Perfectly-Secure Multiparty Computation with Linear Communication Complexity over Any Modulus

Daniel Escudero[1], Yifan Song[2,3], and Wenhao Wang[4]

[1] J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE, NY, USA
[2] Institute for Theoretical Computer Science, Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing, P. R. China
[3] Shanghai Qi Zhi Institute, Beijing, P. R. China
[4] Yale University, CT, USA

Abstract. Consider the task of secure multiparty computation (MPC) among n parties with perfect security and guaranteed output delivery, supporting $t < n/3$ active corruptions. Suppose the arithmetic circuit C to be computed is defined over a finite ring $\mathbb{Z}/q\mathbb{Z}$, for an arbitrary $q \in \mathbb{Z}$. It is known that this type of MPC over such ring is possible, with communication that scales as $O(n|C|)$, *assuming* that q scales as $\Omega(n)$. However, for *constant-size* rings $\mathbb{Z}/q\mathbb{Z}$ where $q = O(1)$, the communication is actually $O(n \log n|C|)$ due to the need of the so-called ring extensions. In most natural settings, the number of parties is variable but the "datatypes" used for the computation are fixed (*e.g.* 64-bit integers). In this regime, no protocol with linear communication exists.

In this work we provide an MPC protocol in this setting: perfect security, G.O.D. and $t < n/3$ active corruptions, that enjoys linear communication $O(n|C|)$, even for *constant-size* rings $\mathbb{Z}/q\mathbb{Z}$. This includes as important particular cases small fields such as $\mathbb{F}_2$, and also the ring $\mathbb{Z}/2^k\mathbb{Z}$. The main difficulty in achieving this result is that widely used techniques such as linear secret-sharing cannot work over constant-size rings, and instead, one must make use of ring extensions that add $\Omega(\log n)$ overhead, while *packing* $\Omega(\log n)$ ring elements in each extension element in order to amortize this cost. We make use of reverse multiplication-friendly embeddings (RMFEs) for this packing, and adapt recent techniques in *network routing* (Goyal *et al.* CRYPTO'22) to ensure this can be efficiently used for non-SIMD circuits. Unfortunately, doing this naively results in a restriction on the minimum width of the circuit, which leads to an extra additive term in communication of $\mathsf{poly}(n) \cdot \mathsf{depth}(C)$. One of our biggest technical contributions lies in designing novel techniques to overcome this limitation by packing elements that are distributed across *different* layers. To the best of our knowledge, all works that have a notion of packing (*e.g.* RMFE or packed secret-sharing) group gates across the same layer, and not doing so, as in our work, leads to a unique set of challenges and complications.

1 Introduction

In secure multiparty computation, or MPC for short, a set of parties $P_1, \ldots, P_n$, each having an input x_i, want to compute a function of their inputs $y = f(x_1, \ldots, x_n)$ in such a way that only the output y is produced, and nothing additional about the parties' inputs is revealed. This can be done assuming a trusted party that receives the inputs, computes the function and returns the output while promising to leak nothing else, but the goal in MPC is to achieve the same guarantees without relying on such trusted party, using only communication among the parties. MPC protocols are resilient against a potential coalition of *corrupted* parties that cooperate, coordinated by an *adversary*, in order to break the remaining parties' privacy. Several MPC protocols have been proposed through time, leading to practical constructions seen in recent years and even some real-world applications and deployments (*e.g.* see https://mpc.cs.berkeley.edu/).

MPC protocols are typically characterized by the amount of corrupted parties t they can tolerate, with three notable settings being $t < n/3$, $t < n/2$ and $t < n$. Protocols with $t \geq n/3$ must allow for some negligible statistical error (if $t < n/2$), or even make use of cryptographic assumptions (if $t \geq n/2$). In the $t < n/3$ case several results are known: not only we can obtain perfectly secure MPC, which safeguards the parties' inputs regardless of the computational power of the adversary, but we can do so with several appealing features such as guaranteed output delivery

(G.O.D. , which ensures the successful completion of the protocol in spite of any adversarial misbehavior), and a communication complexity that requires each party to send in average an amount of messages that is independent of the total number of parties, and depends only on the circuit size of the function f (here we omit additive terms that are independent of the circuit size but only depend on the number of parties) [GLS19]. More precisely, we can represent the function f as an arithmetic circuit C comprised of input, output, addition and multiplication gates over some finite ring $\mathbb{Z}/q\mathbb{Z}$ of integers modulo an integer q, and if $|C|$ denotes the number of multiplication gates, it is possible to obtain perfectly secure MPC over $\mathbb{Z}/q\mathbb{Z}$ with G.O.D. where the total communication complexity is $O(n|C|)$ ring elements, which we refer to as *linear* communication complexity. A common choice is taking q to be a prime, but this claim holds for any *arbitrary* integer q: the *Chinese remainder theorem* reduces the general case to moduli of the form $q = p^k$, for a prime p and an integer k, and works such as [ACD+19] show that perfectly secure MPC with G.O.D. over these moduli is possible, with linear communication complexity. It is relevant to study the general modulus case as it contains several relevant cases such as $q = 2$, 2^{32} or 2^{64}, which have received considerable attention in recent works due to practical benefits [DEF+19].

Unfortunately, the claim of linear communication is not entirely accurate. Shamir secret-sharing is a core technique to enable these results, and for this scheme to work over $\mathbb{Z}/p^k\mathbb{Z}$, p must be strictly larger than n. Once n becomes larger than p, a so-called *Galois ring extension* of degree $\Omega(\log n)$ must be used. It is still true that the communication is $O(n|C|)$ ring elements, but this time each ring element is $\Omega(\log n)$ bits long. Hence, asymptotic communication is truly $O(n \log n|C|)$. In [CCXY18], the authors partially address this issue relying on the technique of reverse multiplication-friendly embeddings (RMFEs), which allow them to remove the $\log n$ overhead when computing many copies of the same circuits, i.e., a SIMD