1}}.\mathsf{ct}$, respectively.

Once again, we will denote the first part of FluidHandoff, which generates a verifiable sharing of x across $\mathcal{P}_\ell$ (with blinding range 2^{α_ℓ}) and $\mathcal{P}_{\ell+1}$ (with blinding range $2^{\alpha_{\ell+1}}$) as ShareRandFluidDouble. Due to the dynamic participation, we cannot preprocess random sharings for later use. As a result, a committee will generate all random sharing it needs once elected and before its round begins. We note that the WFMPC.FluidHandoff protocol requires 3 rounds of communication; one round for $\mathcal{P}_\ell$ to send random masks to $\mathcal{P}_{\ell+1}$, one round for $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ to perform consistency checks (needed to broadcast complaints if any), and one round for $\mathcal{P}_\ell$ to broadcast the shares of $s - r$.

5.2 Multiplication in the Weighted Fluid Setting

Multiplication can be done as before; $\mathcal{P}_\ell$ does that using a Beaver triple, and hands off the resultant shares to $\mathcal{P}_{\ell+1}$ (where handoff also performs degree reduction). However, this will incur additional interaction; parties in $\mathcal{P}_\ell$ must interact to generate the Beaver triple and perform the multiplication, and then interact with $\mathcal{P}_{\ell+1}$ for the handoff. These steps can be performed sequentially by $\mathcal{P}_\ell$ invoking WMPC.GenBeaver, WMPC.Mul, and finally WMPC.FluidHandoff to hand the result to $\mathcal{P}_{\ell+1}$; the round complexity for $\mathcal{P}_\ell$ will be 7 rounds, or 5 rounds if certain steps are done in parallel. To avoid this excessive interaction, we modify the multiplication protocol to involve both committees and thus achieve three goals at once; multiplication, degree reduction, and handoff, bringing the round complexity down to 4 rounds. This protocol, however, slightly increases the reconstruction threshold W_r^ℓ (asymptotically it is still the same).

Our FluidMul protocol is shown in Figures 8 and 9. It starts off with generating (in parallel) $n_\ell + 2$ random sharings across $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ for the elements $\{a_k\}_{k\in[n_\ell]}$ and b, and the mask x. The parties in $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ sum up their shares of $\{a_k\}_{k\in[n_\ell]}$ to get the Beaver triple element a, while one of the random sharings across $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ define the Beaver triple element b. The share sets $\{[a]_{\ell,i}^{\alpha_\ell+\lambda}\}_{i\in[n_\ell]}$, $\{[b]_{\ell,i}^{\alpha_\ell+\lambda}\}_{i\in[n_\ell]}$, $\{[a]_{\ell+1,i}^{\alpha_{\ell+1}-\lambda}\}_{i\in[n_{\ell+1}]}$, and $\{[b]_{\ell+1,i}^{\alpha_{\ell+1}-\lambda}\}_{i\in[n_{\ell+1}]}$ play the joint role of being a part of the Beaver triple and the handoff to $\mathcal{P}_{\ell+1}$ (the reason behind choosing the blinding range of the latter two to be $2^{-\lambda}$ smaller than that of a fresh sharing in $\mathcal{P}_{\ell+1}$ will become clear shortly). This is done by computing shares of $s_1 - a$ and $s_2 - b$ as shown in Figure 9. Parties in $\mathcal{P}_\ell$ also compute verifiable shares $\{[ab]_{\ell,i}^{2\alpha_\ell+2\lambda}\}_{i\in[n_\ell]}$ in the same manner as in WMPC.GenBeaver, that they handoff to $\mathcal{P}_{\ell+1}$ using the sharing set $\{[x]_{\ell,i}^{2\alpha_\ell+3\lambda}\}_{i\in[n_\ell]}$. At this point, $\mathcal{P}_\ell$ is free to leave the protocol.

Each party in $\mathcal{P}_{\ell+1}$ reconstructs $(s_1 - a)$, $(s_2 - b)$, and $(ab - x)$ from the shares broadcast by $\mathcal{P}_\ell$. Then, $\mathcal{P}_{\ell+1}$ completes the handoff using its shares of b and x, hence obtaining shares $\{[s_2]_{\ell+1,i}^{\alpha_{\ell+1}-\lambda}\}_{i\in[n_{\ell+1}]}$

FluidMul: Takes as input the public parameters of $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$, namely, pp_ℓ and $\mathsf{pp}_{\ell+1}$, and the set of shares $\{[s_1]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}$ and $\{[s_2]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}$ to be multiplied (held by $\mathcal{P}_\ell$). Outputs $\{[s_1 s_2]^{\alpha_{\ell+1}}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$ for $\mathcal{P}_{\ell+1}$ (with α_ℓ and $\alpha_{\ell+1}$ as in Fig. 7).

1. Parties in $\mathcal{P}_\ell$ invoke the following functions in parallel:
   - For each $k \in [n_\ell]$, $\mathsf{ShareRandFluidDouble}(\mathsf{pp}_\ell, \mathsf{pp}_{\ell+1}, \alpha_\ell+\lambda, \alpha_{\ell+1}-\lambda)$ to obtain $\left\{\{[a_k]^{\alpha_\ell+\lambda}_{\ell,i}\}_{i\in[n_\ell]}\right\}_{k\in[n_\ell]}$ and $\left\{\{[a_k]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}\right\}_{k\in[n_\ell]}$.
   - $\mathsf{ShareRandFluidDouble}(\mathsf{pp}_\ell, \mathsf{pp}_{\ell+1}, \alpha_\ell+\lambda, \alpha_{\ell+1}-\lambda)$ to obtain $\{[b]^{\alpha_\ell+\lambda}_{\ell,i}\}_{i\in[n_\ell]}$ and $\{[b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$. Each party $P_{\ell,i} \in \mathcal{P}_\ell$ stores the blinded secrets $B_{\alpha_\ell+\lambda,i}$ and $B_{\alpha_{\ell+1}-\lambda,i}$ and the set V.
   - $\mathsf{ShareRandFluidDouble}(\mathsf{pp}_\ell, \mathsf{pp}_{\ell+1}, 2\alpha_\ell+3\lambda, \alpha_{\ell+1})$ to obtain $\{[x]^{2\alpha_\ell+3\lambda}_{\ell,i}\}_{i\in[n_\ell]}$ and $\{[x]^{\alpha_{\ell+1}}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$.

   So $\mathcal{P}_\ell$ gets the first set of $\{a_k\}_{k\in[n_\ell]}$, b and x shares, while $\mathcal{P}_{\ell+1}$ gets the second set of shares for $\{a_k\}_{k\in[n_\ell]}$, b and x.
2. Each $P_{\ell,j} \in \mathcal{P}_\ell$ computes its share $[a]^{\alpha_\ell+\lambda}_{\ell,j}$ as (also each $P_{\ell+1,j} \in \mathcal{P}_{\ell+1}$ computes $[a]^{\alpha_{\ell+1}+\lambda}_{\ell+1,j}$ using same formulas with $\ell+1$ instead of ℓ except for n_ℓ):

$$[a]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh/r} = \sum_{k\in[n_\ell]} [a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh/r} \bmod p_{\ell,j}$$

$$[a]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{ct} = \prod_{k\in[n_\ell]} [a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{ct} \bmod Q_\ell$$

3. Every party $P_{\ell,i} \in V$ computes and broadcasts the following commitments $\mathsf{ct}_{\ell,a_k b_i} = \left([a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{ct}\right)^{B_{\alpha_\ell+\lambda,i}} \bmod Q_\ell$, $\forall\ k \in [n_\ell]$.
4. Each party $P_{\ell,i} \in \mathcal{P}_\ell$ then checks whether:

$$\left(g_{\ell,i}^{[a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh}} h_{\ell,i}^{[a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{r}}\right)^{[b_j]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh}} \stackrel{?}{\equiv} \mathsf{ct}_{\ell,a_k b_j} \bmod q_{\ell,i}\ , \quad \forall\ k \in [n_\ell], \forall j \in V$$

   If the check fails for some $\mathsf{ct}_{\ell,a_k b_j}$, $P_{\ell,i}$ broadcasts $([a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{ct}, [b_j]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh})$ as a complaint, and all parties remove j from V.
5. Each $P_{\ell,i} \in \mathcal{P}_\ell$ and $P_{\ell+1,i} \in \mathcal{P}_{\ell+1}$ recomputes its shares $[b]^{\alpha_\ell+\lambda}_{\ell,i}$ and $[b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}$ based on the updated set V.

Fig. 8. Multiplication in the fluid weighted setting.

and $\{[ab]^{\alpha_{\ell+1}}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$. With these shares, and its shares of a, a party in $\mathcal{P}_{\ell+1}$ can locally compute the multiplication output shares $\{[s_1 s_2]^{\alpha_{\ell+1}}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$.

To enable simultaneous degree reduction, our multiplication protocol must produce shares $\{[s_1 s_2]^{\alpha_{\ell+1}}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$ with a blinding range equal to that of a fresh sharing. However, we still require that all shares for a committee to be $2^{-\lambda}$-statistically close to shares of random values, including $\{[s_2]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$, $\{[a]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$, and $\{[b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}\}_{i\in[n_{\ell+1}]}$. This requires generating these shares with $L = 2^{W_p^{\ell+1}+\lambda}$ and blinding range $2^{W_p^{\ell+1}+2\lambda}$, implying that $\alpha_{\ell+1}$ (the blinding range for a fresh sharing in $\mathcal{P}_{\ell+1}$) $= W_p^{\ell+1} + 3\lambda$). This also requires that $W_r^{\ell+1} > 2\alpha_{\ell+1} + 3\lambda = 2W_p^{\ell+1} + 9\lambda$ (to ensure that $ab - x$ can be reconstructed from $\{[ab-x]^{2\alpha_\ell+3\lambda}_{\ell,i}\}_{i\in[n_\ell]}$), or $W_r^{\ell+1} = 2W_p^{\ell+1} + 10\lambda$. This does not cause any issues as it is asymptotically the same.

5.3 Putting it All Together

To compute a circuit C with depth d, our maliciously-secure weighted fluid MPC protocol with GOD proceeds as follows:

6. Each party $P_{\ell,i} \in \mathcal{P}_\ell$ then computes its share $[ab]^{2\alpha_\ell+2\lambda}_{\ell,i}$ as:

$$[ab]^{2\alpha_\ell+2\lambda}_{\ell,i}.\mathsf{sh} = \sum_{k\in[n_\ell]} \left([a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh}\right)\left([b]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh}\right) \bmod p_{\ell,i}$$

$$[ab]^{2\alpha_\ell+2\lambda}_{\ell,i}.\mathsf{r} = \sum_{k\in[n_\ell]} \left([a_k]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{r}\right)\left([b]^{\alpha_\ell+\lambda}_{\ell,i}.\mathsf{sh}\right) \bmod p_{\ell,i}$$

$$[ab]^{2\alpha_\ell+2\lambda}_{\ell,i}.\mathsf{ct} = \prod_{k\in[n_\ell]}\prod_{j\in V} \mathsf{ct}_{\ell,a_k b_j} \bmod Q_\ell$$

7. Each $P_{\ell,i} \in \mathcal{P}_\ell$ invokes:
   - $\mathsf{WMPC.Neg}(\mathsf{pp}_\ell, [a]^{\alpha_\ell+\lambda}_{\ell,i})$, $\mathsf{WMPC.Neg}(\mathsf{pp}_\ell, [b]^{\alpha_\ell+\lambda}_{\ell,i})$, and $\mathsf{WMPC.Neg}(\mathsf{pp}_\ell, [x]^{2\alpha_\ell+3\lambda}_{\ell,i})$, generating $[-a]^{\alpha_\ell+\lambda}_{\ell,i}$, $[-b]^{\alpha_\ell+\lambda}_{\ell,i}$, and $[-x]^{2\alpha_\ell+3\lambda}_{\ell,i}$, respectively.
   - $\mathsf{WMPC.Add}(\mathsf{pp}_\ell, [s_1]^{\alpha_\ell}_{\ell,i}, [-a]^{\alpha_\ell+\lambda}_{\ell,i})$, $\mathsf{WMPC.Add}(\mathsf{pp}_\ell, [s_2]^{\alpha_\ell}_{\ell,i}, [-b]^{\alpha_\ell+\lambda}_{\ell,i})$, and $\mathsf{WMPC.Add}(\mathsf{pp}_\ell, [ab]^{2\alpha_\ell+2\lambda}_{\ell,i}, [-x]^{2\alpha_\ell+3\lambda}_{\ell,i})$, generating $[s_1 - a]^{\alpha_\ell+\lambda}_{\ell,i}$, $[s_2 - b]^{\alpha_\ell+\lambda}_{\ell,i}$, and $[ab - x]^{2\alpha_\ell+3\lambda}_{\ell,i}$, respectively, and broadcast these shares.
8. Each party in $\mathcal{P}_{\ell+1}$ invokes $\mathsf{pVWRSS.Reconstruct}$ over the broadcast shares (using parameters pp_ℓ) to obtain $(s_1 - a)$, $(s_2 - b)$, and $(ab - x)$.
9. Each party in $\mathcal{P}_{\ell+1}$ invokes:
   - $\mathsf{WMPC.scalarAdd}(\mathsf{pp}_{\ell+1}, [b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}, s_2 - b)$ to obtain $[s_2]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}$.
   - $\mathsf{WMPC.scalarAdd}(\mathsf{pp}_{\ell+1}, [x]^{\alpha_{\ell+1}}_{\ell+1,i}, ab - x)$ to obtain $[ab]^{\alpha_{\ell+1}}_{\ell+1,i}$.
   - $\mathsf{WMPC.scalarMul}(\mathsf{pp}_{\ell+1}, [s_2]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}, s_1 - a)$ to obtain $[s_1 s_2 - a s_2]^{\alpha_{\ell+1}}_{\ell+1,i}$.
   - $\mathsf{WMPC.scalarMul}(\mathsf{pp}_{\ell+1}, [a]^{\alpha_{\ell+1}-\lambda}_{\ell+1,i}, s_2 - b)$ to obtain $[a s_2 - ab]^{\alpha_{\ell+1}}_{\ell+1,i}$.
   - $\mathsf{WMPC.Add}(\mathsf{pp}_{\ell+1}, [s_1 s_2 - a s_2]^{\alpha_{\ell+1}}_{\ell+1,i}, [a s_2 - ab]^{\alpha_{\ell+1}}_{\ell+1,i}$ to obtain $[s_1 s_2 - ab]^{\alpha_{\ell+1}}_{\ell+1,i}$.
   - $\mathsf{WMPC.Add}(\mathsf{pp}_{\ell+1}, [s_1 s_2 - ab]^{\alpha_{\ell+1}}_{\ell+1,i}, [ab]^{\alpha_{\ell+1}}_{\ell+1,i})$ to obtain $[s_1 s_2]^{\alpha_{\ell+1}}_{\ell+1,i}$, which is the share of the multiplication output.

Fig. 9. Multiplication in the fluid weighted setting, continued.

Input Phase: Clients $\mathcal{C}$ share their private inputs among $\mathcal{P}_1$ using $\mathsf{pVWRSS.Share}$ with pp_1. Servers in $\mathcal{P}_1$ verify these shares by invoking $\mathsf{pVWRSS.Verify}$. For incorrect shares (for which complaints are found to be valid), servers replace the corrupt client input with some pre-determined default value.

Execution Phase: The circuit is divided into $d+1$ rounds, and so the computation is performed round-by-round, and gate-by-gate in each round, with $\mathcal{P}_\ell$ being responsible for computing the output shares of each wire in the ℓth round, and securely handing off these shares to $\mathcal{P}_{\ell+1}$ (these steps are repeated until the end of round d):

- Local operations (negate, add, scalar add, and scalar multiply gates): Based on the gate type, each server in $\mathcal{P}_\ell$ locally invokes $\mathsf{WMPC.Neg}$, $\mathsf{WMPC.Add}$, $\mathsf{WMPC.scalarAdd}$, or $\mathsf{WMPC.scalarMul}$. Then servers then invoke $\mathsf{WFMPC.FluidHandoff}$ to hand over the gate output shares to $\mathcal{P}_{\ell+1}$ with a blinding range of $\alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda$.
- Multiplication gate: Servers in both $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$ invoke $\mathsf{FluidMul}$ to perform multiplication. The multiplication output shares will have blinding range of $\alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda$.

Output Phase: $\mathcal{P}_d$ completes all $\mathsf{WFMPC.FluidHandoff}$ and $\mathsf{WFMPC.FluidMul}$ protocols with $\mathcal{P}_{d+1}$ for $\mathcal{P}_{d+1}$ to obtain the shares of all output wires. $\mathcal{P}_{d+1}$ then hands off the output shares to the clients $\mathcal{C}$ using $\mathsf{WFMPC.FluidHandoff}$. $\mathcal{C}$ then generate a Beaver triple (using $\mathsf{WMPC.GenBeaver}$). As before, $\mathcal{C}$ also generate fresh shares of 1 and multiply them by the received output shares (using the Beaver triple), and broadcast the resultant shares, allowing all clients to recover the output.

Security. In Appendix E.2, we prove the following theorem.

Theorem 5. *The protocol above realizes a weighted fluid MPC functionality secure against malicious R-adaptive adversaries with guaranteed output delivery.*

Remark 4 (Supporting maximal fluidity). Our protocol does not support maximal fluidity, a committee member *must* be online for multiple communication rounds. In return, we achieve GOD *in the weighted setting* without significant increase in round complexity (as we discuss in Appendix E.1). We leave achieving maximal fluidity in the weighted setting as a future work direction.

Acknowledgments

We thank anonymous reviewers for helpful comments and suggestions. We also thank Julia Guskind for discussions/involvement during the early stage of this work particularly around proactivity for WRSS.

References

1. The aztec protocol mpc ceremony, https://aztec.network/blog/aztec-crs-the-biggest-mpc-setup-in-history-has-successfully-finished
2. Abraham, I., Chouatt, E., Damgård, I., Gilad, Y., Stern, G., Yakoubov, S.: Nested yoso mpc: Near optimal resilience without an mpc setup. Cryptology ePrint Archive (2025)
3. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: ACM Symposium on Theory of Computing (2023)
4. Applebaum, B., Nir, O.: Upslices, downslices, and secret-sharing with complexity of 1.5^n. In: Annual International Cryptology Conference (2021)
5. Asharov, G., Lindell, Y.: A full proof of the bgw protocol for perfectly secure multiparty computation. Journal of Cryptology **30**(1), 58–151 (2017)
6. Asmuth, C., Bloom, J.: A modular approach to key safeguarding. IEEE Transactions on Information Theory (1983)
7. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference (1992)
8. Beimel, A., Weinreb, E.: Monotone circuits for monotone weighted threshold functions. Information Processing Letters (2006)
9. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: ACM Symposium on Theory of Computing (1988)
10. Ben-Sasson, E., Chiesa, A., Green, M., Tromer, E., Virza, M.: Secure sampling of public parameters for succinct zero knowledge proofs. In: IEEE Symposium on Security and Privacy (2015)
11. Benhamouda, F., Halevi, S., Stambler, L.: Weighted secret sharing from wiretap channels. In: Information-Theoretic Cryptography (2023)
12. Bienstock, A., Escudero, D., Polychroniadou, A.: On linear communication complexity for (maximally) fluid mpc. In: Annual International Cryptology Conference (2023)
13. Bienstock, A., Escudero, D., Polychroniadou, A.: Actively secure mpc with o (| c|) computation and communication via crt. In: Annual International Cryptology Conference (2026)
14. Bingol, M.A.: Multi-party setup ceremony for generating tokamak zk-snark parameters. Cryptology ePrint Archive (2024)
15. Braun, L., Damgård, I., Orlandi, C.: Secure multiparty computation from threshold encryption based on class groups. In: Annual International Cryptology Conference (2023)
16. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Chainlink Labs (2021)
17. Canetti, R., Kolby, S., Ravi, D., Soria-Vazquez, E., Yakoubov, S.: Taming adaptivity in yoso protocols: The modular way. In: Theory of Cryptography Conference (2023)
18. Chandramouli, A., Choudhury, A., Patra, A.: A survey on perfectly secure verifiable secret-sharing. ACM Computing Surveys **54**(11s), 1–36 (2022)
19. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid mpc: Secure multiparty computation with dynamic participants. In: Annual International Cryptology Conference (2021)
20. Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: ACM Symposium on Theory of Computing (1986)

21. Cohen, R., Haitner, I., Omri, E., Rotem, L.: Characterization of secure multiparty computation without broadcast. Journal of Cryptology **31**(2), 587–609 (2018)
22. Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. Journal of Cryptology **30**(4), 1157–1186 (2017)
23. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference (2007)
24. Damgård, I.B., Kamp, S.H., Loss, J., Nielsen, J.B.: Asynchronous yoso a la paillier. Cryptology ePrint Archive (2025)
25. Das, S., Xiang, Z., Tomescu, A., Spiegelman, A., Pinkas, B., Ren, L.: Verifiable secret sharing simplified. In: IEEE Symposium on Security and Privacy (2025)
26. David, B., Deligios, G., Goel, A., Ishai, Y., Konring, A., Kushilevitz, E., Liu-Zhang, C.D., Narayanan, V.: Perfect mpc over layered graphs. In: Annual International Cryptology Conference (2023)
27. David, B., Mondal, A., Satish, R.: Rumors mpc: God for dynamic committees, low communication via constant-round chat. In: International Conference on the Theory and Application of Cryptology and Information Security (2025)
28. Deligios, G., Goel, A., Liu-Zhang, C.D.: Maximally-fluid mpc with guaranteed output delivery. Cryptology ePrint Archive (2023)
29. Dolev, S., Eldefrawy, K., Lampkins, J., Ostrovsky, R., Yung, M.: Proactive secret sharing with a dishonest majority. In: International Conference on Security and Cryptography for Networks (2016)
30. Ersoy, O., Pedersen, T.B., Kaya, K., Selçuk, A.A., Anarim, E.: A crt-based verifiable secret sharing scheme secure against unbounded adversaries. Security and Communication Networks **9**(17), 4416–4427 (2016)
31. Escudero, D., Masserova, E., Polychroniadou, A.: Towards scalable yoso mpc via packed secret-sharing. In: International Conference on the Theory and Application of Cryptology and Information Security (2025)
32. Evangelou, N., Chotkan, R., Nasrulin, B., Decouchant, J.: Taming double-spending in offline payments with reputation-weighted loan networks. arXiv preprint arXiv:2504.05143 (2025)
33. Farràs, O., Guiot, M.: Reducing the share size of weighted threshold secret sharing schemes via chow parameters approximation. In: Theory of Cryptography Conference (2025)
34. Farràs, O., Guiot, M.: Revisiting beimel-weinreb weighted threshold secret sharing schemes. In: Computer Security Foundations Symposium (2026)
35. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Cryptography with weights: Mpc, encryption and signatures. In: Annual International Cryptology Conference (2023)
36. Garg, S., Jain, A., Mukherjee, P., Wang, M.: Scalable multiparty computation from non-linear secret sharing. In: Annual International Cryptology Conference (2024)
37. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The round complexity of verifiable secret sharing and secure multicast. In: ACM Symposium on Theory of Computing (2001)
38. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: Yoso: You only speak once: Secure mpc with stateless ephemeral roles. In: Annual International Cryptology Conference (2021)
39. Goel, A., Sasmal, S., Wang, M.: Weighted cryptography with weight-independent complexity. In: Annual International Cryptology Conference (2026)
40. Goldreich, O., Ron, D., Sudan, M.: Chinese remaindering with errors. In: ACM Symposium on Theory of Computing (1999)
41. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Annual International Cryptology Conference (1995)
42. Kolby, S., Ravi, D., Yakoubov, S.: Towards efficient yoso mpc without setup. IACR Cryptol. ePrint Arch. (2022)
43. Micali, S., Goldreich, O., Wigderson, A.: How to play any mental game. In: ACM Symposium on Theory of Computing (1987)
44. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks. In: ACM Symposium on Principles of Distributed Computing (1991)
45. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual International Cryptology Conference (1992)
46. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid mpc for dishonest majority. In: Annual International Cryptology Conference (2022)
47. Shehata, K., Fangqi, H., Thyagarajan, S.A.: Verifiable weighted secret sharing. In: Crypto Valley Conference (2025)
48. Tonkikh, A., Freitas, L.: Swiper: a new paradigm for efficient weighted distributed protocols. In: ACM Symposium on Principles of Distributed Computing (2024)
49. Yao, A.C.: Protocols for secure computations. In: Annual Symposium on Foundations of Computer Science (1982)
50. Yurek, T., Xiang, Z., Xia, Y., Miller, A.: Long live the honey badger: Robust asynchronous {DPSS} and its applications. In: USENIX Security Symposium (2023)

A Additional Preliminaries

A.1 Secret Sharing

A (t, n) threshold secret sharing scheme allows a dealer to distribute a secret $s \in \mathbb{F}$ among n parties. It generates shares $\{[s]_i\}_{i\in[n]}$ such that s can be recovered from any share subset of size at least t, and so any share subset of size at maximum $t-1$ does not leak any information about the secret. This represents unweighted secret sharing in the strict threshold setting with perfect privacy; each party's share counts as 1 towards the reconstruction threshold t, the privacy threshold is $t-1$, and the distributions of any share subset of size $\leq t-1$ for any two secrets are identical.

In the weighted ramp setting (that of WRSS and our scheme), parties have weights, so the share of a party i with weight w_i contributes w_i towards the threshold. Also, there is a gap (larger than 1) between the reconstruction and privacy thresholds, which we denote as W_r and W_p, respectively. Moreover, WRSS achieves ϵ-statistical privacy instead of perfect privacy.

We unify the notion for the weighted/unweighted and strict threshold/ramp settings by having the definition parametrized by two weight thresholds W_r and W_p, the total number of parties n, and a weight set for these n parties $\{w_i\}_{i\in[n]}$ (the unweighted version has $w_i = 1$ for $i \in [n]$, and the strict threshold version has $W_r = W_p + 1$). We also require ϵ-statistical privacy (to capture perfect privacy, the privacy property below would require the two distributions to be identical instead of ϵ-statistically close).

Definition 1. *A $(W_r, W_p, n, \{w_i\}_{i\in[n]})$-secret sharing scheme over field $\mathbb{F}$ with reconstruction threshold W_r, privacy threshold W_p, and a set of n parties with weights $\{w_i\}_{i\in[n]}$, where $W_r, W_p, n, w_i \in \mathbb{N}$, consists of a pair of algorithms* $(\mathsf{Share}, \mathsf{Reconstruct})$ *such that:*

- Share *is a randomized algorithm that on input secret $s \in \mathbb{F}$ outputs an n-tuple of shares $\{[s]_i\}_{i\in[n]}$ such that each $[s]_i$ has a weight of w_i.*
- $\mathsf{Reconstruct}$ *is a deterministic algorithm that on input a subset of shares whose total weight is at least W_r outputs a secret $s \in \mathbb{F}$.*

and satisfying the following properties:

- ***Correctness.*** *For all share subsets of total weight $\geq W_r$,* $\mathsf{Reconstruct}$ *outputs the original secret s. Formally, $\forall s \in \mathbb{F}$, and $\forall S = \{i_1, \ldots, i_k\} \subseteq [n]$ such that $\sum_{j\in[k]} w_{i_j} \geq W_r$, we have*

$$\Pr_{\mathsf{Share}(s)\to([s]_1,\ldots,[s]_n)} \left[\mathsf{Reconstruct}([s]_{i_1}, \ldots, [s]_{i_k}) = s\right] = 1$$

- ***ϵ-Statistical Privacy.*** *For any unauthorized set of shares (i.e., its total weight is $\leq W_p$), the distributions of the shares for any two distinct secrets are ϵ-statistically close. Formally, $\forall s, s' \in \mathbb{F}, \forall S = \{i_1, \ldots, i_k\} \subseteq [n]$ such that $\sum_{j\in[k]} w_{i_j} \leq W_p$, the following distributions are ϵ-statistically close:*

$$\{([s]_i \in S) : ([s]_1, \ldots, [s]_n) \leftarrow \mathsf{Share}(s)\}$$

$$\{([s']_i \in S) : ([s']_1, \ldots, [s']_n) \leftarrow \mathsf{Share}(s')\}$$

A.2 Proactive Secret Sharing

Proactive secret sharing extends the standard notion of secret sharing with a mechanism to refresh the shares such that the new shares are for the same original secret. However, these new shares cannot be combined with the old ones to obtain any information about the secret. We adapt the refresh definition from [29] to extend Definition 1 with a reshare algorithm for share refreshing.

Definition 2. *A $(W_r, W_p, n, \{w_i\}_{i\in[n]})$-proactive secret sharing scheme over field $\mathbb{F}$ with reconstruction threshold W_r, privacy threshold W_p, and a set of n parties with weights $\{w_i\}_{i\in[n]}$, where $W_r, W_p, n, w_i \in \mathbb{N}$, consists of three algorithms* $(\mathsf{Share}, \mathsf{Reshare}, \mathsf{Reconstruct})$*, where* $(\mathsf{Share}, \mathsf{Reconstruct})$ *are as in Definition 1, and* $\mathsf{Reshare}$ *is defined as follows:*

– Reshare *is a randomized algorithm that on input n-tuple of shares* $\{[s]_i\}_{i\in[n]}$ *for some secret* $s \in \mathbb{F}$*, outputs a fresh n-tuple of shares* $\{[s]'_i\}_{i\in[n]}$ *for the same s and same weight structure, i.e., each* $[s']_i$ *has a weight* w_i*.*

and satisfying the following properties:

– ***Correctness.*** *As in Definition 1 such that the share subset is taken from the output of the same share/reshare instance, i.e., not mixed across different share/reshare instances.*
– ϵ***-Statistical Privacy.*** *As in Definition 1 such that the total weight of any share subset taken from the output of any share/reshare instance is* $\leq W_p$*.*

Correctness guarantees recovery of the original secret if Reconstruct is executed over a share subset of total weight $\geq W_r$ given that this subset is taken from the output of a particular Share or Reshare invocation. Privacy is guaranteed if any subset of shares, drawn from the output of any Share or Reshare invocation, has a total weight $\leq W_p$. In other words, combining shares drawn from the output of different Share/Reshare invocations is not useful, i.e., cannot help in reconstruction or leaking any information about the secret.

A.3 Verifiable Secret Sharing

Verifiable secret sharing (VSS) extends the standard notion of secret sharing to achieve robustness against active adversaries who may not only control share holders (of total weight up to W_p) but also the dealer. Informally, if the dealer is honest, then the collusion of share holders of total weight $\leq W_p$ cannot prevent the honest share holders from reconstructing the correct secret. While if the dealer is dishonest, then the collusion of this dealer and malicious share holders (again with total weight $\leq W_p$) cannot affect the reconstructed value once the sharing phase is over. In other words, any execution of the sharing phase determines a unique value that will be recovered during the reconstruction phase.

We adapt the VSS definitions from [18, 25, 37] to extend Definition 1 to capture verifiability. That is, correctness and privacy now hold when the dealer is honest, and we have an additional property, called completeness, that handles the dishonest dealer case (other works refer to this property as commitment [18, 37], we use completeness as in [25]).

Definition 3. *A* $(W_r, W_p, n, \{w_i\}_{i\in[n]})$*-verifiable secret sharing scheme over field* $\mathbb{F}$ *with reconstruction threshold* W_r*, privacy threshold* W_p*, and a set of n parties with weights* $\{w_i\}_{i\in[n]}$*, where* $W_r, W_p, n, w_i \in \mathbb{N}$*, consists of a pair of algorithms* (Share, Reconstruct) *as in Definition 1, and satisfies the following properties (for any adversary that controls a party subset with total weight* $\leq W_p$*):*

– ***Correctness.*** *If the dealer is honest, then all honest parties will output the original secret s at the end of the reconstruction phase.*
– ϵ***-Statistical Privacy.*** *If the dealer is honest, then the privacy notion from Definition 1 applies.*
– ***Completeness.*** *If the dealer is dishonest, then there exists a value* $\tilde{s} \in \mathbb{F} \cup \perp$ *such that all honest parties hold shares of* $\tilde{s}$*. Also, all honest parties will output* $\tilde{s}$ *at the end of the reconstruction phase.*

Remark 5 (Verifiable proactivity notion). The notion of verifiable proactive secret sharing is basically extending the notion in Definition 3 with the reshare algorithm from Definition 2. In terms of properties, for an honest dealer, correctness and privacy are the same as in Definition 3, and for completeness, once the share phase is completed (and so there exist a value $\tilde{s} \in \mathbb{F} \cup \perp$ such that all honest parties hold shares of $\tilde{s}$), all future reshare instances will produce fresh shares of the same $\tilde{s}$.

A.4 Other Cryptographic Building Blocks

Commitments. A commitment scheme consists of three algorithms: Setup, Commit, and Open. Setup takes as input a security parameter κ and generates a set of public parameters pp. Commit takes pp, a message m, and randomness r as inputs, and outputs a commitment ct to m. Open takes pp and ct as inputs and

produces a decommitment $d = (m, r)$. A secure commitment scheme must satisfy two properties: *hiding*, meaning that ct does not reveal any information about m, and *binding*, meaning that ct cannot be opened to m' such that $m' \neq m$.

In our verifiable WRSS scheme, the individual share commitments are Pedersen commitments [45]. In Pedersen commitments, the setup phase involves choosing a large prime p and a safe prime $q = 2p + 1$, and $h \leftarrow\$ G_p$ (where G_p is the unique subgroup of $\mathbb{Z}_q^*$ of order p with a generator g), such that nobody knows the discrete log of h with respect to g. To commit to a value $m \in \mathbb{Z}_p$, sample $r \leftarrow\$ \mathbb{Z}_p$, then compute $\mathsf{ct} = \mathsf{Commit}(m; r) = g^m h^r \bmod q$. Opening is done by revealing both m and r. Pedersen commitments are perfectly hiding and computationally binding.

Non-interactive zero-knowledge (NIZK) proofs. Let $\mathcal{R}$ be an efficiently computable binary relation which consists of pairs (x, ω), and let $\mathcal{L}_\mathcal{R}$ be the language associated with $\mathcal{R}$ (so ω is a witness that the statement x belongs to $\mathcal{L}_\mathcal{R}$). A zero-knowledge proof (ZKP) system consists of three algorithms: Setup, Prove, and Verify. Setup takes as inputs a security parameter κ and the specifications of $\mathcal{L}_\mathcal{R}$, and outputs public parameters pp. Prove takes as inputs pp, a statement x, and a witness ω for x and outputs a proof π attesting that $x \in \mathcal{L}_\mathcal{R}$. Verify takes pp, statement x, and π as inputs, and outputs 1 if π is valid and 0 otherwise. Being non-interactive means that the prover can generate π on its own without interacting with the verifier and send π to the prover in one message, and that the verifier can verify π without interacting with the prover.

We require a secure NIZK proof system that satisfies the following properties: *Completeness* stating that any proof that is generated in an honest way will be accepted by the verifier; *zero knowledge* stating that a proof π for a statement x does not reveal anything about the witness ω; and *simulation extractability* stating that a prover that produces a valid proof must know a valid witness even after seeing many simulated proofs. In other words, there exists an extractor algorithm that can extract a valid witness from a valid proof despite giving the prover an access to the proof simulation oracle. This property implies non-malleability, i.e., a malicious prover cannot forge a new valid proof by modifying an accepted proof into a new one without knowing the witness.

Range proofs. A range proof allows a prover to convince a verifier that some hidden input is within a particular range without revealing the exact input value. We use range proofs over committed values; given a commitment scheme with public parameters pp, the proof is for the relation:

$$\mathcal{R} = \{((\mathsf{ct}, a, b), (m, r)) : \mathsf{ct} = \mathsf{Commit}(\mathsf{pp}, m, r) \wedge a \leq m \leq b\}$$

Knowledge of the discrete log proofs. A proof for the knowledge of the discrete log allows a prover to convince a verifier that it knows the discrete log x (with respect to g) of $y = g^x \bmod p$ without revealing anything about x, where g is a generator of cyclic group G of order p for which the discrete log assumption holds. That is, the proof is for the relation:

$$\mathcal{R} = \{(y, x) : y = g^x \bmod p\}$$

We use both proof systems in a black box manner, where any secure (one that achieves the completeness, zero knowledge, and soundness extractability properties from above) non-interactive range/knowledge of the discrete log proof system can be employed in our constructions.

B Correctness and Security Proofs for Proactive and Verifiable WRSS

B.1 Correctness and Security of Proactive WRSS

Theorem 6 (Theorem 1 restated). *For $W_r \geq W_p + 4\lambda + \Theta(1)$ and $W - W_p = \Theta(\lambda)$, the proactive WRSS scheme from Section 3.1 is perfectly correct and statistically private (cf. Definition 2).*

Proof. We show how our proactive WRSS scheme achieves the correctness and privacy properties. We focus on the $\mathsf{Reshare}$ algorithm as this is the additional component in proactive WRSS on top of the (secure) WRSS scheme.

Correctness. To prove correctness, we have to prove that the new shares have fresh randomness, are correct shares for the same secret s, and respect the parties' weights. In Reshare, the parties basically run a distributed randomness generation protocol, so that each party P_i obtains two shares $[r]_i^0$ and $[r]_i^1$. These shares are distinct and valid shares of the *same* $r = \sum_{i\in[n]} r_i$, which follows from the additive homomorphism and correctness of WRSS (recall in proactive WRSS we assume semi-honest parties). As discussed under the privacy of proactive WRSS (shown next), the shares of $[r]_i^0$ and $[r]_i^1$ generated by Reshare are statistically close to shares of r generated by WRSS.Share with $L = 2^{W_p+3\lambda}$ and $L = 2^{W_p+2\lambda}$ respectively; for the sake of proving correctness, we can assume the latter as the former still carry the same underlying secret and satisfy reconstruction correctness. After that, each party P_i negates its share $[r]_i^0$. We show that the negation computation is correct (where $L = 2^{W_p+2\lambda}$ and U'_L is a different random integer over $[L]$):

$$\begin{aligned}[-r]_i^0 &= (L+1)p_0 - [r]_i^0 \bmod p_0 \\ &= (L+1)p_0 - r - U_L p_0 \bmod p_i \\ &= \underbrace{(p_0 - r) + U'_L p_0 \bmod p_i}_{i^{th} \text{ share of } -r \text{ with blinding factor within } 2^{W_p+3\lambda}}\end{aligned}$$

Note that the value used in computing the negation, i.e., $L = 2^{W_p+2\lambda}$, is in fact $L = n2^{W_p+2\lambda}$ as discussed before; all parties use this exact value when computing the shares of $s - r$. By the additive homomorphism property of WRSS, $[s]_i + [-r]_i^0$ indeed produces a valid share $[s-r]_i$. By the correctness of the WRSS reconstruction algorithm, each P_i can recover the masked secret $s-r$ from the broadcast shares. Then, each P_i computes a fresh share $[s]'_i$ (where $L = 2^{W_p+\lambda}$):

$$\begin{aligned}[r]_i^1 + (s-r) \bmod p_i &= r + U_L p_0 + s - r \bmod p_i \\ &= s + U_L p_0 \bmod p_i\end{aligned}$$

As shown, $[s]'_i$ is for the same secret s but with a fresh blinding range within $2^{W_p+2\lambda}$ as desired. Also, as the mask shares P_i obtains have weight w_i and the computation is done with respect to p_i, this will produce a refreshed share with weight w_i. As such, having an authorized set of these fresh shares, under the same reconstruction threshold W_r, allows reconstructing the secret s. Finally, and recall that proactive WRSS is semi-honestly secure, we have $W \geq W_r$ (which implies $W \geq W_p + 4\lambda + \Theta(1)$ or $W - W_p = \Theta(\lambda)$, as desired), allowing the parties to perform reconstruct successfully.

Privacy. To prove privacy, we have to show that the reshare protocol steps, any unauthorized subset of refreshed shares, or any combination of unauthorized subsets of old and refreshed shares, do not leak any information about s.

Note that since each party P_i contributes its own random r_i, the mask r is random and unknown to all parties. We show that the masked secret shares $\{[s-r]_i\}_{i\in[n]}$, that are made public, do not reveal anything about s. We do this in two stages. In the first stage, we assume that the shares of the mask r generated by Reshare ($\{[r]_i^0, [r]_i^1\}_{i\in[n]}$) are shares of r generated by WRSS.Share with $L = 2^{W_p+2\lambda}$ and $L = 2^{W_p+\lambda}$, respectively (which we denote as $\{[\overline{r}]_i^0, [\overline{r}]_i^1\}_{i\in[n]}$) and show that the shares $\{[s-\overline{r}]_i\}_{i\in[n]}$ do not leak any information about s. Then, in the second stage, we show that the shares $\{[r]_i^0, [r]_i^1\}_{i\in[n]}$ and $\{[\overline{r}]_i^0, [\overline{r}]_i^1\}_{i\in[n]}$ are statistically close. Combining these two stages shows that the masked secret shares $\{[s-r]_i\}_{i\in[n]}$ do not leak any information about s, as required.

For the first stage of the proof, we use the Integer Masking Lemma from [35]. We note that our proof uses similar arguments as [35]; we state the full proof for completeness.

Lemma 1 (Integer Masking Lemma [35]). *Let p, $0 \leq r_1, r_2 < p$, and $M < N$ be any integers, and D be an arbitrary distribution over the universe $\{r_1, r_1+p, r_1+2p, \dots\} \cap |M|$. Then, we have (where SD is the statistical distance and the addition is over the integers):*

$$SD\left((D + U_N \mid U_N \bmod p = r_2),\ (U_N \mid U_N \bmod p = r_1 + r_2)\right) \leq \frac{M}{N} + \frac{2p}{N}$$

Using this lemma, we show that the statistical distance between the revealed shares and a random sharing that is independent of s, both with the same blinding range, is $2^{-\lambda}$ close. First, note that the distribution of $\{[s-\overline{r}]_i\}_{i\in[n]}$ can be viewed from the perspective of the blinded secret associated with the shares. This is because the shares are computed from the blinded secret via CRT and vice versa. Thus, we can shift our focus to the blinded secret associated with these shares, say X with $L = 2^{W_p+2\lambda}$, where:

$$X = \underbrace{s + U_{2^{W_p+\lambda}p_0}}_{D} + \underbrace{\left(p_0 - r + U'_{2^{W_p+2\lambda}p_0}\right)}_{U_N \mid U_N \bmod p_0 = p_0 - r}$$

Lemma 1 implies that the distribution of X above is $2^{-\lambda}$-statistically close to a fresh sharing of $s + (p_0 - r) \equiv s - r \bmod p_0$. In particular, we have:

$$\begin{aligned}
&SD\left((D + U_N \mid U_N \bmod p_0 = (p_0 - r)), (U_N \mid U_N \mod p_0 = s + (p_0 - r))\right) \\
&\quad\leq \frac{2^{W_p+\lambda}}{2^{W_p+2\lambda}} + \frac{2p_0}{2^{W_p+2\lambda}} \\
&\quad\leq 2^{-\lambda} + 2^{-(W_p+\lambda-1)} \\
&\quad\leq poly(\lambda)2^{-\lambda}
\end{aligned}$$

Since $r \in \mathbb{F}_{p_0}$ is random, this means that $s - r$ is also a random element from $\mathbb{F}_{p_0}$. Thus, the shares $\{[s-\overline{r}]_j\}_{j\in[n]}$ are $2^{-\lambda}$-statistically close to a sharing of some random element independent of s, meaning that these shares do not leak any information about s.

For the second stage, we show that the shares $\{[r]_i^0, [r]_i^1\}_{i\in[n]}$ (i.e., shares of r generated by $\mathsf{Reshare}$) and the shares $\{[\overline{r}]_i^0, [\overline{r}]_i^1\}_{i\in[n]}$ (i.e., shares of r generated by $\mathsf{WRSS.Share}$) are statistically close. Towards this, we use the following lemma.

Lemma 2. *Let $X = U_x$, $Y = U_y \bmod x$, and $z = y \bmod x$, then $SD(X,Y) = \frac{z(x-z)}{xy}$.*

Proof. Let $y = \alpha x + z$. For any $a < z$, $\Pr[Y = a] = \frac{\alpha+1}{y}$. For any $a \geq z$, $\Pr[Y = a] = \frac{\alpha}{y}$. Applying the definition of statistical distance gives us:

$$\begin{aligned}
SD(X,Y) &= \frac{1}{2}\sum_a \left|\Pr[X = a] - \Pr[Y = a]\right| \\
&= \frac{1}{2}\left(\sum_{a=0}^{z-1}\left|\frac{1}{x} - \frac{\alpha+1}{y}\right| + \sum_{a=z}^{x-1}\left|\frac{1}{x} - \frac{\alpha}{y}\right|\right) \\
&= \frac{1}{2}\left(z\left(\frac{x-z}{xy}\right) + (x-z)\left(\frac{z}{xy}\right)\right) \\
&= \frac{z(x-z)}{xy}
\end{aligned}$$

as required. □

Consider the shares $\{[r]_i^0\}_{i\in[n]}$, and let $\bar{B} \subset [n]$ be the set of corrupt parties. The adversary's view of these shares is $\{[r]_i^0\}_{i\in\bar{B}}$, which again we can equivalently view as $R^0 \bmod P_{\bar{B}}$ where $P_{\bar{B}} = \prod_{i\in\bar{B}} p_i$ and R^0 is the blinded secret associated with the sharing. We can represent R^0 as (where $U_{L,i}$ is the value used in the blinded mask R_i^0):

$$R^0 = \sum_{i\in[n]} R_i^0 = \sum_{i\in[n]} r_i^0 + U_{L,i}p_0 = r + \left(\sum_{i\in[n]} U_{L,i}\right)p_0 = r + \sigma_n p_0$$

Similarly, the adversary's view of the shares $\{[\overline{r}]_i^0\}_{i\in[n]}$ can be equivalently viewed as $\overline{R^0} = r + U_L p_0 \bmod P_{\bar{B}}$ for $L = 2^{W_p+2\lambda}$. Clearly:

$$\begin{aligned} SD\left(R^0 \bmod P_{\bar{B}}, \overline{R^0} \bmod P_{\bar{B}}\right) &= SD\left((r+\sigma_n p_0) \bmod P_{\bar{B}}, (r+U_L p_0) \bmod P_{\bar{B}}\right) \\ &= SD\left(\sigma_n \bmod P_{\bar{B}}, U_L \bmod P_{\bar{B}}\right) \end{aligned}$$

Let $\sigma_n = U_{L,1} + \sigma_n'$ where $U_{L,1}$ is uniformly chosen over $[0, L]$ and unknown to the adversary (WLOG, let P_1 be an honest party) and $\sigma_n' = \sum_{i=2}^{n} U_{L,i}$. If we define $f(x) = x + \sigma_n' \bmod P_{\bar{B}}$, then by the data processing inequality:

$$\begin{aligned} SD(U_L \bmod P_{\bar{B}}, U_{P_{\bar{B}}}) &\geq SD\left(f(U_L \bmod P_{\bar{B}}), f(U_{P_{\bar{B}}})\right) \\ &\geq SD\left(\left((U_L \bmod P_{\bar{B}}) + \sigma_n' \bmod P_{\bar{B}}\right), \left(U_{P_{\bar{B}}} + \sigma_n' \bmod P_{\bar{B}}\right)\right) \\ &\geq SD\left(\left(U_L + \sigma_n' \bmod P_{\bar{B}}\right), U_{P_{\bar{B}}} \bmod P_{\bar{B}}\right) \\ &\geq SD\left(\sigma_n \bmod P_{\bar{B}}, U_{P_{\bar{B}}} \bmod P_{\bar{B}}\right) \end{aligned}$$

Further, Lemma 2 gives us that:

$$\begin{aligned} SD(U_L \bmod P_{\bar{B}}, U_{P_{\bar{B}}}) &= \frac{(L \bmod P_{\bar{B}})(P_{\bar{B}} - (L \bmod P_{\bar{B}}))}{L P_{\bar{B}}} \\ &\leq \frac{(P_{\bar{B}})^2}{4 L P_{\bar{B}}} \\ &\leq \frac{P_{\bar{B}}}{4L} \end{aligned}$$

Applying the the triangle inequality, we get:

$$\begin{aligned} SD\left(\sigma_n \bmod P_{\bar{B}}, U_L \bmod P_{\bar{B}}\right) &\leq SD\left(\sigma_n \bmod P_{\bar{B}}, U_{P_{\bar{B}}}\right) + SD\left(U_{P_{\bar{B}}}, U_L \bmod P_{\bar{B}}\right) \\ &\leq SD\left(U_L \bmod P_{\bar{B}}, U_{P_{\bar{B}}}\right) + SD\left(U_{P_{\bar{B}}}, U_L \bmod P_{\bar{B}}\right) \\ &\leq \frac{2P_{\bar{B}}}{4L} \\ &\leq \frac{2^{W_p}}{2^{W_p+2\lambda+1}} \\ &\leq poly(\lambda) 2^{-2\lambda} \end{aligned}$$

Thus, the shares $\{[r]_i^0\}_{i\in[n]}$ and $\{[\overline{r}]_i^0\}_{i\in[n]}$, from the perspective of the adversary are $2^{-2\lambda}$-statistically close. A similar argument applies for the shares $\{[r]_i^1\}_{i\in[n]}$, where because $L = 2^{W_p+\lambda}$, we get that that the shares $\{[r]_i^1\}_{i\in[n]}$ and $\{[\overline{r}]_i^1\}_{i\in[n]}$ from the perspective of the adversary are $2^{-\lambda}$-statistically close.

Finally, by the triangle inequality, get that the statistical distance between the shares $\{[s-r]_i\}_{i\in[n]}$ and shares of some random element independent of s is $\leq$ the sum of the statistical distance between the shares $\{[s-\overline{r}]_i\}_{i\in[n]}$ and shares of some random element independent of s (which we showed to be $poly(\lambda)2^{-\lambda}$) and the statistical distance between the shares $\{[r]_i^0\}_{i\in[n]}$ and $\{[\overline{r}]_i^0\}_{i\in[n]}$ from the perspective of the adversary (which we showed to be $poly(\lambda)2^{-2\lambda}$), resulting in the shares $\{[s-r]_i\}_{i\in[n]}$ being $2^{-\lambda}$-statistically close to shares of some random element independent of s, as required.

Now we show that the fresh shares also preserve privacy and that they form a different system of congruences than the one associated with the older shares, meaning that fresh and old shares cannot be combined.

Note that $\{[r]_i^1\}_{i\in[n]}$ is $2^{-\lambda}$-statistically close to $\{[\overline{r}]_i^1\}_{i\in[n]}$ from the perspective of the adversary. Hence, for each $i \in B$, P_i's fresh share $[s]_i'$ is $2^{-\lambda}$ statistically close to $s + U_L' p_0 \bmod p_i$, where U_L' is a uniformly random value unknown to all parties, and crucially, different from the U_L used in the old shares of s (i.e $S = s + U_L p_0$). Thus, any unauthorized subset of the fresh shares will not reveal any information about s,

which follows by the privacy of WRSS as these fresh shares have the same access structure as the original ones. Moreover, the old shares cannot be combined with the new shares as they correspond to a different system of congruences.

As a result, our reshare protocol preserves the privacy of s. □

B.2 Correctness and Security of VWRSS

We first prove the security of the aggregated commitment. We also prove two lemmas that establish correctness of individual share commitment extraction. These are needed for the security proof of the aggregated commitment as well as the correctness proof of the VWRSS scheme.

Lemma 3. *If $x \bmod p = y$ for some prime p, then $x \bmod 2p = y$ or $x \bmod 2p = p + y$.*

Proof. $x \bmod p = y$ implies that $x = kp + y$, where $y < p$ and k is some integer. Thus, $x \bmod 2p = kp + y \bmod 2p$. There are two cases to consider: when k is even and when k is odd. Let us look at the first case: if k is even, we can represent it as $k = 2l$ where l is another integer. Thus, $x \bmod 2p = 2lp + y \bmod 2p = y \bmod 2p = y$. In the remaining case where k is odd, we can represent it as $k = 2l + 1$, where l is some other integer. Thus, $x \bmod 2p = 2lp + p + y \bmod 2p = p + y \bmod 2p = p + y$. Thus, we end up with $x \bmod 2p = y$ in the first case, and $x \bmod 2p = p + y$ in the second case, as required. □

Lemma 4. *Given the parameter setting in Figure 2, for all $i \in [n]$, $\mathsf{ct}_i = g_i^{[s]_i} h_i^{r_i} \bmod q_i$ can be extracted from $\mathsf{ct} = g^S h^R \bmod Q$ by computing $\mathsf{ct}_i = \mathsf{ct} \bmod q_i$.*

Proof. Since q_i divides Q, $g_i = g \bmod q_i$, and $h_i = h \bmod q_i$, we have:

$$\begin{aligned}
\mathsf{ct} \bmod q_i &= \left(g^S h^R \bmod Q\right) \bmod q_i \\
&= (g \bmod q_i)^S (h \bmod q_i)^R \bmod q_i \\
&= (g_i^S)(h_i^R) \bmod q_i \\
&= (g_i^{S \bmod \phi(q_i)})(h_i^{R \bmod \phi(q_i)}) \bmod q_i \\
&= (g_i^{S \bmod 2p_i})(h_i^{R \bmod 2p_i}) \bmod q_i
\end{aligned}$$

Recall that $S \bmod p_i = [s]_i$ and $R \bmod p_i = r_i$, by construction. Then, by Lemma 3, we have two cases for S: $S \bmod 2p_i = [s]_i$ or $S \bmod 2p_i = p_i + [s]_i$. Since g_i is a generator for the group G_{p_i} (which is of order p_i), we have $g_i^{p_i} \equiv 1 \bmod q_i$. Thus, the second case collapses to the first case:

$$g_i^{S \bmod 2p_i} \equiv g_i^{p_i + [s]_i} \equiv g_i^{p_i} g_i^{[s]_i} \equiv 1 \cdot g_i^{[s]_i} \equiv g_i^{[s]_i} \bmod q_i$$

Similarly, we have two cases for R: $R \bmod 2p_i = r_i$ or $R \bmod 2p_i = p_i + r_i$. Since $h_i \in G_{p_i}$, we know that $h_i = g_i^x$ for some integer x, hence the second case collapses to the first:

$$h_i^{R \bmod 2p_i} \equiv h_i^{p_i + r_i} \equiv h_i^{p_i} h_i^{r_i} \equiv (g_i^{p_i})^x h_i^{r_i} \equiv h_i^{r_i} \bmod q_i$$

Thus, we can consider $S \bmod 2p_i = [s]_i$ and $R \bmod 2p_i = r_i$. Substituting this in $\mathsf{ct} \bmod q_i$, we get:

$$(g_i^{S \bmod 2p_i})(h_i^{R \bmod 2p_i}) \bmod q_i = g_i^{[s]_i} h_i^{r_i} \bmod q_i = \mathsf{ct}_i$$

as required. □

Lemma 5. *Assuming the security of Pedersen commitments, the aggregated commitment to the blinded secret S (given the parameter setting in Figure 2) is perfectly hiding and computationally binding.*

Proof. The proof proceeds in two stages. We first prove that the aggregated commitment is perfectly hiding and computationally binding under a trusted setup, i.e., no one knows the discrete log of each h_i. Then, we prove that the distributed setup in VWRSS generates parameters with the same guarantees as a trusted setup. In other words, perfect hiding and computational binding are preserved under this distributed parameter setup.

For the first stage, recall that that for our aggregated commitment, $\mathsf{ct} = g^S h^R \bmod Q$, we have $\mathsf{ct} \equiv \mathsf{ct}_i \bmod q_i$ and $\mathsf{ct}_i = g_i^{[s]_i} h_i^{r_i} \bmod q_i$ is a Pedersen commitment to $[s]_i$ using randomness r_i (where $r_i = R \mod p_i$ is uniformly distributed in $\mathbb{Z}_{p_i}$ since $R \leftarrow\$ \mathbb{Z}_P$, where $P = \prod_{j\in[n]} p_j$) for all $i \in [n]$. Under a trusted setup, we have parameters $\{g_i, h_i\}_{i\in[n]}$ are guaranteed to be such that $g_i, h_i \in G_{p_i}$, where $G_{p_i} = \mathbb{Z}_{q_i}^*$ and nobody knows $\log_{g_i} h_i$.

For perfect hiding, we know by CRT that the mapping between ct and $\{\mathsf{ct}_i\}_{i\in[n]}$ is bijective. By Lemma 4, we can extract $\{\mathsf{ct}_i\}_{i\in[n]}$ from ct (via $\mathsf{ct} \bmod q_i$ for $i \in [n]$) and each Pedersen commitment ct_i itself is perfectly hiding, it follows that ct does not reveal any information about S as well. Thus our aggregated commitment ct is perfectly hiding, as required.

For computational binding, this also follows by a reduction to the computational binding of Pedersen commitments as follows. Suppose there exists an adversary $\mathcal{A}$ that breaks the computational binding of the aggregated commitment ct with non-negligible probability (i.e., $\mathcal{A}$ can open $\mathsf{ct} = g^S h^R \bmod Q$ to a different S', or that $\mathcal{A}(g, h, S, R) = (S', R')$ such that $\mathsf{ct} = g^S h^R = g^{S'} h^{R'}$ and $S \neq S'$). Then, we can use $\mathcal{A}$ to construct another adversary $\mathcal{A}'$ that breaks the computational binding of a Pedersen commitment to the individual shares, WLOG, say for $\mathsf{ct}_1 = g_1^{[s]_1} h_1^{r_1} \bmod q_i$, as follows:

- Honestly generate $\mathsf{ct}_j = g_j^{[s]_j} h_j^{r_j} \bmod q_j$ using parameters (g_j, h_j) for $j \in [n] - \{1\}$. This can be done by randomly sampling $S \leftarrow\$ \{0, \ldots, \prod_{i\in[n]} p_i\}$ such that $S \equiv [s]_1 \bmod p_1$, computing $[s]_j \equiv S \bmod p_j$ for all js, and randomly sampling r_j for all js. This ultimately results in generating $n-1$ new Pedersen commitments.
- Compute $\mathsf{ct} \equiv \mathsf{ct}_i \bmod q_i$, $R \equiv r_i \bmod p_i$, $g \equiv g_i \bmod q_i$, and $h \equiv h_i \bmod q_i$ for $i \in [n]$ via CRT.
- Invoke $\mathcal{A}(g, h, S, R)$ which outputs (S', R').
- Output $([s]_1', r_1')$, where $[s]_1' = S' \bmod p_1$ and $r_1' = R' \bmod p_1$.

All of these commitments are honestly generated by $\mathcal{A}'$ and sent to $\mathcal{A}$, and so $\mathcal{A}$ does not know the $log_{g_i}(h_i)$ for any i. $\mathcal{A}$ will output (with non-negligible probability) S' and R' such that $g^S h^R = g^{S'} h^{R'} = \mathsf{ct}$. By the bijective mapping of CRT, this results in the binding of at least one of the Pedersen commitments ct_i that are aggregated to form ct to be broken. However, from the perspective of $\mathcal{A}$, all of the individual commitments ct_i are equivalent ($\mathcal{A}$ does not know that $\mathcal{A}'$ received $g_1^{[s]_1} h_1^{r_1}$ as input). Hence, the probability that $([s]_1', r_1')$ is such that $g_1^{[s]_1'} h_1^{r_1'} = g_1^{[s]_1} h_1^{r_1}$ is $1/n$, which is non-negligible. The probability that $\mathcal{A}$ breaks the computational binding property for the Pedersen commitment $g_1^{[s]_1} h_1^{r_1}$ is hence overall non-negligible, which is a contradiction to the computational binding property of Pedersen commitments. Thus, our aggregate commitment ct must also be computationally binding, as required.

For the second stage, we have to show that our distributed setup produces parameters $\{g_i, h_i\}_{i\in[n]}$ that achieve the same properties as the trusted setup, namely that $g_i, h_i \in G_{p_i} = \mathbb{Z}_{q_i}^*$ and $\log_{g_i}(h_i)$ is unknown for all $i \in [n]$.

It is clear that the distributed setup generates parameters $g_i, h_i \in G_{p_i} = \mathbb{Z}_{q_i}^*$. Suppose that after the distributed setup, $\mathcal{A}$ knows $\log_{g_i}(h_i)$ for some $i \in [n]$. Recall that $h_i = \prod_{j\in[n]} g_i^{x_{ij}}$, which for the purposes of analysis is expressed as:

$$h_i = \left(\prod_{P_j \in honest} g_i^{x_{ij}} \right) \left(\prod_{P_k \in corrupt} g_i^{x_{ik}} \right)$$

$\mathcal{A}$ only sees $g_i^{x_{ij}}$ for all $P_j \in honest$ and does not know the discrete logs x_{ij}. Thus, the first product in the expression above is a random element, and so the only way for $\mathcal{A}$ is to manipulate the discrete logs x_{ik} of the corrupt parties. Here, we have two sub-cases:

1. For all $P_k \in \textit{corrupt}$, $\mathcal{A}$ knows all the discrete logs x_{ik}. This will not help $\mathcal{A}$ since $\log_{g_i} h_i = \sum_{P_j \in honest} x_{ij} + \sum_{P_k \in corrupt} x_{ik}$ is random. Thus, this case is reduced to solving the discrete log problem, which succeeds with negligible probability.
2. There is at least one corrupt party P_ℓ with contribution $g_i^{x_{i\ell}}$ such that $\mathcal{A}$ does not know the discrete log $x_{i\ell}$. As described in 1, $\mathcal{A}$ selects an element for P_ℓ, namely $g_\ell^{x_{i\ell}}$, that allows them to compute $\log_{g_i} h_i$. However, for corrupt party P_ℓ's contribution to be accepted, the proof $\pi_{i,x_{e\ell}}$ (which attests that P_ℓ, i.e., $\mathcal{A}$, knows $x_{i\ell}$) must verify correctly. By the simulation extractability property of the proof system, there exists an extractor that can extract $\log_{g_i} g_i^{x_{i\ell}}$ from the proof, implying that $\mathcal{A}$ can break the discrete log assumption, which is a contradiction. Thus, this case happens with a negligible probability.

Thus, the distributed setup generates parameters $\{g_i, h_i\}$ that achieve the same guarantees as the trusted setup, completing the proof. □

Theorem 7 (Theorem 2 restated). *For $W_r \geq W_p + 2\lambda + \Theta(1)$ and $W - 2W_p = \Theta(\lambda)$, the VWRSS scheme from Section 3.2 is perfectly correct, statistically private, and computationally complete (cf. Definition 3).*

Proof. We now show that VWRSS achieves the correctness, privacy, and completeness properties.

Correctness. Correctness of VWRSS relies on the correctness of commitment parameter generation, WRSS, NIZKs, and commitment extraction in the consistency check.

For commitment parameter generation, correctness follows by correctness of CRT and the arithmetic operations performed. Also, by the completeness of the knowledge of the discrete log proof system, each $g_j^{x_{ji}}$ with a valid proof will be accepted. Furthermore, as long as there are at least two honest parties, the computed h_is and h are random group elements whose discrete log with respect to g_i and g, respectively, are unknown.

WRSS is used with the correct parameters to generate S and each $[s]_i$, and a dealer provides a range proof that the committed S is in the range that allows correct reconstruction. By the completeness property of range proofs, an honestly generated proof will be accepted by the verifier. As for the share consistency check, by Lemma 4, a Pedersen commitment to $[s]_i$ can be correctly extracted from the aggregated commitment ct. Thus, honestly generated shares will pass both consistency and range checks and will be accepted by the parties (recall in the correctness property the dealer is honest).

Finally, in the malicious case, we have $W \geq W_r + W_p$ (with $W_r \geq W_p + 2\lambda + \Theta(1)$ we have $W \geq 2W_p + 2\lambda + \Theta(1)$ or $W - 2W_p = \Theta(\lambda)$, as desired) ensures that the the honest parties' total weight is $\geq W_r$, allowing them to reconstruct the secret s.

Privacy. This follows from the statistical privacy of WRSS. We only need to show that the additional components added to VWRSS, namely, the range proof and the commitment to S, do not violate the privacy guarantee of WRSS. That is, they do not reveal any information about S, and subsequently s.

By the zero-knowledge property of the NIZK proof system, the range proof π does not reveal any information about the witness S (assuming that a NIZK proof system that satisfies statistical zero-knowledge is being employed in the construction). The commitment to S, $\mathsf{ct} = g^S h^R \bmod Q$, is perfectly hiding by Lemma 5. Thus, both the proof and the commitment do not reveal any information about S, and overall VWRSS achieves statistical privacy.

Completeness. Proving completeness means showing that if the dealer is malicious, then with overwhelming probability there exists a value $\tilde{s} \in \mathbb{F}_{p_0} \cup \perp$ such that all honest parties hold valid shares of $\tilde{s}$, and they will output $\tilde{s}$ at the end of the reconstruction phase. Thus, an adversary who controls the dealer and a subset of the parties (with a total weight $\leq W_p$) succeeds with negligible probability in influencing this unique value $\tilde{s}$, or its reconstruction, once the share phase is over.

The dealer can break completeness if it can (1) distribute inconsistent shares to honest parties that pass the consistency check, or (2) influence the reconstruction phase to make honest parties recover a different value from the one corresponding to their consistent shares.

For the former, and WLOG, let P_i be the honest party who is given an inconsistent share $[\tilde{s}']_i$ corresponding to the secret $\tilde{s}'$ and all other honest parties are given shares of a secret $\tilde{s}$. Since commitments to

the individual shares can be extracted from the commitment to the blinded $\tilde{s}$, we can focus only on P_i's share commitment, i.e., $\mathsf{ct}'_i = \mathsf{ct}' \bmod q_i$. If $[\tilde{s}']_i$ does not pass the consistency check, then P_i will broadcast a complaint, and all honest parties will reject their shares and output $\perp$, preserving completeness, i.e., the unique value that they agree on is $\tilde{s} = \perp$. Hence, what remains is analyzing when P_i would accept an inconsistent share, which covers two cases:

Case 1—the dealer knows $\log_{g_i} h_i$. By Lemma 5, this case is not possible.

Case 2—the dealer does not know $\log_{g_i} h_i$. Since honest party P_i will accept their inconsistent share only if it passes the consistency check, we know that $g_i^{[\tilde{s}']_i} h_i^{r_i} \equiv g_i^{[\tilde{s}]_i} h_i^{r'_i} \bmod q_i$. Note that the adversary knows $[\tilde{s}]_i$, $[\tilde{s}']_i$, r_i (as the dealer sent these to P_i), and r'_i. Thus, the dealer can compute $\log_{g_i} h_i = ([\tilde{s}']_i - [\tilde{s}]_i)(r'_i - r_i)^{-1} \bmod p_i$, which is a contradiction. Thus, without knowing the discrete log of h_i, the dealer cannot distribute an inconsistent share to P_i.

Based on the above, a malicious dealer can only distribute consistent shares for a unique secret $\tilde{s}$ to the honest parties.

The second option for the adversary to break completeness, as mentioned earlier, is preventing the successful reconstruction of $\tilde{s}$. Suppose the adversary is able to do that, and let $\{[\tilde{s}]_i\}_{i \in honest}$ be the set of consistent shares that belong to the honest parties. We have two cases:

Case 1—$\mathsf{Reconstruct}(\{[\tilde{s}]_i\}_{i \in honest}) = \tilde{s}$. Here, and although honest parties' shares are consistent and allow recovering $\tilde{s}$, the authorized set B includes at least one inconsistent share supplemented by a corrupt party P_k. So, we have $B = \{[\tilde{s}]_i\}_{i \in honest} \cup \{[\tilde{s}']_k\}$ such that $\mathsf{Reconstruct}(B) \neq \tilde{s}$. This means that P_k holds an inconsistent share that passes the consistency check, leading honest parties to include $[\tilde{s}']_k$ in B. This happens if the adversary is able to compute $[\tilde{s}']_k$ such that $g_k^{[\tilde{s}']_k} h_k^{r_k} \equiv \mathsf{ct} \equiv g_k^{[\tilde{s}]_k} h_k^{r'_k} \bmod q_k$ (for example, using a similar strategy to the one in Remark 1 but now the target is P_k and another corrupt party $P_{k'}$ is the one who provides P_k with a contribution based on honest parties' inputs). As discussed previously, by the simulation extractability property of knowledge of the discrete log proof system, this succeeds with negligible probability. Thus, honest parties are able to detect the inconsistent share and exclude it from B.

Case 2—$\mathsf{Reconstruct}(\{[\tilde{s}]_i\}_{i \in honest}) \neq \tilde{s}$. Here, and even if $B = \{[\tilde{s}]_i\}_{i \in honest}$, the consistent honest parties' shares cannot correctly recover $\tilde{s}$. This is only possible if the malicious dealer used an out-of-range blinded secret $\tilde{S} \notin [0, 2^{W_r})$ to generate the consistent shares. However, honest parties only accept their shares if π (the range proof attesting that the blinded secret $\tilde{S}$ committed in ct is within $[0, 2^{W_r})$) verifies. This means that the adversary was able to produce a proof that verifies for an out of range $\tilde{S}$. By the simulation extractability property, there exists an extractor that can extract the witness, i.e., $\tilde{S}$ and the commitment randomness R, from this proof. Thus, the only way for the adversary to do so is by breaking the binding property of the commitment, i.e., it can be opened to a different value (different from the extracted one) that is out of range, which is a contradiction. Thus, this case will succeed with a negligible probability.

Accordingly, the adversary is only able to break the completeness property of VWRSS with negligible probability, as required. □

B.3 Correctness and Security of Proactive VWRSS

Theorem 8 (Theorem 3 restated). *For $W_r \geq W_p + 4\lambda + \Theta(1)$ and $W - 2W_p = \Theta(\lambda)$, the proactive VWRSS scheme from Section 3.4 is perfectly correct, statistically private, and computationally complete.*

Proof. We show that our proactive VWRSS scheme achieves the correctness, privacy, and completeness properties. Throughout this proof, we will assume that the shares $\{[x]_i^0, [x]_i^1\}_{i \in [n]}$ are of the form $\{(x + U_{2^{W_p + 2\lambda}p_0} \bmod p_i), (x + U_{2^{W_p + \lambda}p_0} \bmod p_i)\}_{i \in [n]}$, i.e., they are shares of x generated using $\mathsf{WRSS.Share}$. The shares $\{[x]_i^0, [x]_i^1\}_{i \in [n]}$ (which are actually generated during $\mathsf{Reshare}$) are statistically close to shares of x generated using $\mathsf{WRSS.Share}$ (using a similar argument as in Appendix B.1) and their commitments are statistically private (see Appendix B.2).

Correctness. This follows by the correctness of proactive WRSS and VWRSS, and correctness of the additional verification steps within the reshare protocol. We elaborate on the latter below.

We ensure that the two share sets $\{[x_i]^0_j\}_{j\in[n]}$ and $\{[x_i]^1_j\}_{j\in[n]}$ are for the same x_i using a check over the commitments. For some contribution x_i, P_i computes two commitments $\mathsf{ct}^0_{x_i} = g^{X^0_i} h^{R^0_{x_i}} \bmod Q$ and $\mathsf{ct}^1_{x_i} = g^{X^1_i} h^{R^1_{x_i}} \bmod Q$ for the blinded secrets $X^0_i = x_i + U_L p_0$ and $X^1_i = x_i + U_{L'} p_0$, where $L = 2^{W_p+2\lambda}$ and $L' = 2^{W_p+\lambda}$, respectively. Note that both $R^0_{x_i}, R^1_{x_i} \leftarrow\!\$\ r_i + U_{2^{W+\lambda}p_0}$. The two commitments are for the same x_i if $\mathsf{ct}^0_{x_i} \cdot (\mathsf{ct}^1_{x_i})^{-1} \bmod Q \equiv 1 \bmod q_0$. This works because (the final step follows by a similar argument to the one in Lemma 4):

$$\begin{aligned}
\mathsf{ct}^0_{x_i} \cdot (\mathsf{ct}^1_{x_i})^{-1} \bmod Q &\equiv g^{X^0_i} h^{R^0_{x_i}} \cdot (g^{X^1_i} h^{R^1_{x_i}})^{-1} \bmod q_0 \\
&\equiv g^{X^0_i - X^1_i} h^{R^0_{x_i} - R^1_{x_i}} \bmod q_0 \\
&\equiv (g \bmod q_0)^{X^0_i - X^1_i} (h \bmod q_0)^{R^0_{x_i} - R^1_{x_i}} \bmod q_0 \\
&\equiv g_0^{(x_i + U_L p_0) - (x_i + U_{L'} p_0)} h_0^{(r_{x_i} + U_{2^{W+\lambda}p_0}) - (r_{x_i} + U'_{2^{W+\lambda}p_0})} \bmod q_0 \\
&\equiv g_0^{(U_L - U_{L'}) p_0 \bmod 2p_0} h_0^{(U_{2^{W+\lambda}} - U'_{2^{W+\lambda}}) p_0 \bmod 2p_0} \bmod q_0 \\
&\equiv 1 \bmod q_0
\end{aligned}$$

Accordingly, the set V includes parties who distributed valid two sets of shares of the same x_i to all parties, and generating a refreshed share is based on the shares of the random masks only from this set (the intermediate steps are the same as in $\mathsf{WRSS.Reshare}$ explained earlier). Computing the commitments, ct^0_x and ct^1_x, that correspond to the blinding of x (the secret underlying the mask shares) where $L = 2^{W_p+2\lambda}$, proceeds as follows (similar approach/argument applies to ct^1_x but with $L = 2^{W_p+\lambda}$):

$$\begin{aligned}
\mathsf{ct}^0_x &= \prod_{i\in V} \mathsf{ct}^0_{x_i} \bmod Q \\
&= \prod_{i\in V} g^{X^0_i} h^{R^0_{x_i}} \bmod Q \\
&= g^{\sum_{i\in V}(x_i + U_L p_0)} h^{\sum_{i\in V}(r_{x_i} + U_{2^{W+\lambda}p_0})} \bmod Q
\end{aligned}$$

This is indeed a valid commitment to the share $([x]^0_j, r^0_{x,j})$ because the blinded secret underlying $[x]^0_j$ is equal to the value in the exponent of g:

$$[x]^0_j = \sum_{i\in V} [x_i]^0_j = \underbrace{\sum_{i\in V}(x_i + U_L p_0)}_{\text{The exponent of } g \text{ in } \mathsf{ct}^0_x} \bmod p_j$$

and the randomness of ct^0_x is equal to the value in the exponent of h:

$$r^0_{x,j} = \sum_{i\in V} r^0_{x_i,j} = \underbrace{\sum_{i\in V} r_{x_i} + U_{2^{W+\lambda}p_0}}_{\text{The exponent of } h \text{ in } \mathsf{ct}^0_x} \bmod p_j$$

Notice that this check does not require knowing the blinded secrets X^0_i, only the publicly computable commitments $\mathsf{ct}^0_{x_i}$, for $i \in V$, are needed.

Now we discuss the correctness of computing the commitment for the share $([s-x]^0_i, r_{s-x,i})$. As shown, this requires first computing a valid commitment to the blinded secret underlying the negated share $([-x]^0_i, r^0_{-x,i})$

(where $-x + U'_L p_0$, with $L = 2^{W_p+2\lambda}$, is the blinded secret of $-x$, and $(R^0_{-x}) = -r_x + U'_{2^{W+\lambda}} p_0$ is the randomness of the commitment). Validity of this commitment is shown below:

$$\begin{aligned}
\mathsf{ct}^0_{-x} &= g^{(2^{W_p+2\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} (\mathsf{ct}^0_x)^{-1} \bmod Q \\
&= g^{(2^{W_p+2\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} (g^{\sum_{i\in V}(x_i+U_L p_0)} h^{\sum_{i\in V}(r_{x_i}+U_{2^{W+\lambda}} p_0)})^{-1} \bmod Q \\
&= g^{-\left(\sum_{i\in V} x_i\right)+\left(2^{W_p+2\lambda}+1-\sum_{i\in V} U_L\right)p_0} h^{-\left(\sum_{i\in V} r_{x_i}\right)+\left(2^{W+\lambda}+1-\sum_{i\in V} U_{2^{W+\lambda}}\right)p_0} \bmod Q \\
&= g^{-x+U'_L p_0} h^{-r_x+U'_{2^{W+\lambda}} p_0} \bmod Q
\end{aligned}$$

Based on that, we show that ct^0_{s-x} as computed in our scheme is indeed a valid commitment to the blinded secret underlying the shares of $s - x$ (where $s - x + U''_L p_0$ is the blinded secret of $s - x$):

$$\begin{aligned}
\mathsf{ct}^0_{s-x} &= \mathsf{ct}_s \mathsf{ct}^0_{-x} \bmod Q \\
&= \left(g^{s+U_L p_0} h^{R_s}\right) \left(g^{-x+U'_L p_0} h^{(R^0_{-x})}\right) \bmod Q \\
&= g^{s-x+(U_L+U'_L)p_0} h^{R_s+(R^0_{-x})} \bmod Q \\
&= g^{s-x+U''_L p_0} h^{(R_s+R^0_{-x})} \bmod Q
\end{aligned}$$

Now, we show how reshare indeed generates a valid commitment for the refreshed blinded secret that underlies the refreshed shares. This commitment is computed based on the reconstructed $s - x$ and the commitments to the blinded secret X^1_i, as follows (with $L = 2^{W_p+\lambda}$):

$$\mathsf{ct}'_s = g^{s-x} \mathsf{ct}^1_x = g^{s-x} g^{x+U_L p_0} h^{R^1_x} = g^{s+U_L p_0} h^{R^1_x} \bmod Q$$

Accordingly, the reshare protocol in proactive VWRSS produces correct fresh shares and correct commitments that allow checking their consistency. These shares preserve reconstruction correctness as they have blinding factors within the required range since all arithmetic operations performed preserve this range. We note that our assumption of the shares $\{[x]^0_i, [x]^1_i\}_{i\in[n]}$ being of the form $\{(x + U_{2^{W_p+2\lambda}p_0} \bmod p_i), (x + U_{2^{W_p+\lambda}p_0} \bmod p_i)\}_{i\in[n]}$ does not affect correctness.

Finally, again in the malicious case, we have $W \geq W_r + W_p$ (with $W_r \geq W_p + 4\lambda + \Theta(1)$ we obtain $W \geq 2W_p + 4\lambda + \Theta(1)$ or $W - 2W_p = \Theta(\lambda)$, as desired) ensures that the the total weight of the honest parties $\geq W_r$, allowing all honest parties to reconstruct shares and perform Reshare.

Privacy. To prove statistical privacy of pVWRSS, we need to show that the views of the adversary of the sharings of any two secrets are statistically close.

Consider any two secrets $s_0, s_1 \in \mathbb{Z}_{p_0}$, and any unauthorized set $\bar{B}$ with $\sum_{i\in B} w_i \leq W_p$, corresponding to the corrupt parties. Let $View_b$ be the denote the adversary's view (corrupt shares, commitment, and range proof) when sharing s_b. We will show that:

$$SD(View_0, View_1) \leq 6 \cdot 2^{-\lambda} + \mathsf{negl}_{ZK}(\lambda)$$

For the sake of simplicity, we will exclude the range proof from the view of the adversary, as by the zero-knowledge property of the proof system, the range proof contributes only $\mathsf{negl}_{ZK}(\lambda)$ to the distance between the two views. Let us hence represent $View_b$ as:

$$View_b = ((S_b \bmod P_{\bar{B}}), (R_{s_b} \bmod P_{\bar{B}}), \mathsf{ct}_b)$$

where S_b is the blinded secret of s_b, R_{s_b} is the randomness used in the aggregated commitment ct_b, and $P_{\bar{B}} = \prod_{i\in\bar{B}} p_i$. We note again that is fine to include $(S_b \bmod P_{\bar{B}}), (R_{s_b} \bmod P_{\bar{B}})$ in $View_b$ is instead of the individual corrupt shares shares $\{[s_b]_i, r_{s_b,i}\}_{i\in\bar{B}}$ as there exists a bijective mapping between them via CRT. We define an ideal distribution $Ideal$ as follows:

– Sample $c_S, c_R \leftarrow\$ \mathbb{Z}_{P_{\bar{B}}}$

– For each $i \in \bar{B}$, set $\overline{\mathsf{ct}_i} = g_i^{c_S \bmod p_i} h_i^{c_R \bmod p_i} \bmod q_i$
– For each $j \in [n] - \bar{B}$ (i.e honest parties) sample $\overline{\mathsf{ct}_j} \leftarrow\$ G_{p_j}$
– Compute $\overline{\mathsf{ct}} \equiv \overline{\mathsf{ct}_i} \bmod q_i$ for $i \in [n]$ via CRT.
– Set the ideal distribution as $Ideal = (c_S, c_R, \overline{\mathsf{ct}})$

Since this ideal distribution $Ideal$ is independent of b, if we can show that $SD(View_b, Ideal)$ is $c2^{-\lambda}$, then by the triangle inequality:

$$SD(View_0, View_1) \leq SD(View_0, Ideal) + SD(Ideal, View_1) \leq c'2^{-\lambda}$$

as required.

Firstly, let us show that set of corrupt shares in $View_b$ ($\{(S_b \bmod P_B), (R \bmod P_B)\}$) are statistically close to those in $Ideal$ ($\{c_S, c_R\}$). To show this, we use the following lemma.

Lemma 6 ([35]). *Let $M < L$ be arbitrary integers, p be an arbitrary integer that is coprime with M, and s be any integer. We have:*

$$SD\left((s + pU_L) \bmod M, \quad U_M\right) < M/L$$

By Lemma 6 with $M = P_{\bar{B}} \leq 2^{W_p}$, $L = 2^{W_p+\lambda}$, $p = p_0$, and $S_b = s_b + p_0 U_L$:

$$SD(S_b \bmod P_{\bar{B}}, U_{P_{\bar{B}}}) < \frac{2^{W_p}}{2^{W_p+\lambda}} = 2^{-\lambda}$$

Similarly, by Lemma 6 with $M = P_{\bar{B}} \leq 2^{W_p}$, $L = 2^{W+\lambda} > 2^{W_p+\lambda}$, $p = p_0$, and $R_{s_b} = r_{s_b} + p_0 U_L$:

$$SD(R_{s_b} \bmod P_{\bar{B}}, U_{P_{\bar{B}}}) < \frac{2^{W_p}}{2^{W+\lambda}} < \frac{2^{W_p}}{2^{W_p+\lambda}} = 2^{-\lambda}$$

Since S_b and R_{s_b} are independently sampled, the joint distribution $(S_b \bmod P_{\bar{B}}, R_{s_b} \bmod P_{\bar{B}})$ is $2 \cdot 2^{-\lambda}$-close to $(U_{P_{\bar{B}}}, U_{P_{\bar{B}}}) = (c_S, c_R)$.

We now show that the commitment ct_b is statistically close to $\overline{\mathsf{ct}}$, conditioned on the corrupt shares. Let us fix any realization of $(S_b \bmod P_{\bar{B}}, R_{s_b} \bmod P_{\bar{B}}) = (c_S, c_R)$. By Lemma 4, $\mathsf{ct}_b \bmod q_i = g_i^{c_S \bmod p_i} h_i^{c_R \bmod p_i}$ for all $i \in \bar{B}$, which are exactly the same as the $\overline{\mathsf{ct}_i}$ in $Ideal$. Thus, we have to show that $\mathsf{ct}_b \bmod P_J$ (where $P_J = \prod_{i \notin \bar{B}} p_i$) is statistically close to uniform ($\overline{\mathsf{ct}} \bmod P_J$) conditioned on the corrupt share realization.

By Lemma 4 again, $\mathsf{ct}_b \bmod \prod_{j \notin \bar{B}} q_j = g_J^{S_b \bmod P_J} h_J^{R_{s_b} \bmod P_J}$. Now $R_{s_b} \bmod P_{\bar{B}} = c_R$ implies that $R_{s_b} = r_{s_b} + U_{2^{W+\lambda}} p_0$ is sampled with the restriction $r_{s_b} + U_{2^{W+\lambda}} p_0 \equiv c_R \bmod P_{\bar{B}}$, or that $r_{s_b} + U_{2^{W+\lambda}} \equiv c_R (p_0)^{-1} \bmod P_{\bar{B}}$. Thus, R_{s_b} conditioned on $R_{s_b} \bmod P_{\bar{B}} = c_R$ is of the form (where $k \leftarrow\$ \left\{0, 1, \ldots, \left(\frac{2^{W+\lambda}}{2^{W_p}}\right) - 1\right\}$, i.e., $k \leftarrow\$ U_{2^{W - W_p + \lambda}}$):

$$R_{s_b} = \left((c_R (p_0)^{-1} \bmod P_{\bar{B}}) + k P_{\bar{B}}\right) p_0$$

This in turn will lead to the following:

$$\begin{aligned} R_{s_b} \bmod P_J &= \left((c_R (p_0)^{-1} \bmod P_{\bar{B}}) + U_{2^{W - W_p + \lambda}} P_{\bar{B}}\right) p_0 \bmod P_J \\ &= (c_R (p_0)^{-1} \bmod P_{\bar{B}}) p_0 + U_{2^{W - W_p + \lambda}} P_{\bar{B}} p_0 \bmod P_J \end{aligned}$$

Once again, by Lemma 6 with $M = P_J \leq 2^{W - W_p}$, $L = 2^{W - W_p + \lambda}$, $p = p_0 P_{\bar{B}}$, and $s = (c_R (p_0)^{-1} \bmod P_{\bar{B}}) p_0$:

$$SD((c_R (p_0)^{-1} \bmod P_{\bar{B}}) p_0 + U_{2^{W - W_p + \lambda}} (P_{\bar{B}} p_0) \bmod P_J, U_{P_J}) < \frac{2^{W - W_p}}{2^{W - W_p + \lambda}} = 2^{-\lambda}$$

Finally, consider $S_b \bmod P_J$. We know that h_J is a generator for G_{P_J}, and so the map $z \to g_J^{S_b} h_J^z$ (from $\mathbb{Z}_{P_j}$ to G_{P_J}) is a bijection (and hence so is its inverse). By the data processing inequality:

$$\begin{aligned} & SD\left(g_J^{S_b} h_J^{R_{s_b} \bmod P_J} \mid R_{s_b} \bmod P_J = c_R, \ g_J^{S_b} U_{G_{P_J}}\right) \\ \leq\ & SD\left(h_J^{R_{s_b} \bmod P_J} \mid R_{s_b} \bmod P_{\bar{B}} = c_R, \ U_{G_{P_J}}\right) \\ \leq\ & SD\left(R_{s_b} \bmod P_J \mid R_{s_b} \bmod P_{\bar{B}} = c_R, U_{P_J}\right) < 2^{-\lambda} \end{aligned}$$

Thus, the commitment ct_b is $2^{-\lambda}$ statistically close to $\overline{\mathsf{ct}}$, conditioned on the corrupt shares. Combining this with the previous proved fact that $(S_b \bmod P_{\bar{B}}, R_{s_b} \bmod P_{\bar{B}})$ is $2 \cdot 2^{-\lambda}$-close to $(U_{P_{\bar{B}}}, U_{P_{\bar{B}}}) = (c_S, c_R)$, we get:

$$SD(View_b, Ideal) \leq 2(2^{-\lambda}) + 2^{-\lambda}$$

which after substituting into the triangle inequality would produce:

$$\begin{aligned} SD(View_0, View_1) &\leq SD(View_0, Ideal) + SD(Ideal, View_1) \\ &\leq 3(2^{-\lambda}) + 3(2^{-\lambda}) \end{aligned}$$

Thus, for any two secrets s_0 and s_1, the views of the adversary are $2^{-\lambda}$ statistically close, as required. We note that our assumption that the shares $\{[x]_i^0, [x]_i^1\}_{i\in[n]}$ are of the form $\{(x + U_{2^{W_p+2\lambda}p_0} \bmod p_i), (x + U_{2^{W_p+\lambda}p_0} \bmod p_i)\}_{i\in[n]}$ may lead to that the actual view of a secret s_b is not $View_b$, but rather statistically close to it (for example, if a secret has been refreshed). By the triangle inequality, this only adds $poly(n)2^{-\lambda}$ to the statistical distance between the actual views of two secrets s_0 and s_1, and so we maintain statistical privacy.

Completeness. By the completeness of VWRSS, once the initial share phase is over, there exists a unique value $\tilde{s} \in \mathbb{F}_{p_0} \cup \perp$ such that all honest parties hold shares of $\tilde{s}$ that can be correctly reconstructed. Proving completeness of our proactive VWRSS amounts to proving that any future reshare instances will produce fresh shares of the same $\tilde{s}$ that honest parties can reconstruct.

Since Reshare uses VWRSS for all individual sharing instances, i.e., the share sets of the mask x, by the completeness property of VWRSS, we know that any VWRSS sharing instance will result in having honest parties holding shares for a unique value that can be reconstructed. Thus, the adversary can only influence Reshare to produce a secret $\tilde{s}' \neq \tilde{s}$ by causing, WLOG, an honest party (say P_j) to hold mask shares $([x]_j^0, r_{x,j}^0)$ and $([x']_j^1, r_{x',j}^1)$ such that $x \neq x'$. That is, if this succeeds, honest parties will hold shares corresponding to a secret $\tilde{s} - x + x' \neq \tilde{s}$ when Reshare concludes, thus breaking completeness.

By the completeness of VWRSS, making the honest parties accept these shares means that we have $\mathsf{ct}_{x_i}^0 = g^{x_i + U_L p_0} h^{r_{x_i} + U_{2^{W+\lambda}} p_0} \bmod Q$ and $\mathsf{ct}_{x_i'}^1 = g^{x_i' + U_{L'} p_0} h^{r_{x_i'} + U'_{2^{W+\lambda}} p_0} \bmod Q$, where $L = 2^{W_p+\lambda}$ and $L' = 2^{W_p+2\lambda}$, respectively. The equality check hence becomes:

$$\mathsf{ct}_{x_i}^0 (\mathsf{ct}_{x_i'}^1)^{-1} \bmod q_0 = g_0^{x_i} h_0^{r_{x_i}} (g_0^{x_i'} h_0^{r_{x_i'}})^{-1} = g_0^{x_i - x_i'} h_0^{r_{x_i} - r_{x_i'}}$$

If $\mathsf{ct}_{x_i}^0 (\mathsf{ct}_{x_i'}^1)^{-1} \equiv 1 \bmod q_0$, then it would imply that $\log_{g_0}(h_0) = (x_i - x_i')(r_{x_i'} - r_{x_i})^{-1} \bmod p_0$, i.e the Adversary is able to compute $\log_{g_0}(h_0)$, solving the discrete log problem. Thus, proactive VWRSS is computationally complete. □

C Correctness of Verifiable Arithmetic Operations in WMPC

In this section, we show the correctness of all arithmetic operations that we use in our WMPC protocol. We start with correctness of individual operations, in Appendix C.1 and C.2, while stating the conditions on the blinding ranges of their inputs to preserve correctness. Then, in Appendix C.3, we discuss the impact on the blinding range when these operations are connected within a circuit representing the private computation.

C.1 Local Operations

The input shares of these (local) arithmetic operations have blinding range 2^α.

Addition. The blinding range for the output share of an addition gate is $\alpha + 1$. Thus, reconstruction correctness is preserved when $\alpha < W_r - 1$.

$$\begin{aligned}
[s_1]_i^\alpha.\mathsf{sh} + [s_2]_i^\alpha.\mathsf{sh} &\equiv ((S_1)_\alpha \bmod p_i) + ((S_2)_\alpha \bmod p_i) \bmod p_i \\
&\equiv (s_1 + p_0 U_{2^{\alpha-\lambda}}) + (s_2 + p_0 U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv (s_1 + s_2) + p_0(U_{2^{\alpha-\lambda}} + U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv (s_1 + s_2) + p_0(U_{2^{\alpha-\lambda+1}}) \bmod p_i \\
&\equiv (S_1 + S_2)_{\alpha+1} \bmod p_i \\
&\equiv [s_1 + s_2]_i^{\alpha+1}.\mathsf{sh}
\end{aligned}$$

$$\begin{aligned}
[s_1]_i^\alpha.\mathsf{r} + [s_2]_i^\alpha.\mathsf{r} &\equiv (R_{s_1} \bmod p_i) + (R_{s_2} \bmod p_i) \bmod p_i \\
&\equiv (R_{s_1} + R_{s_2}) \bmod p_i \\
&\equiv [s_1 + s_2]_i^{\alpha+1}.\mathsf{r}
\end{aligned}$$

$$\begin{aligned}
[s_1]_i^\alpha.\mathsf{ct}[s_2]_i^\alpha.\mathsf{ct} &\equiv \left(g^{(S_1)_\alpha} h^{R_{s_1}}\right)\left(g^{(S_2)_\alpha} h^{R_{s_2}}\right) \bmod Q \\
&\equiv \left(g^{s_1 + p_0 U_{2^{\alpha-\lambda}}} h^{R_{s_1}}\right)\left(g^{s_2 + p_0 U_{2^{\alpha-\lambda}}} h^{R_{s_2}}\right) \bmod Q \\
&\equiv g^{((s_1+s_2)+p_0(U_{2^{\alpha-\lambda}}+U_{2^{\alpha-\lambda}}))} h^{R_{s_1}+R_{s_2}} \bmod Q \\
&\equiv g^{(S_1+S_2)_{\alpha+1}} h^{(R_{s_1}+R_{s_2})} \bmod Q \\
&\equiv [s_1 + s_2]_i^{\alpha+1}.\mathsf{ct} \bmod Q
\end{aligned}$$

Scalar addition. The blinding range for the output share of a scalar addition gate is α. Thus, reconstruction correctness is preserved when $\alpha < W_r$.

$$\begin{aligned}
c + [s]_i^\alpha.\mathsf{sh} &\equiv c + (S_\alpha \bmod p_i) \bmod p_i \\
&\equiv c + (s + p_0 U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv (c + s) + p_0 U_{2^{\alpha-\lambda}} \bmod p_i \\
&\equiv (c + S)_\alpha \bmod p_i \\
&\equiv [c + s]_i^\alpha.\mathsf{sh} \bmod p_i
\end{aligned}$$

$$\begin{aligned}
[s]_i^\alpha.\mathsf{r} &\equiv R_s \mod p_i \\
&\equiv [c + s]_i^\alpha.\mathsf{r} \bmod p_i
\end{aligned}$$

$$\begin{aligned}
g^c[s]_i^\alpha.\mathsf{ct} &\equiv g^c \left(g^{S_\alpha} h^{R_s}\right) \bmod Q \\
&\equiv g^c \left(g^{s + p_0 U_{2^{\alpha-\lambda}}} h^{R_s}\right) \bmod Q \\
&\equiv g^{(c+s)+p_0 U_{2^{\alpha-\lambda}}} h^{R_s} \bmod Q \\
&\equiv g^{(c+S)_\alpha} h^{R_s} \bmod Q \\
&\equiv [c + s]^\alpha.\mathsf{ct} \bmod Q
\end{aligned}$$

Negation. The blinding range for the output share of a negation gate is α. Thus, reconstruction correctness is preserved when $\alpha < W_r$.

$$
\begin{aligned}
(2^{\alpha-\lambda}+1)p_0 - ([s]_i^\alpha.\mathsf{sh}) &\equiv (2^{\alpha-\lambda}+1)p_0 - (S_\alpha \bmod p_i) \bmod p_i \\
&\equiv (2^{\alpha-\lambda}+1)p_0 - (s + p_0 U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv -s + p_0(2^{\alpha-\lambda}+1-U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv -s + p_0 U'_{2^{\alpha-\lambda}} \bmod p_i \\
&\equiv (-S)_\alpha \bmod p_i \\
&\equiv [-s]_i^\alpha.\mathsf{sh} \bmod p_i
\end{aligned}
$$

$$
\begin{aligned}
(2^{W+\lambda}+1)p_0 - ([s]_i^\alpha.\mathsf{r}) &\equiv (2^{W+\lambda}+1)p_0 - (R_s \bmod p_i) \bmod p_i \\
&\equiv (2^{W+\lambda}+1)p_0 - (r_s + p_0 U_{2^{W+\lambda}}) \bmod p_i \\
&\equiv -r_s + p_0(2^{W+\lambda}+1-U_{2^{W+\lambda}}) \bmod p_i \\
&\equiv -r_s + p_0(U'_{2^{W+\lambda}}) \bmod p_i \\
&\equiv (R_{-s}) \bmod p_i \\
&\equiv [-s]_i^\alpha.\mathsf{r} \bmod p_i
\end{aligned}
$$

$$
\begin{aligned}
& g^{(2^{\alpha-\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} \left([s]_i^\alpha.\mathsf{ct}\right)^{-1} \\
&\equiv g^{(2^{\alpha-\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} \left(g^{S_\alpha} h^{R_s}\right)^{-1} \bmod Q \\
&\equiv g^{(2^{\alpha-\lambda}+1)p_0} h^{(2^{W+\lambda}+1)p_0} \left(g^{(s+p_0 U_{2^{\alpha-\lambda}})} h^{(r_s+p_0 U_{2^{W+\lambda}})}\right)^{-1} \bmod Q \\
&\equiv g^{(2^{\alpha-\lambda}+1)p_0 - (s+p_0 U_{2^{\alpha-\lambda}})} h^{(2^{W+\lambda}+1)p_0 - (r_s+p_0 U_{2^{W+\lambda}}))} \bmod Q \\
&\equiv g^{(-s+p_0 U'_{2^{\alpha-\lambda}})} h^{(-r_s+p_0 U'_{2^{W+\lambda}})} \bmod Q \\
&\equiv g^{(-S)_\alpha} h^{(R_{-s})} \bmod Q \\
&\equiv [-s]_i^\alpha.\mathsf{ct} \bmod Q
\end{aligned}
$$

Here, $(2^{\alpha-\lambda}+1)p_0$ and $(2^{W+\lambda}+1)p_0$ represent the upper bounds on the values of the blinded secret and the randomness of the original sharing of s.

Scalar multiplication. The blinding range for the output share of a scalar multiplication gate is $\alpha+\log(c) = \alpha+\lambda$ (as $c \in \mathbb{F}_{p_0}$ can have a magnitude of up to 2^λ). Thus, reconstruction correctness is preserved when $\alpha < W_r - \lambda$.

$$
\begin{aligned}
c[s]_i^\alpha.\mathsf{sh} &\equiv c(S \bmod p_i) \bmod p_i \\
&\equiv c(s + p_0 U_{2^{\alpha-\lambda}}) \bmod p_i \\
&\equiv (cs) + cp_0 U_{2^{\alpha-\lambda}} \bmod p_i \\
&\equiv (cs) + p_0 U_{2^\alpha} \bmod p_i \\
&\equiv (cS)_{\alpha+\lambda} \bmod p_i \\
&\equiv [cs]_i^{\alpha+\lambda}.\mathsf{sh} \bmod p_i
\end{aligned}
$$

$$
\begin{aligned}
c[s]_i^{\alpha}.\mathsf{r} &\equiv c(R_s \bmod p_i) \bmod p_i \\
&\equiv c(r_s + p_0 U_{2^{W+\lambda}}) \bmod p_i \\
&\equiv (cr_s) + cp_0 U_{2^{W+\lambda}}) \bmod p_i \\
&\equiv (cr_s) + p_0 U_{2^{W+2\lambda}}) \bmod p_i \\
&\equiv (R_{cs}) \bmod p_i \\
&\equiv [cs]_i^{\alpha+\lambda}.\mathsf{r} \bmod p_i
\end{aligned}
$$

$$
\begin{aligned}
([s]_i^{\alpha}.\mathsf{ct})^c &\equiv \left(g^{S_\alpha} h^{R_s}\right)^c \bmod Q \\
&\equiv \left(g^{(s+p_0 U_{2^{\alpha-\lambda}})} h^{(r_s+p_0 U_{2^{W+\lambda}})}\right)^c \bmod Q \\
&\equiv g^{c(s+p_0 U_{2^{\alpha-\lambda}})} h^{c(r_s+p_0 U_{2^{W+\lambda}})} \bmod Q \\
&\equiv g^{((cs)+cp_0 U_{2^{\alpha-\lambda}})} h^{((cr_s)+cp_0 U_{2^{W+\lambda}})} \bmod Q \\
&\equiv g^{((cs)+p_0 U_{2^{\alpha}})} h^{((cr_s)+p_0 U_{2^{W+2\lambda}})} \bmod Q \\
&\equiv g^{(cS)_{\alpha+\lambda}} h^{(R_{cs})} \bmod Q \\
&\equiv [cs]_i^{\alpha+\lambda}.\mathsf{ct} \bmod Q
\end{aligned}
$$

C.2 Beaver Triple Generation and Multiplication

Beaver triple generation. We show correctness of the various verification and computation steps for the protocol from Figure 5.

$$
\begin{aligned}
\mathsf{ct}_{a_k b_i} &\equiv ([a_k]_i^{\alpha+\lambda}.\mathsf{ct})^{(B_i)_{\alpha+\lambda}} \bmod Q \\
&\equiv (g^{(A_k)_{\alpha+\lambda}} h^{R_{a_k}})^{(B_i)_{\alpha+\lambda}} \bmod Q \\
&\equiv (g^{(A_k)_{\alpha+\lambda}(B_i)_{\alpha+\lambda}} h^{R_{a_k}(B_i)_{\alpha+\lambda}}) \bmod Q \\
&\equiv g^{(a_k+U_{2^\alpha} p_0)(b_i+U'_{2^\alpha} p_0)} h^{(r_{a_k}+U_{2^{W+\lambda}} p_0)(b_i+U'_{2^\alpha} p_0)} \bmod Q \\
&\equiv g^{(a_k b_i+(a_k U'_{2^\alpha}+b_i U_{2^\alpha}+p_0 U_{2^\alpha} U'_{2^\alpha})p_0)} h^{(r_{a_k} b_i+(r_{a_k} U'_{2^\alpha}+b_i U_{2^{W+\lambda}}+p_0 U_{2^{W+\lambda}} U_{2^\alpha})p_0)} \bmod Q \\
&\equiv g^{(a_k b_i+(U_{2\alpha+\lambda})p_0)} h^{(r_{a_k} b_i+U_{2^{W+\alpha+2\lambda}} p_0)} \bmod Q \\
&\equiv g^{(A_k B_i)_{2\alpha+2\lambda}} h^{(R_{a_k}(B_i)_{\alpha+\lambda})} \bmod Q
\end{aligned}
$$

Correctness of the check performed in step 4 in Figure 5 (where the fifth line below follows by Lemma 3):

$$
\begin{aligned}
\mathsf{ct}_{a_k b_j} \bmod q_i &\stackrel{?}{\equiv} g^{(A_k B_j)_{2\alpha+2\lambda}} h^{(R_{a_k}(B_j)_{\alpha+\lambda})} \bmod q_i \\
&\stackrel{?}{\equiv} g^{((A_k)_{\alpha+\lambda})((B_j)_{\alpha+\lambda})} h^{(R_{a_k})(B_j)_{\alpha+\lambda})} \bmod q_i \\
&\stackrel{?}{\equiv} g_i^{((A_k)_{\alpha+\lambda} \bmod \phi(q_i))((B_j)_{\alpha+\lambda} \bmod \phi(q_i))} h_i^{(R_{a_k} \bmod \phi(q_i))((B_j)_{\alpha+\lambda} \bmod \phi(q_i))} \bmod q_i \\
&\stackrel{?}{\equiv} g_i^{((A_k)_{\alpha+\lambda} \bmod 2p_i)((B_j)_{\alpha+\lambda} \bmod 2p_i)} h_i^{(R_{a_k} \bmod 2p_i)((B_j)_{\alpha+\lambda} \bmod 2p_i)} \bmod q_i \\
&\stackrel{?}{\equiv} g_i^{((A_k)_{\alpha+\lambda} \bmod p_i)((B_j)_{\alpha+\lambda} \bmod p_i)} h_i^{(R_{a_k} \bmod p_i)((B_j)_{\alpha+\lambda} \bmod p_i)} \bmod q_i \\
&\stackrel{?}{\equiv} g_i^{([a_k]_i^{\alpha+\lambda}.\mathsf{sh})([b_j]_i^{\alpha+\lambda}.\mathsf{sh})} h_i^{([a_k]_i^{\alpha+\lambda}.\mathsf{r})([b_j]_i^{\alpha+\lambda}.\mathsf{sh})} \bmod q_i
\end{aligned}
$$

Correctness of verifying complaints is as follows:

$$\begin{aligned}
([a_k]_i^{\alpha+\lambda}.\mathsf{ct} \bmod q_i)^{[b_j]_i^{\alpha+\lambda}.\mathsf{sh}} &\equiv (g^{(A_k)_{\alpha+\lambda}} h^{R_{a_k}} \bmod q_i)^{[b_j]_i^{\alpha+\lambda}.\mathsf{sh}} \bmod q_i \\
&\equiv (g_i^{[a_k]_i^{\alpha+\lambda}.\mathsf{sh}} h_i^{[a_k]_i^{\alpha+\lambda}.\mathsf{r}})^{[b_j]_i^{\alpha+\lambda}.\mathsf{sh}} \bmod q_i \\
&\equiv g_i^{([a_k]_i^{\alpha+\lambda}.\mathsf{sh})([b_j]_i^{\alpha+\lambda}.\mathsf{sh})} h_i^{([a_k]_i^{\alpha+\lambda}.\mathsf{r})([b_j]_i^{\alpha+\lambda}.\mathsf{sh})} \bmod q_i \\
&\equiv \mathsf{ct}_{a_k b_j} \bmod q_i
\end{aligned}$$

We now show the correctness of computing a share of the element ab in a Beaver triple, as well as its corresponding commitment, performed in step 5 in Figure 5. We note that this step has each party P_i recompute its share $[b]_i^{\alpha+\lambda}$ according to the updated set V. This means that $[b]_i^{\alpha+\lambda} = \sum_{j \in v} [b_j]_i^{\alpha+\lambda}$ such that $\forall j \in V$, $\mathsf{ct}_{a_k b_j}$ is valid for $k \in [n]$. We make the approximation $2^{2\alpha+2\lambda+\log(n|V|)} \approx 2^{2\alpha+2\lambda}$ (we elaborate on why this does not cause any issues in Appendix C.3).

$$\begin{aligned}
\sum_{k\in[n]} ([a_k]_i^{\alpha+\lambda}.\mathsf{sh})([b]_i^{\alpha+\lambda}.\mathsf{sh}) &= \sum_{k\in[n]} \left([a_k]_i^{\alpha+\lambda}.\mathsf{sh} \left(\sum_{j\in V} [b_j]_i^{\alpha+\lambda}.\mathsf{sh} \right) \right) \bmod p_i \\
&= \sum_{k\in[n]} \left((A_k)_{\alpha+\lambda} \left(\sum_{j\in V} (B_j)_{\alpha+\lambda} \right) \right) \bmod p_i \\
&= \sum_{k\in[n]} \sum_{j\in V} (A_k)_{\alpha+\lambda} (B_j)_{\alpha+\lambda} \bmod p_i \\
&= \sum_{k\in[n]} \sum_{j\in V} (A_k B_j)_{2\alpha+2\lambda} \bmod p_i \\
&= \sum_{k\in[n]} \sum_{j\in V} (a_k b_j) + U_{2^{2\alpha+\lambda}} p_0 \bmod p_i \\
&= ((ab) + U_{2^{2\alpha+\lambda+\log(n|V|)}} p_0) \bmod p_i \\
&= (AB)_{2\alpha+2\lambda} \bmod p_i \\
&= [ab]_i^{2\alpha+2\lambda}.\mathsf{sh}
\end{aligned}$$

$$\begin{aligned}
\sum_{k\in[n]} ([a_k]_i^{\alpha+\lambda}.\mathsf{r})([b]_i^{\alpha+\lambda}.\mathsf{sh}) &= \sum_{k\in[n]} \left([a_k]_i^{\alpha+\lambda}.\mathsf{r} \left(\sum_{j\in V} [b_j]_i^{\alpha+\lambda}.\mathsf{sh} \right) \right) \bmod p_i \\
&= \sum_{k\in[n]} \left(R_{a_k} \left(\sum_{j\in V} (B_j)_{\alpha+\lambda} \right) \right) \bmod p_i \\
&= \sum_{k\in[n]} \sum_{j\in V} R_{a_k} (B_j)_{\alpha+\lambda} \bmod p_i \\
&= R_{ab} \bmod p_i \\
&= [ab]_i^{2\alpha+2\lambda}.\mathsf{r}
\end{aligned}$$

$$
\begin{aligned}
[ab]_i^{2\alpha+2\lambda}.\mathsf{ct} &\equiv \prod_{k\in[n]}\prod_{j\in V} \mathsf{ct}_{a_k b_j} \bmod Q \\
&\equiv \prod_{k\in[n]}\prod_{j\in V} g^{(A_k B_j)_{2\alpha+2\lambda}} h^{(R_{a_k}(B_j)_{\alpha+\lambda})} \bmod Q \\
&\equiv g^{\left(\sum_{k\in[n]}\sum_{j\in V}(A_k B_j)_{2\alpha+2\lambda}\right)} h^{\left(\sum_{k\in[n]}\sum_{j\in V} R_{a_k}(B_j)_{\alpha+\lambda}\right)} \\
&\equiv g^{(AB)_{2\alpha+2\lambda}} h^{R_{ab}} \bmod Q
\end{aligned}
$$

Multiplication. Once we have a valid Beaver triple $\{[a]_i^{\alpha+\lambda}, [b]_i^{\alpha+\lambda}, [ab]_i^{\alpha+2\lambda}\}$, it is straightforward to see that applying the corresponding local arithmetic operations in $\mathsf{WMPC.Mul}$ (Figure 6) on the shares $[s_1]_i^{\alpha}$ and $[s_1]_i^{\alpha}$ results in a valid share $[s_1 s_2]_i^{\alpha+2\lambda}$ of $s_1 s_2$. This follows by the correctness of local arithmetic operations (Appendix C.1) and Beaver triple generation.

C.3 Correctness of Circuit Execution

So far, we established correctness of the individual arithmetic operations. However, a circuit (representing the private computation) usually have many of these operations combined with each other. This raises several issues when it comes to the blinding range of the output shares:

- Addition, scalar multiplication, and multiplication result in an increase in the blinding range of the resultant shares. While we perform degree reduction after every scalar multiplication and multiplication gate, we do not do so for additions. Accumulated increase due to multiple additions may violate reconstruction correctness especially if the resultant shares are fed into another arithmetic gate (like multiplication).
- Degree reduction does not refresh shares exactly back to a blinding range of a fresh share (i.e., $2^{W_p+2\lambda}$) but rather to a blinding range of $|V|2^{W_p+2\lambda} \leq 2^{W_p+2\lambda+\log(n)}$ (where V is the set associated with the random double sharing of the mask x in $\mathsf{WMPC.DegreeReduction}$).
- As mentioned in Appendix C.2, the shares of $[ab]_i^{2\alpha+2\lambda}$ actually have a blinding range of $n|V|2^{2\alpha+2\lambda} \leq 2^{2\alpha+2\lambda+2\log(n)}$.

We address these concerns via the following circuit invariant. For $W_r = 2W_p + 8\lambda$, if the input shares to all arithmetic operations have blinding range $< 2^{W_p+2\lambda+\lambda/4}$, their final output wires (after degree reductions, when needed) also have blinding range $< 2^{W_p+2\lambda+\lambda/4}$. Hence, the final output of every gate can be reconstructed since $2^{W_p+2\lambda+\lambda/4} < 2^{W_r}$, and that the output of a gate can safely be fed into the input of another.[13]

We argue that this variant is preserved. Let the input shares to each arithmetic operation have a blinding range 2^{α} such that $\alpha < W_p + 2\lambda + \lambda/4$.

Scalar addition does not change the blinding range of the output share. Hence, the output shares will have a blinding range $2^{\alpha} < 2^{W_p+2\lambda+\lambda/4}$.

Negation also does not change the blinding range of the output share. So, the output shares will have a blinding range $2^{\alpha} < 2^{W_p+2\lambda+\lambda/4}$. We note that when evaluating a negation gate, each party keeps track

[13] Multiplying $[s_1]_i^{\alpha}$ and $[s_2]_i^{\alpha}$ uses a Beaver triple $\{[a]_i^{\alpha+\lambda}, [b]_i^{\alpha+\lambda}, [ab]_i^{\alpha+2\lambda}\}$ whose generation involves reconstructing $ab-x$ from the shares $[ab-x]^{2\alpha+3\lambda}$. The shares of $ab-x$ actually have a blinding range of $2^{2\alpha+2\lambda+\log(n|V|)+\lambda+\log(|V|} \leq 2^{2\alpha+3\lambda+3\log(n)}$—the term $2\alpha + 2\lambda + \log(n|V|)$ corresponds to the actual blinding range of the shares of ab, the additional λ corresponds to the fact that the shares of x should have blinding range 2^{λ} larger, and the term $\log(|V|)$ is due to the fact that the shares of the mask x are the sum of contributions of parties from the set V associated with the random double sharing. Thus, to guarantee reconstruction correctness, we choose to require $2\alpha + 3\lambda + c\lambda < W_r = 2W_p + 8\lambda$, where $c\lambda$ (for $c > 0$) represents some extra slack that absorbs the $\log(n)$ term ($\lambda \gg \log(n)$). For $c = 1/2$, this leads to the circuit invariant $\alpha < W_p + 2\lambda + \lambda/4$. We note that this choice of c is arbitrary, and that the circuit invariant can be $\alpha < W_p + 2\lambda + d\lambda$ for a d such that $0 < d < 1/2 - (3\log(n)/2\lambda)$.

of the blinding range of the input to the negation gate, and hence can always compute the upper bound $(2^{\alpha-\lambda}+1)p_0$ (see Appendix C.1) without any interaction.

Scalar multiplication increases the blinding range of the output share to $2^{\alpha+\lambda}$, which is $> 2^{W_p+2\lambda+\lambda/4}$ (recall a fresh share has $\alpha = W_p + 2\lambda$, so the condition will be violated even when operating on a fresh share). However, after every scalar multiplication we invoke WMPC.DegreeReduction using a random double sharing $\{[x]_i^{\alpha_1}, [x]_i^{\alpha_2}\}$ with $\alpha_1 = \alpha + 2\lambda$ and $\alpha_2 = W_p + 2\lambda$, which involves the reconstruction of $s - x$ from the shares $[s-x]_i^{\alpha_1}$. Here, note that the blinding range for the shares $[s-x]_i^{\alpha_1}$ is actually $\alpha_1 = |V|2^{\alpha+2\lambda} = 2^{\alpha+2\lambda+\log(|V|)} \leq 2^{\alpha+2\lambda+\log(n)}$ (where V is the set associated with the random double sharing). Still, we can safely reconstruct since $2^{\alpha+2\lambda+\log(n)} < 2^{W_p+4\lambda+\lambda/4+\log(n)} < 2^{W_r}$, and this is why we drop $\log(|V|)$ in the protocol description. Similarly, the blinding range the refreshed share $[s]_i^{\alpha_2}$ (i.e., output of degree reduction) is $\alpha_2 = |V|2^{W_p+2\lambda} = 2^{W_p+2\lambda+\log(|V|)} \leq 2^{W_p+2\lambda+\log(n)} < 2^{W_p+2\lambda+\lambda/4}$, since $\log(n) \ll \lambda$.

Addition increases the blinding range of the output share to $2^{\alpha+1}$. Thus, as long as $2^{\alpha} < 2^{W_p+2\lambda+\lambda/4-1}$, the output share will have a blinding range $< 2^{W_p+2\lambda+\lambda/4}$. All parties can keep track of the blinding range of inputs to addition gates, and if an input has blinding range $2^{\alpha} > 2^{W_p+2\lambda+\lambda/4-1}$, the parties invoke WMPC.DegreeReduction, with $\alpha_1 = \alpha + \lambda$ and $\alpha_2 = W_p + 2\lambda$, to reduce the blinding range and so preserve correctness.

Multiplication uses a Beaver triple $\{[a]_i^{\alpha+\lambda}, [b]_i^{\alpha+\lambda}, [ab]_i^{\alpha+2\lambda}\}$ whose generation involves the reconstruction of $ab - x$ from the shares $[ab-x]_i^{2\alpha+3\lambda}$. The actual blinding range of these shares, as discussed previously, is $2^{2\alpha+3\lambda+\log(n|V|)+\log(|V|)} < 2^{2\alpha+3\lambda+3\log(n)}$. For $\alpha < W_p + 2\lambda + \lambda/4$, we have $2\alpha + 3\lambda + 3\log(n) < 2W_p + 7\lambda + \lambda/2 + 3\log(n) < W_r$ for $\log(n) \ll \lambda$. Hence, the shares of $ab - x$ are still reconstructible, allowing us to safely omit $\log(n|V|)$ and $\log(|V|)$ in the protocol description.

The online portion of the multiplication involves the reconstruction of $(s_1 - a)$ and $(s_2 - b)$ from the shares $[s_1 - a]_i^{\alpha+\lambda}$ and $[s_2 - b]_i^{\alpha+\lambda}$. The shares of $[s_1 - a]_i^{\alpha+\lambda}$ actually have a blinding range $n2^{\alpha+\lambda} = 2^{\alpha+\lambda+\log(n)} < 2^{W_p+3\lambda+\lambda/4+\log(n)} < 2^{W_r}$, meaning that it can be safely reconstructed. Similarly, the shares of $[s_2 - b]_i^{\alpha+\lambda}$ actually have blinding range $|V|2^{\alpha+\lambda} = 2^{\alpha+\lambda+\log(|V|)} \leq 2^{\alpha+\lambda+\log(n)} < 2^{W_p+3\lambda+\lambda/4+\log(n)} < 2^{W_r}$, meaning that it too can be safely reconstructed (again, this is why we drop $\log(n)$ and $\log|V|$ in the protocol description).

Multiplication increases the blinding range of the output share to $2^{\alpha+2\lambda}$, which is $> 2^{W_p+2\lambda+\lambda/4}$. However, similar to scalar multiplication, every multiplication in our MPC protocol is followed by a degree reduction that consumes a random double sharing $\{[x]_i^{\alpha_1}, [x]_i^{\alpha_2}\}$ with $\alpha_1 = W_p + 5\lambda$ and $\alpha_2 = W_p + 2\lambda$.

D Proof of Theorem 4

We prove security of our weighted MPC protocol, denoted as Π, by showing that the view of the real world adversary $\mathcal{A}$ (who can adaptively corrupt a set of servers with a total weight $\leq W_p$, and statically corrupt $< m/3$ clients) in a real world execution of Π computing a functionality f can be generated using an ideal world simulator $\mathcal{S}$ given only the output of f. This is captured by the definition below (adapted from [19] to capture our weighted setting).[14]

Definition 4. *Let $f : X_1 \times \cdots \times X_m \to Y$ be a functionality and let Π be a weighted MPC protocol for computing f with unit-weight m clients and n servers with weights $\{w_i\}_{i\in[n]}$. We say that Π achieves τ-security if for every* PPT *malicious rushing adversary $\mathcal{A}$ in the real world, there exists a* PPT *simulator $\mathcal{S}$ in the ideal world such that for every probabilistic environment $\mathcal{E}$ if $\mathcal{A}$ controls (adaptively-corrupted) $I \subset [n]$ of total weight $\leq \tau \sum_{i\in[n]} w_i$ and (statically-corrupted) $< \tau m$ clients, it holds that for every auxiliary input $z \in \{0,1\}^*$:*

$$\mathsf{IDEAL}_{f,\mathcal{S},I,\mathcal{E}}(z) \stackrel{c}{\equiv} \mathsf{REAL}_{\Pi,\mathcal{A},I,\mathcal{E}}(z)$$

[14] The definition in this and next section captures f with a single output. It is straightforward to extend the definition and our proofs to have m outputs.

In the rest of this section, we present the proof of Theorem 4, which covers overhead analysis (Appendix D.1) and the full construction of the simulator and view indistinguishability (Appendix D.2). Similar to prior work [12,19,26,35,46], in this and the next section, we prove security in a standalone execution. Similar to the simulator in [19], our simulator works in a straight-line fashion (it does not rewind the environment or access its internal state).

D.1 Overhead Analysis

We analyze the overhead of the various phases in our maliciously-secure weighted MPC protocol, and we compare its overhead against [35]. Before that, we note the following. In terms of the overhead of range proofs, and taking folklore bit-decomposition based range proofs as an example, the proof size is linear in the bit-length of the range. This translates to proof size $O(W)$ in our case (blinded secrets lie in the range $[0, 2^{W_r})$, which is of bit-length $O(W)$) which are asymptotically small enough for our use.

Preprocessing Phase. In this phase, servers generate all Beaver triples and random double sharings needed for multiplications and degree reductions, respectively. Every random double sharing results in $O(nW)$ per-server communication complexity—$O(W)$ for shares, $O(W)$ for the commitments, $O(W)$ for range proofs, and $O(nW)$ for complaints (the latter is $O(nw_i)$ per server of weight w_i, which we simplify to $O(nW)$)—and a round complexity of 2 rounds. Every Beaver triple generation (generating $\{[a]_i^{W_p+3\lambda}, [b]_i^{W_p+3\lambda}, [ab]_i^{W_p+4\lambda}\}$) results in $O(n^2W)$ per-server communication complexity—$O(nW)$, $O(nW)$, $O(n^2W)$, and $O(nW)$ for the shares, commitments, complaints, and range proofs respectively for a, $O(W)$, $O(W)$, $O(nW)$, and $O(W)$ for the shares, commitments, complaints, and range proofs respectively for b, $O(nW)$ for Beaver triple commitments (i.e., $\mathsf{ct}_{a_k b_i}$), $O(n^2W)$ for Beaver triple commitment complaints, and $O(w_i)$ for the degree reduction—and a round complexity of 4 rounds.

Online Phase. In this phase, the clients share their inputs among the servers, which results in per-client communication complexity of $O(W)$ ($O(W)$ for the shares, $O(W)$ for the commitment, and $O(W)$ for the range proof). The circuit is processed layer-by-layer, with all gates in a layer being computed in parallel. Local gates do not incur any communication cost, while computing a multiplication results in a per-server communication complexity of $O(w_i)$ and a round complexity of 1 round. Furthermore, performing degree reduction (after each multiplication) results in a per-server communication overhead of $O(w_i)$ and a round complexity of 1. Thus, in the worst case scenario where each layer in the circuit C of depth d has at least one multiplication gate, the total per-server communication overhead is $O(w_i)$ with a round complexity of $2d$.

Output Phase. In this phase, servers handoff their internal state to the clients who generate a fresh Beaver triple and a fresh sharing of 1 to obtain the final output. Handoff requires per-server communication overhead of $O(nW) + O(m^2\lambda)$ (which is $O(W) + O(m\lambda)$ for the shares, $O(W) + O(m\lambda)$ for commitments, $O(nW) + O(m^2\lambda)$ for complaints, $O(W) + O(m\lambda)$ for range proofs, and $O(w_i)$ for broadcasting masked shares) and a round complexity of 2 rounds. Generating the shares of 1 results in a per-client communication overhead of $O(m^2\lambda)$ (for the 2 invocations of $\mathsf{WMPC.ShareRandDouble}$) and round complexity of 3 (2 rounds for $\mathsf{WMPC.ShareRandDouble}$, 1 round for $\mathsf{WMPC.DegreeReduction}$), generating a Beaver triple results in a per-client communication overhead of $O(m^3\lambda)$ and round complexity of 2, and broadcasting the output share requires per-client communication complexity of $O(\lambda)$ and a round complexity of 1.

Comparison to [35]. In comparison to the semi-honest weighted MPC protocol from [35], our protocol has per-server communication overhead of $O(n^2W)$ in the preprocessing and output phases (compared to $O(W)$), and round complexity of $2d$ (compared to d rounds) in the online phase, along with an asymptotic corruption threshold of $< W/3$ (compared to $< W/2$). The additional overhead in our protocol is mainly due to handling malicious security within degree reduction and multiplication.

D.2 Simulator Construction and Security

We define an ideal world adversary $\mathcal{S}$ that interacts with the environment $\mathcal{E}$ and simulates a copy of the real world adversary $\mathcal{A}$. For simplicity, we let $\mathcal{A}$ be a proxy for $\mathcal{E}$. $\mathcal{S}$ receives the auxiliary inputs $(C, w, p_0, g_0, \lambda)$

where C is the circuit representing the function f to be computed, $w = \{w_i\}_{i\in[n]}$ is the set of servers' weights, $\mathbb{F}_{p_0}$ is the field over which C is computed, g_0 is the generator of the subgroup $\mathbb{Z}^*_{q_0}$ of order p_0 with $q_0 = 2p_0 + 1$. $\mathcal{S}$ also receives the final output $y = C(s_1, \ldots, s_m)$ from the ideal functionality, where s_i is the private input of the ith client. The task of $\mathcal{S}$ is to simulate a view for $\mathcal{A}$ in the ideal world that is indistinguishable from the view of $\mathcal{A}$ in the real world when executing Π.

Intuitively, security follows by the security of proactive pVWRSS under honest majority. Honest servers, whose total weight is $\geq W_r$, will always produce valid shares of the circuit output. Also, the additional layer in the output phase, in which clients multiply output shares by a sharing of 1, enables $\mathcal{S}$ to simulate the output phase correctly thus producing the correct output $C(s_1, \ldots, s_m)$, as we elaborate further below. Any invalid shares provided by corrupt servers during the computation are detected and excluded, so they will not impact computation correctness or privacy. As in the real protocol, $\mathcal{S}$ will set the input to a predetermined default value for any corrupt client that distributes invalid shares for its input. Performing computation over verifiable shares under honest majority achieves GOD.

For clarity, we split the security proof by describing the simulation of the various protocol phases, namely, preprocessing, input, circuit execution, and output. We show that across all these phases the simulation either has no error or a negligible one (statistical or computational), making the overall simulated view computationally indistinguishable from the real world view.

Additional notation. Let J be the set of honest servers, so $J = [n] - I$, $I_{\mathcal{C}}$ be the set of corrupt clients, and $J_{\mathcal{C}}$ be the set of honest clients, so $J_{\mathcal{C}} = [m] - I_{\mathcal{C}}$. Let $W_p^{\mathcal{C}}$ and $W_r^{\mathcal{C}}$ be the clients' privacy and reconstruction thresholds, respectively.

Note on $\mathcal{S}$'s access to corrupt clients' inputs, and $\mathcal{A}$'s adaptive server corruption. For each client input, $\mathcal{S}$ receives all shares sent to honest servers. As we assume honest majority, these shares have a total weight $\geq W_r$, which allows $\mathcal{S}$ to reconstruct all corrupt client inputs by invoking pVWRSS.Reconstruct. By the security of VWRSS, only valid shares will be accepted, and a predetermined default value will be used for corrupt clients who distribute invalid shares of their inputs. Recall that clients are statically corrupt, so $\mathcal{A}$ specifies $I_{\mathcal{C}}$ at the onset of the protocol.

We support adaptive server corruption, so $\mathcal{A}$ can choose to corrupt servers throughout protocol execution. Recall that servers do not have private inputs, clients supplement these inputs. Thus, a server's internal state includes the input shares it received from clients, and shares of the intermediate state of the computation (or computation output at the end of the computation). $\mathcal{S}$, as we will see later, will generate valid shares on behalf of honest clients for random inputs (which $\mathcal{A}$ cannot distinguish by the security of VWRSS), and will track the state on behalf of honest servers throughput the simulation by performing the exact steps of the computation. At any point $\mathcal{A}$ corrupts a server, it obtains access to this state. $\mathcal{A}$ cannot corrupt servers with a total weight $> W_p$, thus the additional state it obtains over time by corrupting servers does not impact security or correctness. Crucially, as Beaver triple and random double sharing servers generate are random elements, which $\mathcal{S}$ can generate on its own for honest parties as done in the real world. Thus, $\mathcal{S}$ does not need to equivocate the state of honest servers when get corrupted. As a result, simulation of our protocol for static server corruption is equivalent to adaptive corruption. Again, recall that clients are statically corrupt, so adaptive server corruption does not impact input and output phases.

Note on output computation. As $\mathcal{S}$ does not have access to honest clients' inputs, special care is needed to ensure that the simulated execution output agrees with the real world execution output. We resolve this issue by multiplying the output shares by the shares of 1 in a custom way. In particular, $\mathcal{S}$ generates a special Beaver triple that looks valid to $\mathcal{A}$, but has underlying secrets $(a, b, ab^* \neq ab)$. The shares of a and b are generated honestly, but when generating the shares of $ab^* \neq ab$, $\mathcal{S}$ chooses one honest client (say $\mathcal{C}_1$) and proceeds as follows (o represents the output computed by the servers):

- $\mathcal{S}$ computes $b_1^* = a^{-1}(y - o + ab_1) \bmod p_0$ and finds a $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda} = b_1^* + U_{2^{W_p^{\mathcal{C}}+2\lambda}} p_0 \equiv (B_1)_{W_p^{\mathcal{C}}+3\lambda} \bmod p_{\mathcal{C},i}$ for all corrupt clients $i \in I_{\mathcal{C}}$ (later, in the full construction of $\mathcal{S}$, we show how $\mathcal{S}$ can efficiently sample such a $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}$).
- $\mathcal{S}$, on behalf of $\mathcal{C}_1$, generates and broadcasts commitments $\{\mathsf{ct}_{a_k b_1^*}\}_{k\in[n]}$ using $B^*_{W_p^{\mathcal{C}}+3\lambda,1}$, while for the remaining honest clients, $\mathcal{S}$ generates their commitments using $B_{W_p^{\mathcal{C}}+3\lambda,1}$.

– $\mathcal{S}$ (on behalf of remaining honest clients) does not complain about the commitments from $\mathcal{C}_1$.

By generating a Beaver triple in this way, using it to perform a multiplication will produce the correct output, i.e., the simulated and real execution outputs are the same, as shown below (all computation steps are mod$p_{\mathcal{C},j}$):

$$\begin{aligned}
&(o-a)[1]_j^{W_p^{\mathcal{C}}+2\lambda} + (1-b)[a]_j^{W_p^{\mathcal{C}}+3\lambda} + [ab^*]_j^{W_p^{\mathcal{C}}+4\lambda} \\
&= (o-a)(1+U_{2^{W_p^{\mathcal{C}}+\lambda}}p_0) + (1-b)(a+U_{2^{W_p^{\mathcal{C}}+2\lambda}}p_0) + (ab^*+U_{2^{W_p^{\mathcal{C}}+3\lambda}}p_0) \\
&= (o-a) + (1-b)a + ab^* + \left((o-a)U_{2^{W_p^{\mathcal{C}}+\lambda}} + (1-a)U_{2^{W_p^{\mathcal{C}}+2\lambda}} + U_{2^{W_p^{\mathcal{C}}+3\lambda}}\right)p_0 \\
&= (o-ab+ab^*) + U_{2^{W_p^{\mathcal{C}}+3\lambda}}p_0 \\
&= \left(o-ab+a(b-b_1+a^{-1}(y-o+ab_1))\right) + U_{2^{W_p^{\mathcal{C}}+3\lambda}}p_0 \\
&= (o-ab+ab-ab_1+y-o+ab_1) + U_{2^{W_p^{\mathcal{C}}+3\lambda}}p_0 \\
&= y + U_{2^{W_p^{\mathcal{C}}+3\lambda}}p_0 \\
&= [y]_j^{W_p^{\mathcal{C}}+4\lambda}
\end{aligned}$$

As we noted in Section 4.3, the shares of 1 are generated by first invoking WMPC.ShareRandDouble($\alpha_1 = W_p^{\mathcal{C}}+2\lambda, \alpha_2 = W_p^{\mathcal{C}}+3\lambda$) to get a pair of shares of some random x', subtracting them to get shares of 0 with blinding range $2^{W_p^{\mathcal{C}}+3\lambda}$, invoking WMPC.DegreeReduction to get shares of 0 with a blinding range $2^{W_p^{\mathcal{C}}+2\lambda}$ (whose corresponding random double sharing can be generated in parallel with the previous random double sharing of x'), and then locally adding 1 to obtain shares of 1.

We proceed now with showing the simulation of the various protocol phases, and that the real and ideal world views are indistinguishable.

Setup Phase Simulation Clients and the servers generate their key pairs (used as their identifiers and for signing all messages they send), and the parameters needed for pVWRSS. Simulating the setup phase for servers proceeds as follows:

- For all $j \in J$, $\mathcal{S}$ generates keypairs $(\mathsf{sk}_j, \mathsf{vk}_j)$, as well as (p_j, q_j, g_j) based on w_j, and broadcasts all $(\mathsf{vk}_j, p_j, q_j, g_j)$. $\mathcal{S}$ receives $\{(\mathsf{vk}_i, p_i, q_i, g_i)\}_{i\in I}$ from $\mathcal{A}$.
- For all $j \in J$, $\mathcal{S}$ randomly samples $\{g_k^{x_{kj}} \bmod q_k\}_{k\in[n]\cup\{0\}}$ and broadcasts these values. $\mathcal{S}$ receives $\left\{\{g_k^{x_{ki}}\}_{k\in[n]\cup\{0\}}\right\}_{i\in I}$ from $\mathcal{A}$.
- For all $j \in J$, $\mathcal{S}$ computes $h_j = \prod_{k\in[n]} g_j^{x_{jk}} \bmod q_j$ and $h_0 = \prod_{k\in[n]} g_0^{x_{0k}} \bmod q_0$.
- $\mathcal{S}$ computes g, h, Q as in the protocol.

Simulating the setup phase for the unit-weight clients proceeds in a similar way, generating $g_{\mathcal{C}}, h_{\mathcal{C}}, Q_{\mathcal{C}}$ (note that clients need only to run the setup before the output phase begins). $\mathcal{S}$ simulates the setup phase in an honest way (as in the original protocol), thus the simulated view is indistinguishable from the real world view.

Preprocessing Phase Simulation Servers generate all Beaver triples needed for multiplication, and all random double sharing needed for degree reductions. On behalf of honest servers, $\mathcal{S}$ simulates WMPC.ShareRandDouble and WMPC.GenBeaver (which includes simulating WMPC.ShareRand) in an honest way (as shown below). Overall, the simulated view of the preprocessing phase is identical to the real world view.

WMPC.ShareRand(α)**:** Servers collectively generate a set of shares for some random value x. This is simulated as follows:

– For all $j \in J$, $\mathcal{S}$ samples $x_j \leftarrow\$ \mathbb{F}_{p_0}$ and invokes pVWRSS.Share to share x_j among all servers using parameters (g, h, Q) with $L = 2^{\alpha-\lambda}$. This results in broadcasting $[x_k]_j^\alpha.\mathsf{ct}$ and $[x_k]_j^\alpha.\pi$, and sending the individual shares to corrupt servers in I, on behalf of P_j. $\mathcal{S}$ stores $\{[x_k]_j^\alpha\}_{j\in J}$ for all $k \in J$.
– $\mathcal{S}$ receives $\left\{\{[x_i]_j^\alpha\}_{j\in J}\right\}_{i\in I}$ from $\mathcal{A}$.
– $\mathcal{S}$ simulates pVWRSS.Verify over all individual shares; it does nothing for the shares it generated on behalf of honest servers, while for shares generated by a corrupt servers, if all shares sent by P_i $(i \in I)$ to honest servers are valid, $\mathcal{S}$ adds i in a set V'.
– $\mathcal{S}$ sets $V = J \cup V'$ as the set associated with the random sharing.
– Finally, $\mathcal{S}$ computes (and stores) the following:
  • $\{[x]_j^\alpha\}_{j\in J}$ where $[x]_j^\alpha.\mathsf{sh/r} = \left(\sum_{k\in V}[x_k]_j^\alpha.\mathsf{sh/r}\right) \bmod p_j$
  • $[x]_j^\alpha.\mathsf{ct} = \left(\prod_{k\in V}[x_k]_j^\alpha.\mathsf{ct}\right) \bmod Q$

$\mathcal{S}$ simulates WMPC.ShareRand using simulations of sub-protocols that do not have simulation errors, thus there is no simulation error in this step.

WMPC.ShareRandDouble(α_1, α_2): Servers collectively generate two sets of shares for some random value x. This is simulated as follows:

– For all $j \in J$, $\mathcal{S}$ samples $x_j, r_{x_j} \leftarrow\$ \mathbb{F}_{p_0}$ and invokes pVWRSS.Share to share x_j among all servers using parameters (g, h, Q) twice with $L = 2^{\alpha_1-\lambda}$ and $L = 2^{\alpha_2-\lambda}$ respectively, and randomnesses $R_{x_j,\alpha_1}, R_{x_j,\alpha_2} \leftarrow\$ r_{x_j} + U_{2^{W+\lambda}p_0}$ respectively. This results in broadcasting $[x_k]_j^{\alpha_1}.\mathsf{ct}$, $[x_k]_j^{\alpha_2}.\mathsf{ct}$, $[x_k]_j^{\alpha_1}.\pi$, and $[x_k]_j^{\alpha_2}.\pi$, and sending the individual shares from each set to each corrupt server in I, on behalf of P_j. $\mathcal{S}$ stores $\{[x_k]_j^{\alpha_1}\}_{j\in J}$ and $\{[x_k]_j^{\alpha_2}\}_{j\in J}$ for all $k \in J$.
– $\mathcal{S}$ receives $\left\{\{[x_i]_j^{\alpha_1}\}_{j\in J}, \{[x_i]_j^{\alpha_2}\}_{j\in J}\right\}_{i\in I}$ from $\mathcal{A}$.
– $\mathcal{S}$ simulates pVWRSS.Verify over all individual shares; it does nothing for the shares it generated on behalf of honest servers, while for shares generated by a corrupt server, if all shares sent by P_i $(i \in I)$ to honest servers are valid, $\mathcal{S}$ adds i in a set V'.
– For all $i \in I$, $\mathcal{S}$ checks if $([x_i]_j^{\alpha_1}.\mathsf{ct})([x_i]_j^{\alpha_2}.\mathsf{ct})^{-1} \bmod Q \equiv 1 \bmod q_0$. If the check passes, it adds i to a set V''.
– $\mathcal{S}$ sets $V = J \cup (V' \cap V'')$, which is the set of all servers that generated valid shares of the random value x.
– Finally, $\mathcal{S}$ computes (and stores) the following on behalf of honest servers:
  • $\{[x]_j^{\alpha_1}\}_{j\in J}$ where $[x]_j^{\alpha_1}.\mathsf{sh/r} = \left(\sum_{k\in V}[x_k]_j^{\alpha_1}.\mathsf{sh/r}\right) \bmod p_j$
  • $\{[x]_j^{\alpha_2}\}_{j\in J}$ where $[x]_j^{\alpha_2}.\mathsf{sh/r} = \left(\sum_{k\in V}[x_k]_j^{\alpha_2}.\mathsf{sh/r}\right) \bmod p_j$
  • $[x]_j^{\alpha_1}.\mathsf{ct} = \prod_{k\in V}[x_k]_j^{\alpha_1}.\mathsf{ct} \bmod Q$
  • $[x]_j^{\alpha_2}.\mathsf{ct} = \prod_{k\in V}[x_k]_j^{\alpha_2}.\mathsf{ct} \bmod Q$

$\mathcal{S}$ simulates WMPC.ShareRandDouble using simulations of sub-protocols that do not have simulation errors, thus there is no simulation error in this step.

WMPC.GenBeaver$(\alpha = W_p + 2\lambda)$: Beaver triples are generated independently of the clients' inputs. Simulating their generation proceeds as follows:

– $\mathcal{S}$ simulates $n+1$ invocations of WMPC.ShareRand$(\alpha+\lambda)$ and one invocation of WMPC.ShareRandDouble$(2\alpha+3\lambda, \alpha+2\lambda)$ in parallel. $\mathcal{S}$ stores the following on behalf of honest servers:
  • The shares $\{[a_k]_j^{\alpha+\lambda}\}_{j\in J}$ for each $k \in [n]$.
  • The shares $\{[b]_j^{\alpha+\lambda}\}_{j\in J}$.
  • The blinded secrets $\{(B_j)_{\alpha+\lambda}\}_{j\in J}$ underlying the sharing of each P_j's contribution b_j.
  • The shares $\{[x]_j^{2\alpha+3\lambda}\}_{j\in J}$.
  • The shares $\{[x]_j^{\alpha+2\lambda}\}_{j\in J}$.

  $\mathcal{S}$ also stores the set V of the shares $\{[b]_k^{\alpha+\lambda}\}_{k\in[n]}$.
– $\mathcal{S}$ computes (and stores) the following on behalf of honest servers:
  • $\{[a]_j^{\alpha+\lambda}\}_{j\in J}$ where $[a]_j^{\alpha+\lambda}.\mathsf{sh/r} = (\sum_{k\in[n]}[a_k]_j^{\alpha+\lambda}.\mathsf{sh/r}) \bmod p_j$

- $[a]_j^{\alpha+\lambda}.\mathsf{ct} = \prod_{k\in[n]}[a_k]_j^{\alpha+\lambda}.\mathsf{ct} \bmod Q$

– For all $j \in J$, $\mathcal{S}$ computes $\mathsf{ct}_{a_k b_j} = \left([a_k]_j^{\alpha+\lambda}.\mathsf{ct}\right)^{(B_j)_{\alpha+\lambda}} \bmod Q$ for all $k \in [n]$, and broadcasts these commitments.
– $\mathcal{S}$ receives $\{\mathsf{ct}_{a_k b_i}\}_{k\in[n]}$ for all $i \in I$ from $\mathcal{A}$.
– For all $j \in J$, $\mathcal{S}$ checks whether $g_j^{([a_k]_j^{\alpha+\lambda}.\mathsf{sh})([b_i]_j^{\alpha+\lambda}.\mathsf{sh})} h_j^{([a_k]_j^{\alpha+\lambda}.\mathsf{r})([b_i]_j^{\alpha+\lambda}.\mathsf{sh})} \equiv \mathsf{ct}_{a_k b_i} \bmod q_j$ for all $k \in [n]$ and $i \in V \cap I$. If the check fails for a particular $\mathsf{ct}_{a_k b_i}$, $\mathcal{S}$ broadcasts complaints on behalf of P_j and removes i from V. $\mathcal{S}$ also receives complaints from $\mathcal{A}$ on behalf of corrupted servers; if $\mathcal{S}$ receives a complaint containing $\mathsf{ct}_{\alpha+\lambda,a_k}$ and $[b_l]_i^{\alpha+\lambda}$ where $l \in V$ that is valid, then $\mathcal{S}$ removes l from V.
– $\mathcal{S}$ recomputes the shares of $\{[b]_j^{\alpha+\lambda}\}_{j\in J}$ based on the updated set V.
– $\mathcal{S}$ computes the following:
  - Shares $\{[ab]_j^{2\alpha+2\lambda}.\mathsf{sh} = \sum_{k\in[n]}([a_k]_j^{\alpha+\lambda}.\mathsf{sh})([b]_j^{\alpha+\lambda}.\mathsf{sh}) \bmod p_j\}_{j\in J}$.
  - Randomnesses $\{[ab]_j^{2\alpha+2\lambda}.\mathsf{r} = \sum_{k\in[n]}([a_k]_j^{\alpha+\lambda}.\mathsf{r})([b]_j^{\alpha+\lambda}.\mathsf{sh}) \bmod p_j\}_{j\in J}$.
  - Commitment $[ab]_j^{2\alpha+2\lambda}.\mathsf{ct} = \prod_{k\in[n]}\prod_{l\in V}\mathsf{ct}_{a_k b_l} \bmod Q$.
– $\mathcal{S}$ simulates $\mathsf{WMPC.DegreeReduction}$ on $\{[ab]_k^{2\alpha+2\lambda}\}_{k\in[n]}$ using the previously generated random double sharing $(\{\{[x]_k^{2\alpha+3\lambda}\}, \{[x]_k^{\alpha+2\lambda}\}\}_{k\in[n]})$ to obtain the honest shares $\{[ab]_j^{\alpha+2\lambda}\}_{j\in J}$.

We note that the Beaver triple commitments $\{\mathsf{ct}_{a_k b_i}\}_{k\in[n],i\in V}$ are computationally hiding. Recall that $a_k = \sum_{i\in[n]} a_{k,i}$, where each P_i samples $a_{k,i} \leftarrow\$\, \mathbb{F}_{p_0}$. As a result, $[a_k]_j^{\alpha+\lambda}.\mathsf{ct} = g^{(A_k)_{\alpha+\lambda}} h^{R_{a_k}} = \prod_{i\in[n]} g^{(A_{k,i})_{\alpha+\lambda}} h^{R_{a_{k,i}}}$, where $R_{a_k} = \sum_{i\in[n]} R_{a_{k,i}}$ contains at least one honest $R_{a_{k,i}} = r_{a_{k,i}} + U_{2^{W+\lambda}} p_0$ such that $r_{a_{k,i}}$ is uniform in $\mathbb{F}_{p_0}$ and unknown to $\mathcal{A}$, implying that $r_{a_k} \equiv R_{a_k} \bmod p_0$ is also uniform in $\mathbb{F}_{p_0}$ and unknown to $\mathcal{A}$. Thus, $[a_k]_j^{\alpha+\lambda}.\mathsf{ct} \bmod q_0 = g_0^{a_k} h_0^{r_{a_k}}$ is also uniform over G_{p_0}. Let $X_k = [a_k]_j^{\alpha+\lambda}.\mathsf{ct} \bmod q_0 = g_0^{a_k} h_0^{r_{a_k}}$. For $j \in J$, $\mathcal{A}$'s view of the q_0 component of $\{\mathsf{ct}_{a_k b_j}\}_{k\in[n]}$ reduces to the set of tuples $\{(X_k, X_k^{b_j})\}_{k\in[n]}$, where each X_k is uniform over G_{p_0} and b_j is uniform in $\mathbb{F}_{p_0}$ (b_j is generated by an honest party P_j). By the random self-reducibility of DDH, distinguishing $\{(X_k, X_k^{b_j})\}_{k\in[n]}$ from $\{(X_k, Z_k)\}_k$ (where $Z_k \leftarrow\$\, G_{p_0}$) is equivalent to a single DDH instance in G_{p_0}. Thus, for all $j \in J$, the broadcast commitments $\{\mathsf{ct}_{a_k b_j}\}_{k\in[n]}$ are computationally indistinguishable from $\{\left([a_k]_j^{\alpha+\lambda}.\mathsf{ct}\right)^Z\}_{k\in[n]}$ for an independent random exponent Z where each $[a_k]_j^{\alpha+\lambda}.\mathsf{ct}$ is uniform over G_{p_0}. Thus, the Beaver triple commitments $\{\mathsf{ct}_{a_k b_i}\}_{k\in[n],i\in V}$ are computationally hiding.

$\mathcal{S}$ simulates $\mathsf{WMPC.GenBeaver}$ using simulations of sub-protocols that do not have simulation errors, and generates the view in the same way as the real distribution; hence this step has no simulation error.

Input Phase Simulation During the input phase, $\mathcal{S}$ shares randomly-sampled inputs on behalf of all honest clients, and verifies the shares received from the corrupt clients.

– For all $j \in J_{\mathcal{C}}$, $\mathcal{S}$ samples a random input secret $s_j \leftarrow\$\, \mathbb{F}_{p_0}$, and invokes $\mathsf{pVWRSS.Share}$ with $L = 2^{W_p+\lambda}$ to share that among all servers. $\mathcal{S}$ broadcasts the commitments and the range proofs, and sends the shares $[s_j]_i^{W_p+2\lambda}$ to all corrupt servers $i \in I$. $\mathcal{S}$ stores all $\{[s_k]_j^{W_p+2\lambda}\}_{j\in J}$ (i.e shares of the honest servers) for all $k \in J_{\mathcal{C}}$.
– $\mathcal{S}$ receives $\left\{\{[s_i]_j^{W_p+2\lambda}\}_{j\in J}\right\}_{i\in I_{\mathcal{C}}}$ from $\mathcal{A}$, which correspond to the shares of corrupt clients' inputs sent to honest servers.
– $\mathcal{S}$ invokes $\mathsf{pVWRSS.Verify}$ over all shares it received from $\mathcal{A}$ on behalf of corrupt clients. For any share that is invalid, $\mathcal{S}$ will broadcast a complaint. Also, $\mathcal{S}$ will receive complaints from $\mathcal{A}$ and verify these claims. All claims about shares distributed by honest clients will be invalid ($\mathcal{S}$ generated these in an honest way) and these complaints will be dropped. For any valid complaint about shares from corrupt clients, $\mathcal{S}$ will do nothing (it already detected that and broadcast a complaint). If a single valid complaint against a corrupt client exists, then $\mathcal{S}$ sets the input for this corrupt client to some predetermined default value.

$\mathcal{S}$ randomly samples inputs for honest clients in the simulated view, whereas in the real view, honest clients use their real inputs. However, since these sharings are both generated honestly, and our pVWRSS is secure, the probability that $\mathcal{A}$ can distinguish between the shares is $\mathsf{negl}(\lambda)$ (pVWRSS has statistical privacy). Thus, the input phase has $\mathsf{negl}(\lambda)$ simulation error.

Circuit Evaluation Simulation $\mathcal{S}$ honestly simulates the circuit evaluation over the input shares by simulating the evaluation of the output wires of each gate in the circuit C, as follows (we show the simulation of degree reduction at the end):

- Addition gate: $\mathcal{S}$ performs WMPC.Add for all honest servers.
- Negation gate: $\mathcal{S}$ performs WMPC.Neg for all honest servers.
- Scalar addition gate: $\mathcal{S}$ performs WMPC.scalarAdd for all honest servers.
- Scalar multiplication gate: $\mathcal{S}$ performs WMPC.scalarMul for all honest servers.
- Multiplication gate: $\mathcal{S}$ simulates WMPC.Mul on behalf of all honest servers, consuming a Beaver triple, as shown below.
- Also, after each scalar multiplication and multiplication gate, $\mathcal{S}$ simulates WMPC.DegreeReduction consuming the corresponding preprocessed random double sharing, as shown below.

WMPC.Mul($\{[s_1]_k^\alpha\}_{k\in[n]}, \{[s_2]_k^\alpha\}_{k\in[n]}$): Let the Beaver triple consumed by this protocol be $(\{[a]_k^{\alpha+\lambda}\}_{k\in[n]}, \{[b]_k^{\alpha+\lambda}\}_{k\in[n]}, \{[ab]_k^{\alpha+2\lambda}\}_{k\in[n]})$.

- $\mathcal{S}$ performs steps 1-2 in the protocol on behalf of all honest servers.
- $\mathcal{S}$ receives $\left\{[s_1 - a]_i^{\alpha+\lambda}, [s_2 - b]_i^{\alpha+\lambda}\right\}_{i\in I}$ from $\mathcal{A}$.
- $\mathcal{S}$ invokes pVWRSS.Reconstruct twice, once over $\{[s_1 - a]_k^{\alpha+\lambda}\}_{k\in[n]}$, and another over $\{[s_2 - b]_k^{\alpha+\lambda}\}_{k\in[n]}$, to obtain $s_1 - a$ and $s_2 - b$, respectively.
- $\mathcal{S}$ performs steps 4-6 in the protocol on behalf of all honest servers.

As shown, $\mathcal{S}$ broadcasts masked shares $\{[s_1 - a]_j^{\alpha+\lambda}\}_{j\in J}$ and $\{[s_2 - b]_j^{\alpha+\lambda}\}_{j\in J}$. Since the Beaver triple commitments $\{\mathsf{ct}_{a_k b_i}\}_{k\in[n], i\in V}$ are computationally hiding (as discussed previously), the shares of a and b are computationally indistinguishable from shares of a random mask, which by a similar argument used in proving security of proactive VWRSS, we get that the distribution of these messages is indistinguishable from the distribution of the messages generated in a real execution of WMPC.Mul; hence the simulation error in this step is at most $\mathsf{negl}(\lambda)$.

WMPC.DegreeReduction($\{[s]_l^\alpha\}_{l\in[n]}, \alpha+\lambda, W_p+2\lambda$): Let the random double sharing consumed by this protocol be $\{[x]_k^{\alpha+\lambda}\}_{k\in[n]}$ and $\{[x]_k^{W_p+2\lambda}\}_{k\in[n]}$. $\mathcal{S}$ simulates degree reduction as follows:

- For all $j \in J$, $\mathcal{S}$ does the following:
  - Invoke WMPC.Neg($[x]_j^{\alpha+\lambda}$) to obtain $\{[-x]_j^{\alpha+\lambda}\}_{j\in J}$.
  - Invoke WMPC.Add($[s]_j^\alpha, [-x]_j^{\alpha+\lambda}$) to obtain $\{[s - x]_j^{\alpha+\lambda}\}_{j\in J}$.
  - Broadcast $[s - x]_j^{\alpha+\lambda}$ on behalf of server P_j.
- $\mathcal{S}$ receives the shares $\{[s - x]_i^{\alpha+\lambda}\}_{i\in I}$ from $\mathcal{A}$.
- $\mathcal{S}$ invokes pVWRSS.Reconstruct($\{[s - x]_k^{\alpha+\lambda}\}_{k\in[n]}$) to obtain $s - x$.
- For all $j \in J$, $\mathcal{S}$ invokes WMPC.scalarAdd($s - x, [x]_j^{W_p+2\lambda}$) to obtain $[s]_j^{W_p+2\lambda}$.

$\mathcal{S}$ broadcasts the shares $\{[s-x]_j^{\alpha+\lambda}\}_{j\in J}$. As above, the distribution of these messages is indistinguishable from the distribution of the messages generated in a real execution of WMPC.DegreeReduction. Hence, the simulation error in this step is at most $\mathsf{negl}(\lambda)$.

Output Phase Simulation Servers hold the output shares $\{[o]_k^{W_p+2\lambda}\}_{k\in[n]}$. Clients by now should have have run $\mathsf{pVWRSS.Setup}$ to generate their individual parameters $\{p_{\mathcal{C},i}, q_{\mathcal{C},i}\}_{i\in[m]}$, thresholds $W_p^{\mathcal{C}}$, $W_r^{\mathcal{C}}$, and their commitment parameters $(g_{\mathcal{C}}, h_{\mathcal{C}}^{r_{\mathcal{C}}}, Q_{\mathcal{C}})$. All of these are simulated in a similar way for the setup phase of servers (discussed previously) with unit weights for the clients. The simulation of the output phase proceeds as follows:

- $\mathcal{S}$ simulates $\mathsf{WFMPC.FluidHandoff}(\{[o]_k^{W_p+2\lambda}\}_{k\in[n]}, W_p^{\mathcal{C}}+2\lambda)$ to securely hand over the internal state of the servers to the clients. The simulation of the handoff will be discussed in the next section.
- $\mathcal{S}$ simulates an invocation of $\mathsf{WMPC.ShareRandDouble}(W_p^{\mathcal{C}}+2\lambda, W_p^{\mathcal{C}}+3\lambda)$ and $\mathsf{WMPC.ShareRandDouble}(W_p^{\mathcal{C}}+4\lambda, W_p^{\mathcal{C}}+2\lambda)$. $\mathcal{S}$ stores the following on behalf of the honest clients:
  - The shares $\{[x']_j^{W_p^{\mathcal{C}}+2\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - The shares $\{[x']_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - The shares $\{[x]_j^{W_p^{\mathcal{C}}+4\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - The shares $\{[x]_j^{W_p^{\mathcal{C}}+2\lambda}\}_{j\in J_{\mathcal{C}}}$.

  There is no simulation error in this step.
- For $j\in J_{\mathcal{C}}$, $\mathcal{S}$ does the following:
  - Invoke $\mathsf{WMPC.Neg}([x']_j^{W_p^{\mathcal{C}}+3\lambda})$ to obtain $\{[-x']_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - Invoke $\mathsf{WMPC.Add}([x']_j^{W_p^{\mathcal{C}}+2\lambda}, [-x']_j^{W_p^{\mathcal{C}}+3\lambda})$ to obtain $\{[0]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - Invoke $\mathsf{WMPC.scalarAdd}(1, [0]_j^{W_p^{\mathcal{C}}+3\lambda})$ to obtain $\{[1]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.

  There is no simulation error in this step.
- $\mathcal{S}$ simulates $\mathsf{WMPC.DegreeReduction}$ on $\{[1]_k^{W_p^{\mathcal{C}}+3\lambda}\}_{k\in[m]}$ using the previously generated random double sharing $\{[x]_k^{W_p^{\mathcal{C}}+4\lambda}, [x]_k^{W_p^{\mathcal{C}}+2\lambda}\}_{k\in[m]}$ to obtain $\{[1]_j^{W_p^{\mathcal{C}}+2\lambda}\}_{j\in J_{\mathcal{C}}}$. By the same argument as the simulation error of simulating $\mathsf{WMPC.DegreeReduction}$, the simulation error in this step is at most $\mathsf{negl}(\lambda)$.
- $\mathcal{S}$ simulates $m+1$ invocations of $\mathsf{WMPC.ShareRand}(W_p^{\mathcal{C}}+3\lambda)$ and one invocation of $\mathsf{WMPC.ShareRandDouble}(2W_p^{\mathcal{C}}+7\lambda, W_p^{\mathcal{C}}+4\lambda)$ in parallel to obtain:
  - Shares $\{[a_k]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - Shares $\{[b]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - Blinded secrets $\{(B_j)_{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$ underlying the sharing of each $\mathcal{C}_j$'s contribution b_j.
  - Shares $\{[x]_j^{2W_p^{\mathcal{C}}+7\lambda}\}_{j\in J_{\mathcal{C}}}$.
  - Shares $\{[x]_j^{W_p^{\mathcal{C}}+4\lambda}\}_{j\in J_{\mathcal{C}}}$.

  $\mathcal{S}$ also stores the set V of the shares $\{[b]_k^{W_p^{\mathcal{C}}+3\lambda}\}_{k\in[m]}$. There is no simulation error in this step.
- $\mathcal{S}$ computes (and stores) the following on behalf of honest clients:
  - $\{[a]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$ where $[a]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh}/.\mathsf{r} = (\sum_{k\in[m]}[a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh}/.\mathsf{r}) \bmod p_{\mathcal{C},j}$
  - $[a]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{ct} = \prod_{k\in[m]}[a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{ct}$
- For all $j\in J_{\mathcal{C}}$ except one, say $\mathcal{C}_1$, $\mathcal{S}$ computes and broadcasts (there is no simulation error in this step):

$$\left\{\mathsf{ct}_{a_k b_j} = \left([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{ct}\right)^{(B_j)_{W_p^{\mathcal{C}}+3\lambda}} \bmod Q_{\mathcal{C}}\right\}_{k\in V}$$

- $\mathcal{S}$ invokes $\mathsf{pVWRSS.Reconstruct}(\{[o]_k^{W_p^{\mathcal{C}}+2\lambda}\}_{k\in[m]})$ to obtain the output o, and $\mathsf{pVWRSS.Reconstruct}(\{[a]_k^{W_p^{\mathcal{C}}+3\lambda}\}_{k\in[m]})$ to obtain the element a. If $a\equiv 0 \bmod p_0$, then simulation fails (since $a^{-1} \bmod p_0$, needed in the next step, does not exist). This event occurs with negligible probability $\mathsf{negl}(\lambda)$, which is the simulation error of this step.

– $\mathcal{S}$ computes $b_1^* = b_1 + (y - o)a^{-1} \bmod p_0$, where y is the required output obtained from the trusted party computing the functionality f and then finds a $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}$ such that $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda} = b_1^* + U_L^* p_0 \equiv (B_1)_{W_p^{\mathcal{C}}+3\lambda} \bmod p_{\mathcal{C},i}$ for all $i \in I_{\mathcal{C}}$, where $L = 2^{W_p^{\mathcal{C}}+2\lambda}$. $\mathcal{S}$ then computes and broadcasts (on behalf of client $\mathcal{C}_1$):

$$\left\{ \mathsf{ct}_{a_k b_1^*} = \left([a_k]_1^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{ct} \right)^{(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}} \bmod Q_{\mathcal{C}} \right\}_{k \in V}$$

We now argue that $\mathcal{S}$ is able to efficiently compute such a B_1^*, and that the overall simulation error in this step is $\mathsf{negl}(\lambda)$.

For the existence and efficient sampling of $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}$, by CRT, the system of congruences $(B_1^*) \equiv b_1^* \bmod p_0$ and $(B_1^*) \equiv (B_1)_{W_p^{\mathcal{C}}+3\lambda} \bmod p_{\mathcal{C},i}$ for all $i \in I_{\mathcal{C}}$ has a unique solution modulo $p_0 \prod_{i \in I_{\mathcal{C}}} p_{\mathcal{C},i}$ (say B_1^*). Thus, the set $\{B_1^*, (B_1^* + p_0 P_{\mathcal{C},I}), (B_1^* + 2p_0 P_{\mathcal{C},I}), \ldots, (B_1^* + kp_0 P_{\mathcal{C},I}), \ldots\}$ is the set of all valid solutions to the the system of congruences. Since $P_{\mathcal{C},I} < W_p^{\mathcal{C}}$ and $L = 2^{W_p^{\mathcal{C}}+2\lambda}$, there exist at least $2^{2\lambda}$ integers within this set (namely, $\{B_1^* + kp_0 P_{\mathcal{C},I}\}_{k \in [2^{2\lambda}]}$) such that they simultaneously encode the secret b_1^* ($B_1^* + kp_0 P_{\mathcal{C},I} \bmod p_0 = b_1^*$) and agree with every corrupt client's share ($(B_1^* + kp_0 P_{\mathcal{C},I}) \equiv (B_1)_{W_p^{\mathcal{C}}+3\lambda} \pmod{p_{\mathcal{C},i}}$ for all $i \in I_{\mathcal{C}}$). $\mathcal{S}$ can efficiently sample the required $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}$ from this set by computing B_1^* via CRT, and computing $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda} = B_1^* + U_{2^{2\lambda}} P_{\mathcal{C},I} p_0$.

We show that the simulation error for this step is $\mathsf{negl}(\lambda)$. This involves showing that $\mathcal{A}$'s view in the simulation (involving B_1^*) is computationally indistinguishable from that of the real protocol (involving B_1). Because $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda}$ is sampled uniformly from $\{B_1^* + kp_0 P_{\mathcal{C},I}\}_{k \in [2^{2\lambda}]}$ (containing $2^{2\lambda}$ integers), its distribution conditioned on $\mathcal{A}$'s share view ($(B_1)_{W_p^{\mathcal{C}}+3\lambda} \bmod p_{\mathcal{C},i}$ for all $i \in I_{\mathcal{C}}$) is $2^{-2\lambda}$-statistically close to an honestly generated blinded secret for the value b_1^* (for any two target secrets the number of consistent blinded secrets differs by at most one out of $2^{2\lambda}$). Furthermore, o, and so $y - o$, and so $b_1^* = b_1 + (y - o)a^{-1} \bmod p_0$, are is $2^{-\lambda}$-statistically close to uniform over $\mathbb{F}_{p_0}$; thus the shares $\{[b_1^*]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j \in J_{\mathcal{C}}}$ are $2^{-\lambda}$-statistically close to those of the real execution (i.e $\{[b_1]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j \in J_{\mathcal{C}}}$).

We now show that the commitments $\{\mathsf{ct}_{a_k b_1^*}\}_{k \in [n]}$ are computationally indistinguishable from $\{\mathsf{ct}_{a_k b_1}\}_{k \in [n]}$. Let $[a_k]_1^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{ct} \bmod q_0 = g_0^{a_k} h_0^{r_{a_k}}$ be $X_k \in G_{p_0}$, where r_{a_k} is uniform in $\mathbb{F}_{p_0}$ and by Lemma 5, $\log_{g_0}(h_0)$ is unknown by the distributed setup. Since $(B_1)_{W_p^{\mathcal{C}}+3\lambda} \equiv b_1 \bmod p_0$ and $(B_1^*)_{W_p^{\mathcal{C}}+3\lambda} \equiv b_1^* \bmod p_0$, the adversary's view of these commitments in the real and simulated worlds reduces to the sets of tuples $\{(X_k, X_k^{b_1})\}_{k \in [m]}$ and $\{(X_k, X_k^{b_1^*})\}_{k \in [m]}$ respectively. Since these sets are sampled on independently random X_ks, by the random self-reducibility of DDH, distinguishing between them is equivalent to distinguishing a single DDH instance in G_{p_0}. Thus, the simulation error associated with substituting $\mathsf{ct}_{a_k b_1}$ with $\mathsf{ct}_{a_k b_1^*}$ is $\mathsf{negl}(\lambda)$.

Thus, the overall simulation error in this step is $\mathsf{negl}(\lambda)$.

– $\mathcal{S}$ receives $\{\{\mathsf{ct}_{a_k b_i}\}_{k \in V}\}_{i \in I_{\mathcal{C}}}$ from $\mathcal{A}$.

– For all $j \in J_{\mathcal{C}}$, $\mathcal{S}$ checks whether

$$g_j^{([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh})([b_i]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh})} h_j^{([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{r})([b_i]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh})} \equiv \mathsf{ct}_{a_k b_i} \bmod q_j$$

for all $k \in [m]$ and $i \in (V \cap I_{\mathcal{C}})$. If the check fails for a particular $\mathsf{ct}_{a_k b_i}$, $\mathcal{S}$ broadcasts complaints on behalf of $\mathcal{C}_j$ and adds i to a set V'''. $\mathcal{S}$ also receives complaints from $\mathcal{A}$ on behalf of corrupt clients; if $\mathcal{S}$ receives a complaint containing $\mathsf{ct}_{W_p^{\mathcal{C}}+3\lambda, a_k}$ and $[b_l]_i^{W_p^{\mathcal{C}}+3\lambda}$ where $l \in (V \cap I_{\mathcal{C}})$ that is valid, then $\mathcal{S}$ adds l to a set V''''.

– $\mathcal{S}$ does the following:

- Compute set $V^* = V - (V''' \cup V'''')$.
- Recompute the shares $\{[b]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j \in J_{\mathcal{C}}}$ according to the set V^*. There is no simulation error in this step as no messages are added to the view.

- Compute the shares $\{[b^*]_j^{W_p^{\mathcal{C}}+3\lambda}\}_{j\in J_{\mathcal{C}}}$ as (there is no simulation error in this step as no messages are added to the view):

$$[b^*]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh} = \left((B_1^*)_{W_p^{\mathcal{C}}+3\lambda} + \sum_{k\in(V^*-\{1\})} [b_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh}\right) \bmod p_{\mathcal{C},j}$$

$$[b^*]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{r} = \left(\sum_{k\in V^*} [b_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{r}\right) \bmod p_{\mathcal{C},j}$$

– $\mathcal{S}$ computes the following:

- The shares $\{[ab^*]_j^{2W_p^{\mathcal{C}}+6\lambda}\}_{j\in J_{\mathcal{C}}}$ as:

$$[ab^*]_j^{2W_p^{\mathcal{C}}+6\lambda}.\mathsf{sh} = \left(\sum_{k\in V}\left(\left([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh}\right)\left([b_1^*]_j^{W_p^{\mathcal{C}}3\lambda}.\mathsf{sh}\right) + z_k\right)\right) \bmod p_{\mathcal{C},j}$$

where

$$z_k = \sum_{\ell\in V^*-\{1\}}\left(\left([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{sh}\right)\left([b_\ell]_j^{W_p^{\mathcal{C}}3\lambda}.\mathsf{sh}\right)\right)$$

$$[ab^*]_j^{2W_p^{\mathcal{C}}+6\lambda}.\mathsf{r} = \left(\sum_{k\in V}\left(\left([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{r}\right)\left([b_1^*]_j^{W_p^{\mathcal{C}}3\lambda}.\mathsf{sh}\right) + v_k\right)\right) \bmod p_{\mathcal{C},j}$$

where

$$v_k = \sum_{\ell\in V^*-\{1\}}\left(\left([a_k]_j^{W_p^{\mathcal{C}}+3\lambda}.\mathsf{r}\right)\left([b_\ell]_j^{W_p^{\mathcal{C}}3\lambda}.\mathsf{sh}\right)\right)$$

- The commitment

$$[ab^*]_j^{2W_p^{\mathcal{C}}+6\lambda}.\mathsf{ct} = \left(\prod_{k\in[m]} \mathsf{ct}_{a_k b_1^*}\left(\prod_{l\in V^*-\{1\}} \mathsf{ct}_{a_k b_l}\right)\right) \bmod Q_{\mathcal{C}}$$

– $\mathcal{S}$ simulates $\mathsf{WMPC.DegreeReduction}$ on $\{[ab^*]_k^{2W_p^{\mathcal{C}}+6\lambda)}\}_{k\in[m]}$ using the previously generated random double sharing to obtain $\{[ab^*]_j^{W_p^{\mathcal{C}}+4\lambda}\}_{j\in J_{\mathcal{C}}}$. Note that while corrupt clients hold shares $\{[ab]_i^{2W_p^{\mathcal{C}}+6\lambda)}\}_{i\in I_{\mathcal{C}}}$, due to the way that b^* was chosen, they effectively hold the shares $\{[ab^*]_i^{2W_p^{\mathcal{C}}+6\lambda}\}_{i\in I_{\mathcal{C}}}$, so $\mathsf{WMPC.DegreeReduction}$ will proceed correctly. By a similar argument to proactive VWRSS's security, the distribution of messages generated by $\mathcal{S}$ while simulating $\mathsf{WMPC.DegreeReduction}$ is indistinguishable from the distribution o the messages generated in the real world. Hence, the simulation error in this step is $\mathsf{negl}(\lambda)$.

– $\mathcal{S}$ simulates $\mathsf{WMPC.Mul}(\{[o]_i^{W_p^{\mathcal{C}}+2\lambda}\}_{i\in[m]}, \{[1]_i^{W_p^{\mathcal{C}}+2\lambda}\}_{i\in[m]})$ on behalf of the honest parties using the modified Beaver triple to obtain $\{[y]_j^{W_p^{\mathcal{C}}+4\lambda}\}_{j\in J_{\mathcal{C}}}$.

– For all $j \in J_{\mathcal{C}}$, $\mathcal{S}$ broadcasts $[y]_j^{W_p^{\mathcal{C}}+4\lambda}$ on behalf of client $\mathcal{C}_j$. The messages added to the view by $\mathcal{S}$ while simulating this step are the shares $\{[y]_j^{W_p^{\mathcal{C}}+4\lambda}\}_{j\in J_{\mathcal{C}}}$. The distribution of these messages generated is identical to the distribution of the messages generated in a real execution of this step.

Thus, overall, the simulation error of the output phase is at most $\mathsf{negl}(\lambda)$.

E Overhead and Security Analysis Of Weighted Fluid MPC

E.1 Overhead Analysis

Our protocol achieves malicious security with GOD in the weighted setting (at the expense of non-maximal fluidity) with $O((n_{\ell-1}^2+n_\ell^2)W^\ell+n_{\ell+1}^2W^{\ell+1})$ communication complexity per party in $\mathcal{P}_\ell$, per gate ($O(n_{\ell-1}^2W^\ell)$ to receive a share from $\mathcal{P}_{\ell-1}$ via $\mathsf{FluidMul}$ and $O(n_\ell^2W^\ell + n_{\ell+1}^2W^{\ell+1})$ to perform multiplication and hand off the share to $\mathcal{P}_{\ell+1}$ via $\mathsf{FluidMul}$; the communication complexity for other arithmetic operations is lower, at $O(n_{\ell-1}W^\ell)$ to receive a share from $\mathcal{P}_{\ell-1}$ via $\mathsf{FluidHandoff}$ and $O(n_\ell W^\ell + n_{\ell+1}W^{\ell+1})$ to hand off the share to $\mathcal{P}_{\ell+1}$) and round complexity of $2d + O(1)$. The communication and round complexities of our fluid protocol for the input phase and output phase are similar to our non-fluid protocol:

- Input phase: Per-client communication complexity of $O(W^{\ell=1})$.
- Output phase: Per-server (those in $\mathcal{P}_{d+1}$) communication complexity of $O(n_{d+1}W^{d+1}) + O(m^2\lambda)$ and round complexity 2 to hand off output shares to the clients; per-client complexity of $O(m^3\lambda)$ with round complexity 6 to compute the final output.

Fluid MPC [19] requires $O(n^2)$ communication complexity per party, per gate and a round complexity of $d + O(1)$, but only supports security with abort in the unweighted setting. The works [26, 28] achieve (unweighted) maximal fluidity with GOD at the expense of $O(n^5)$ amortized communication per party, per gate and a round complexity of $5d + O(1)$. As observed, our protocol supports weighted parties and is more efficient than [26,28] at the expense of maximal fluidity. We leave achieving maximal fluidity in the weighted setting as future work.

E.2 Security (Proof of Theorem 5)

We prove security of our fluid weighted MPC protocol (denoted as Π). We show that the view of the real world adversary $\mathcal{A}$ (who adaptively corrupt a set of servers with a total weight $\leq W_p$ in any round, and statically corrupt $< m/3$ clients) in a real world execution of Π computing a function f can be generated using an ideal world simulator $\mathcal{S}$ given only the output of f. This is captured by the definition below (adapted from [19] to capture our weighted setting).

Definition 5. *Let $f : X_1 \times \cdots \times X_m \to Y$ be a functionality and let Π be a fluid weighted MPC protocol for computing f with unit-weight m clients, n servers with weights $\{w_i\}_{i\in[n]}$, and D rounds. We say that Π achieves τ-security if for every PPT malicious rushing adversary $\mathcal{A}$ in the real world, there exists a PPT simulator $\mathcal{S}$ in the ideal world such that for every probabilistic environment $\mathcal{E}$ if $\mathcal{A}$ controls a subset of (R-adaptively corrupted) servers $I_\ell \subset \mathcal{P}_\ell$ of total weight $\leq \tau \sum_{i\in\mathcal{P}_\ell} w_i$, $\forall \ell \in [D]$, where $\mathcal{P}_\ell$ is the server committee for round ℓ, and (statically-corrupted) $< \tau m$ clients, it holds that for every auxiliary input $z \in \{0,1\}^*$:*

$$\mathsf{IDEAL}_{f,\mathcal{S},I,\mathcal{E},\mathcal{F}_{wCommittee}}(z) \stackrel{c}{\equiv} \mathsf{REAL}_{\Pi,\mathcal{A},I,\mathcal{E},\mathcal{F}_{wCommittee}}(z)$$

where $\mathcal{F}_{wCommittee}$ is the functionality that selects round committees.

As mentioned in Section 5, we consider R-adaptive security where corrupting a server in round ℓ means passively corrupting that server in prior rounds (so $\mathcal{A}$ obtains the state of this party in all prior rounds this server participated in). This is done under the condition that the total weight of (active and passive) corrupt servers in any round $\ell' \leq \ell$ is $\leq W_p^{\ell'}$. Due to this honest majority assumption, and as explained in the previous section, adaptive server corruption does not help the adversary (compared to static corruption) and no state equivocation on $\mathcal{S}$'s side is needed. Moreover, also due to honest majority in any round, including the first round, $\mathcal{S}$ received enough shares of corrupt client inputs that allow reconstructing these inputs.

We present our model and the full construction of the simulator capturing all phases in the fluid weighted MPC protocol. We show that across all these phases the simulation either has no error or a negligible one

Functionality $\mathcal{F}_{wCommittee}$

$\mathcal{F}_{wCommittee}$ is parameterized by a security parameter λ, a sampling function $Sample : \mathcal{P} \to \mathcal{P}$, the set of all n participating servers $\mathcal{P}$ and their weights $\{w_i\}_{i\in[n]}$, and a set $P = \phi$.

Nominate: Upon receiving the command $(\mathsf{nominate}, P_i)$ from party P_i, set $P = P \cup \{P_i\}$.

Elect: Upon receiving the command (elect) from the environment $\mathcal{E}$, compute $P' \leftarrow Sample(P)$ and the corresponding thresholds (W_p, W_r) based on the total weight of P'. Then, broadcast (P', W_p, W_r), and set $P = \phi$.

Fig. 10. An ideal functionality for weighted committee election.

in the security parameter (statistical or computational), making the overall simulated view computationally indistinguishable from the view in the real world.

Notation. Let I_ℓ be the set of corrupt servers (both passive and active) controlled by a rushing adversary $\mathcal{A}$ in committee $\mathcal{P}_\ell$ of round ℓ, and J_ℓ be the set of honest servers in that committee, so $J_\ell = [n_\ell] - I_\ell$, where $|\mathcal{P}_\ell| = n_\ell$. Also, let $I_\mathcal{C}$ be the set of corrupt clients, and $J_\mathcal{C}$ be the set of honest clients, so $J_\mathcal{C} = [m] - I_\mathcal{C}$. $\mathcal{S}$ receives the auxiliary input (C, p_0, g_0, λ) where C is the circuit representing the function f to be computed, $\mathbb{F}_{p_0}$ is the field over which C is computed, g_0 is the generator of the subgroup $\mathbb{Z}^*_{q_0}$ of order p_0 with $q_0 = 2p_0+1$. $\mathcal{S}$ also receives the output of the circuit $y = C(s_1, \ldots, s_m)$, where s_i is the ith client input.

Committee election. As mentioned in Section 5, similar to prior work [19, 38, 46], we consider committee election as an external functionality that supplements the protocol with a round committee before a round begins. We model this using the $\mathcal{F}_{wCommittee}$ functionality, captured in Figure 10. For each round $\ell \in [d+1]$, where d is the depth of the circuit C, $\mathcal{F}_{wCommittee}$ outputs committee $\mathcal{P}_\ell \subset \mathcal{P}$. A server receives a $\mathsf{nominate}$ input from the environment $\mathcal{E}$, that it relays to $\mathcal{F}_{wCommittee}$ to indicate willingness to participate in the next round (so election will happen over nominated servers). Before a new round $\ell + 1$ starts, $\mathcal{E}$ sends the command elect to $\mathcal{F}_{wCommittee}$ to finalize the election of $\mathcal{P}_{\ell+1}$ (along with the thresholds for that round based on the total weight of $\mathcal{P}_{\ell+1}$). At that point, $\mathcal{P}_{\ell+1}$ runs $\mathsf{pVWRSS.Setup}$ to generate the committee public parameters. We assume that $\mathcal{E}$ sends enough nominations that allow electing a committee with a total weight that satisfies the thresholds needed for pVWRSS.

Real world execution. In the real world execution of Π, $\mathcal{E}$ passes the inputs for all clients, denoted as $s_1, \ldots, s_m$, which they share with the first round committee $\mathcal{P}_1$ (once again, only valid shares will be accepted, and a predetermined default value will be used for corrupt clients who distribute invalid shares of their inputs). The protocol then proceeds in rounds to compute the circuit C representing f as described in the protocol steps until round $d + 1$ concludes, thus producing $C(s_1, \ldots, s_m)$. The protocol supports R-adaptive corruption of servers and static client corruption. Thus, $\mathcal{E}$ specifies the set of corrupt clients at the onset of the protocol, while it chooses to corrupt servers throughput execution with a retroactive effect, as discussed previously. This execution defines the random variable $\mathsf{REAL}_{\Pi,\mathcal{A},I,\mathcal{E},\mathcal{F}_{wCommittee}}(z)$.

Ideal world execution. The ideal world execution takes place in the presence of a trusted party that computes the functionality f, and it defines the random variable $\mathsf{IDEAL}_{f,\mathcal{S},I,\mathcal{E},\mathcal{F}_{wCommittee}}(z)$. Clients send their inputs to this trusted party who computes the output and returns it back to the clients. The ideal world execution in the presence of $\mathcal{F}_{wCommittee}$ and the ideal world adversary $\mathcal{S}$ follows a very similar model to [19], which we review below for completeness:

- **Input phase.** $\mathcal{E}$ passes inputs to all clients, denoted as $s_1, \ldots, s_m$. It corrupts a set of these clients $I_\mathcal{C}$, which remains fixed until the end of the execution. The clients send their inputs to the trusted party; $\mathcal{S}$ sends inputs on behalf of corrupt clients (and so for those who distribute invalid shares of their inputs, $\mathcal{S}$ will send a predetermined default value).
- **Corruption phase during the rounds.** The trusted party initializes $\ell = 1$. Until $\ell = d + 1$, the following steps are repeated:

1. The trusted party sends ℓ to $\mathcal{S}$ and initializes an append-only list $I_\ell = \phi$, i.e., list of corrupted servers in round ℓ.
2. $\mathcal{S}$ sends (j, i) to the trusted party, where j is the round number and i is the index of the corrupted server $P_i \in \mathcal{P}$ (with corresponding weight w_i) in round $j \leq \ell$. The trusted party appends i to the list I_ℓ. This corruption step can be repeated multiple times.
3. $\mathcal{S}$ sends continue to the trusted party, triggering the trusted party to increment ℓ by 1.

– **Output computation.** The trusted party computes $y = C(s_1, \ldots, s_m)$, and sends this output to $\mathcal{S}$.
– **Outputs.** $\mathcal{S}$ outputs an arbitrary function of its view, and honest clients output y that they received from the trusted party.

We define an ideal world adversary $\mathcal{S}$ that interacts with $\mathcal{E}$ and simulates a copy of the real world adversary $\mathcal{A}$. As before, for simplicity, we let $\mathcal{A}$ be a proxy for $\mathcal{E}$. The task of $\mathcal{S}$ is to simulate a view for $\mathcal{A}$ in the ideal world that is indistinguishable from the view of $\mathcal{A}$ in the real world when executing Π. All of this is done in the presence of $\mathcal{F}_{wCommittee}$. We show the simulator construction by showing how to simulate all the phases in our fluid weighted MPC protocol. This simulator reuses many of the building block simulations from Appendix D.

Setup and Preprocessing Phase Simulation This proceeds in a similar way to weighted MPC in the previous section. Client setup is identical as they are only involved in the input and output phases and they are statically corrupted. The only difference is that now each committee once elected, will run its setup phase to generate pp_ℓ, all random (double) sharings needed in the round. Simulating these proceeds as before. Thus, there is no simulation error in this step.

Input Phase Simulation During this stage, the identities of the servers in committee $\mathcal{P}_1$ are known, along with the public parameters of this committee pp_1. The simulation of this proceeds identically to the input phase simulation in the previous section, except that $\mathcal{S}$ simulates pVWRSS.Share using the parameters (g_1, h_1, Q_1) and $L = 2^{W_p^1 + 2\lambda}$ and sends shares to the servers in committee $\mathcal{P}_1$. The simulation of the input phase has a $\mathsf{negl}(\lambda)$ simulation error.

Execution Phase Simulation During this stage, the circuit C is evaluated round by round, with committee $\mathcal{P}_\ell$ handling round ℓ. A round is split into a computation phase and a handoff phase. Committee $\mathcal{P}_{\ell+1}$ is finalized (and its public parameters $\mathsf{pp}_{\ell+1}$ are known) before the handoff phase of round ℓ begins. For round $\ell = \{1, \ldots, d\}$, these two phases are simulated as follows.

Computation Phase: Servers in committee $\mathcal{P}_\ell$ perform all local computations (multiplication is performed during the handoff phase):

– Addition gate: $\mathcal{S}$ performs WMPC.Add for honest servers in $\mathcal{P}_\ell$.
– Negation gate: $\mathcal{S}$ performs WMPC.Neg for honest servers in $\mathcal{P}_\ell$.
– Scalar addition gate: $\mathcal{S}$ performs WMPC.scalarAdd for honest servers in $\mathcal{P}_\ell$.
– Scalar multiplication gate: $\mathcal{S}$ performs WMPC.scalarMul for honest servers in $\mathcal{P}_\ell$.

Handoff Phase: This phase involves both committee $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$, and is simulated as follows:

– For all output shares $\{[s]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}$, representing results of local gates from above, $\mathcal{S}$ simulates $\mathsf{WFMPC.FluidHandoff}\left(\{[s]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}, \alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda\right)$.
– For all multiplication gates with inputs $[s_1]^{\alpha_\ell}_{\ell,i}$ and $[s_2]^{\alpha_\ell}_{\ell,i}\}$, for $i \in [n_\ell]$, $\mathcal{S}$ simulates $\mathsf{WFMPC.FluidMul}\left(\{[s_1]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}, \{[s_2]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}, \alpha_{\ell+1} = W_p^{\ell+1} + 3\lambda\right)$.

The simulations of the individual building blocks are shown below.

$\mathsf{WFMPC.FluidHandoff}\left(\{[s]^{\alpha_\ell}_{\ell,i}\}_{i\in[n_\ell]}, \alpha_{\ell+1}\right)$**:** This is simulated as follows:

1. $\mathcal{S}$ simulates $\mathsf{WFMPC.ShareRandFluidDouble}(P_\ell, P_{\ell+1}, \alpha_\ell+\lambda, \alpha_{\ell+1})$ to obtain the shares $\{[x]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$ and $\{[x]_{\ell+1,j}^{\alpha_{\ell+1}}\}_{j\in J_{\ell+1}}$.
2. For all $j \in J_\ell$, $\mathcal{S}$ does the following:
   - Invoke $\mathsf{WMPC.Neg}\left([x]_{\ell,j}^{\alpha_\ell+\lambda}\right)$ to obtain $\{[-x]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$.
   - Invoke $\mathsf{WMPC.Add}\left([s]_{\ell,j}^{\alpha_\ell}, [-x]_{\ell,j}^{\alpha_\ell+\lambda}\right)$ to obtain $\{[s-x]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$.
   - Broadcast $\{[s-x]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$.
3. $\mathcal{S}$ receives $\{[s-x]_{\ell,i}^{\alpha_\ell+\lambda}\}_{i\in I_\ell}$ from $\mathcal{A}$.
4. $\mathcal{S}$ invokes $\mathsf{pVWRSS.Reconstruct}\left(\{[s-x]_{\ell,i}^{\alpha_\ell+\lambda}\}_{i\in[n_\ell]}\right)$ to obtain $s-x$.
5. For all $j \in J_{\ell+1}$, $\mathcal{S}$ invokes $\mathsf{WMPC.scalarAdd}\left(s-x, [x]_{\ell+1,j}^{\alpha_{\ell+1}}\right)$ to obtain $\{[s]_{\ell+1,j}^{\alpha_{\ell+1}}\}_{j\in J_{\ell+1}}$.

Overall, using similar arguments as in the previous section, simulating the handoff protocol has a $\mathsf{negl}(\lambda)$ simulation error.

$\mathsf{WFMPC.ShareRandFluidDouble}(P_\ell, P_{\ell+1}, \alpha_\ell, \alpha_{\ell+1})$: This is simulated as follows:

1. For all $j \in J_\ell$, $\mathcal{S}$ samples $x_j, r_{x_j} \leftarrow\$ \mathbb{F}_{p_0}$ and shares it by invoking $\mathsf{pVWRSS.Share}$ twice, once with parameters (g_ℓ, h_ℓ, Q_ℓ), $L = 2^{\alpha_\ell-\lambda}$, $(R_{x_j})_{\alpha_\ell} \leftarrow\$\, r_{x_j} + U_{2^{W^{\ell+\lambda}}} p_0$ and another with parameters $(g_{\ell+1}, h_{\ell+1}, Q_{\ell+1})$, $L = 2^{\alpha_{\ell+1}-\lambda}$, $(R_{x_j})_{\alpha_{\ell+1}} \leftarrow\$\, r_{x_j} + U_{2^{W^{\ell+1+\lambda}}} p_0$. $\mathcal{S}$ sends shares from the first set to I_ℓ (corrupt servers in committee $\mathcal{P}_\ell$), and sends shares from the second set to $I_{\ell+1}$ (corrupt servers in committee $\mathcal{P}_{\ell+1}$). $\mathcal{S}$ broadcasts $[x_j]_\ell^{\alpha_\ell}.\mathsf{ct}$, $[x_j]_{\ell+1}^{\alpha_{\ell+1}}.\mathsf{ct}$, $[x_j]_\ell^{\alpha_\ell}.\pi$ and $[x_j]_{\ell+1}^{\alpha_{\ell+1}}.\pi$ for all $j \in J_\ell$ and stores all the shares.
2. $\mathcal{S}$ receives $\left\{\{[x_i]_{\ell,j}^{\alpha_\ell}\}_{j\in J_\ell}, \{[x_i]_{\ell+1,j}^{\alpha_{\ell+1}}\}_{j\in J_{\ell+1}}\right\}_{i\in I_\ell}$ from $\mathcal{A}$.
3. $\mathcal{S}$ simulates $\mathsf{pVWRSS.Verify}$ over $\{[x_k]_{\ell,i}^{\alpha_\ell}\}_{i\in[n_\ell]}$ (while involving $\mathcal{P}_\ell$) and over $\{[x_k]_{\ell+1,i}^{\alpha_{\ell+1}}\}_{i\in[n_{\ell+1}]}$ (while involving $\mathcal{P}_{\ell+1}$) for all $k \in [n_\ell]$; it does nothing for the shares it generated on behalf of honest servers, while if all shares sent by $P_{\ell,i} \in I_\ell$ to honest servers are valid, $\mathcal{S}$ adds i to a set V'.
4. For all $P_{\ell,i} \in I_\ell$, $\mathcal{S}$ checks if
$$([x_i]_\ell^{\alpha_\ell}.\mathsf{ct} \bmod q_0)\left([x_i]_{\ell+1}^{\alpha_{\ell+1}}.\mathsf{ct} \bmod q_0\right)^{-1} \equiv 1 \bmod q_0$$
and adds all $P_{\ell,i}$, for which the check passes, to a set V''.
5. $\mathcal{S}$ sets $V = J_\ell \cup (V' \cap V'')$ as the set associated with this random fluid sharing between committees $\mathcal{P}_\ell$ and $\mathcal{P}_{\ell+1}$.
6. Finally, $\mathcal{S}$ computes and stores the following:
   - $\{[x]_{\ell,j}^{\alpha_\ell}\}_{j\in J_\ell}$, where $[x]_{\ell,j}^{\alpha_\ell}.\mathsf{sh/r} = \sum_{k\in V}[x_k]_{\ell,j}^{\alpha_\ell}.\mathsf{sh/r} \bmod p_{\ell,j}$.
   - $\{[x]_{\ell+1,j}^{\alpha_{\ell+1}}\}_{j\in J_{\ell+1}}$, where $[x]_{\ell+1,j}^{\alpha_{\ell+1}}.\mathsf{sh/r} = \sum_{k\in V}[x_k]_{\ell+1,j}^{\alpha_{\ell+1}}.\mathsf{sh/r} \bmod p_{\ell+1,j}$.
   - $[x]_\ell^{\alpha_\ell}.\mathsf{ct} = \prod_{k\in V}[x_k]_\ell^{\alpha_\ell}.\mathsf{ct} \bmod Q_\ell$.
   - $[x]_{\ell+1}^{\alpha_{\ell+1}}.\mathsf{ct} = \prod_{k\in V}[x_k]_{\ell+1}^{\alpha_{\ell+1}}.\mathsf{ct} \bmod Q_{\ell+1}$.

Overall, using similar arguments as in the previous section, simulating share random double in the fluid setting has a $\mathsf{negl}(\lambda)$ simulation error.

$\mathsf{WFMPC.FluidMul}\left(\{[s_1]_{\ell,i}^{\alpha_\ell}\}_{i\in[n_\ell]}, \{[s_2]_{\ell,i}^{\alpha_\ell}\}_{i\in[n_\ell]}, \alpha_{\ell+1}\right)$: It is simulated as follows:

1. $\mathcal{S}$ simulates $(n_\ell+1)$ invocations of $\mathsf{WFMPC.ShareRandFluidDouble}(\mathcal{P}_\ell, \mathcal{P}_{\ell+1}, \alpha_\ell+\lambda, \alpha_{\ell+1}-\lambda)$ and one invocation of $\mathsf{WFMPC.ShareRandFluidDouble}(\mathcal{P}_\ell, \mathcal{P}_{\ell+1}, 2\alpha_\ell+3\lambda, \alpha_{\ell+1})$ in parallel. This results in $\mathcal{S}$ obtaining:
   - Shares $\{[a_k]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$ for each $k \in [n_\ell]$.
   - Shares $\{[a_k]_{\ell+1,j}^{\alpha_{\ell+1}-\lambda}\}_{j\in J_{\ell+1}}$ for each $k \in [n_\ell]$.
   - Shares $\{[b]_{\ell,j}^{\alpha_\ell+\lambda}\}_{j\in J_\ell}$.
   - Shares $\{[b]_{\ell+1,j}^{\alpha_{\ell+1}-\lambda}\}_{j\in J_{\ell+1}}$.

- Blinded secrets $\{(B_j)_{\alpha_\ell+\lambda}\}_{j\in J_\ell}$ underlying the sharing of each $P_{\ell,j}$'s contribution b_j for the sharing $\{[b]^{\alpha_\ell+\lambda}_{\ell,k}\}_{k\in[n_\ell]}$.
- Shares $\{[x]^{2\alpha_\ell+3\lambda}_{\ell,j}\}_{j\in J_\ell}$.
- Shares $\{[x]^{\alpha_{\ell+1}}_{\ell+1,j}\}_{j\in J_{\ell+1}}$.

$\mathcal{S}$ also stores the set V of the shares $\{[b]^{\alpha_\ell+\lambda}_{\ell,k}\}_{k\in[n_\ell]}$ and $\{[b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,k}\}_{k\in[n_{\ell+1}]}$.

2. $\mathcal{S}$ computes (and stores) the following on behalf of the honest servers:
   - $\{[a]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in J_\ell}$, where $[a]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}/\mathsf{r} = \left(\sum_{k\in[n_\ell]}[a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}/\mathsf{r}\right) \bmod p_{\ell,j}$.
   - $\{[a]^{\alpha_{\ell+1}+\lambda}_{\ell+1,j}\}_{j\in J_{\ell+1}}$, where $[a]^{\alpha_{\ell+1}+\lambda}_{\ell+1,j}.\mathsf{sh}/\mathsf{r} = \left(\sum_{k\in[n_\ell]}[a_k]^{\alpha_{\ell+1}+\lambda}_{\ell+1,j}.\mathsf{sh}/\mathsf{r}\right) \bmod p_{\ell+1,j}$.
   - $[a]^{\alpha_\ell+\lambda}_{\ell}.\mathsf{ct} = \prod_{k\in[n_\ell]}[a_k]^{\alpha_\ell+\lambda}_{\ell}.\mathsf{ct} \bmod Q_\ell$.
   - $[a]^{\alpha_{\ell+1}}_{\ell+1}.\mathsf{ct} = \prod_{k\in[n_\ell]}[a_k]^{\alpha_{\ell+1}}_{\ell+1}.\mathsf{ct} \bmod Q_{\ell+1}$.
3. For all $j\in J_\ell$, $\mathcal{S}$ computes and broadcasts the following:

$$\left\{\mathsf{ct}_{a_kb_j} = \left([a]^{\alpha_\ell+\lambda}_{\ell}.\mathsf{ct}\right)^{(B_j)_{\alpha_\ell+\lambda}} \bmod Q_\ell\right\}_{k\in[n_\ell]}$$

4. $\mathcal{S}$ receives $\left\{\{\mathsf{ct}_{a_kb_i}\}_{k\in[n_\ell]}\right\}_{i\in I_\ell}$ from $\mathcal{A}$.
5. For all $j\in J_\ell$, for all $k\in[n_\ell]$, and for all $i\in V$, $\mathcal{S}$ checks whether

$$g^{\left([a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}\right)\left([b_i]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}\right)}_{\ell,j} h^{\left([a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{r}\right)\left([b_i]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}\right)}_{\ell,j} \equiv \mathsf{ct}_{a_kb_i} \bmod q_{\ell,j}$$

   If the check does not pass for a particular $\mathsf{ct}_{a_kb_i}$, $\mathcal{S}$ broadcasts $\mathsf{ct}_{a_kb_i}$ and $[b_i]^{\alpha_\ell+\lambda}_{\ell,j}$ on behalf of $P_{\ell,j}$. $\mathcal{S}$ also removes the corresponding i from V.
6. $\mathcal{S}$ waits to receive any complaints from $\mathcal{A}$ on behalf of I_ℓ. If a complaint against some $k\in V$ is valid, $\mathcal{S}$ removes k from V.
7. $\mathcal{S}$ recomputes the shares $\{[b]^{\alpha_\ell+\lambda}_{\ell,j}\}_{j\in J_\ell}$ and $\{[b]^{\alpha_{\ell+1}-\lambda}_{\ell+1,j}\}_{j\in J_{\ell+1}}$ according to the updated set V.
8. $\mathcal{S}$ computes the following:
   - Shares $\{[ab]^{2\alpha_\ell+2\lambda}_{\ell,j}.\mathsf{sh} = \sum_{k\in[n_\ell]}([a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh})([b]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}) \bmod p_{\ell,j}\}_{j\in J_\ell}$.
   - Randomnesses $\{[ab]^{2\alpha_\ell+2\lambda}_{\ell,j}.\mathsf{r} = \sum_{k\in[n_\ell]}([a_k]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{r})([b]^{\alpha_\ell+\lambda}_{\ell,j}.\mathsf{sh}) \bmod p_{\ell,j}\}_{j\in J_\ell}$.
   - Commitment $[ab]^{2\alpha_\ell+2\lambda}_{\ell}.\mathsf{ct} = \prod_{k\in[n_\ell]}\prod_{l\in V}\mathsf{ct}_{a_kb_l} \bmod Q_\ell$.
9. $\mathcal{S}$ performs step 6 in the protocol on the behalf of all honest servers in $\mathcal{P}_\ell$.
10. $\mathcal{S}$ invokes $\mathsf{pVWRSS.Reconstruct}\left(\{[s_1-a]^{\alpha_\ell+\lambda}_{\ell,k}\}_{k\in[n_\ell]}\right)$ to obtain s_1-a, $\mathsf{pVWRSS.Reconstruct}\left(\{[s_2-b]^{\alpha_\ell+\lambda}_{\ell,k}\}_{k\in[n_\ell]}\right)$ to obtain s_2-b, and lastly $\mathsf{pVWRSS.Reconstruct}\left(\{[ab-x]^{2\alpha_\ell+3\lambda}_{\ell,k}\}_{k\in[n_\ell]}\right)$ to obtain $ab-x$.
11. $\mathcal{S}$ performs step 8 in the protocol on the behalf of all honest servers in $\mathcal{P}_{\ell+1}$.

Overall, using similar arguments as in the previous section, simulating multiply in the fluid setting has a $\mathsf{negl}(\lambda)$ simulation error.

Output Phase Simulation This stage starts at the beginning of round $d+1$. Here, the round committee $\mathcal{P}_{d+1}$ holds shares of the private computation output $\{[o]^{W_p^{d+1}+3\lambda}_{d+1,k}\}_{k\in[n_{d+1}]}$ to be shared with the clients $\mathcal{C}$. To accomplish this, committee $\mathcal{P}_{d+1}$ hands off $\{[o]^{W_p^{d+1}+3\lambda}_{d+1,k}\}_{k\in[n_{d+1}]}$ to $\mathcal{C}$, by executing $\mathsf{WFMPC.FluidHandoff}$. Consequently, during the handoff phase of round $d+1$, $\mathcal{S}$ simulates an execution of $\mathsf{WFMPC.FluidHandoff}\left(\{[o]^{W_p^{d+1}+3\lambda}_{d+1,i}\}_{i\in[n_{d+1}]}, \alpha_{d+2} = W_p^{\mathcal{C}}+2\lambda\right)$, resulting in $\mathcal{C}$ holding the shares $\{[o]^{W_p^{\mathcal{C}}+2\lambda}_{\mathcal{C},i}\}_{i\in[m]}$.

With the completion of this handoff, $\mathcal{C}$ finish the protocol by following the simulator for the output phase from the previous section. Overall, simulating the output phase has a $\mathsf{negl}(\lambda)$ simulation error.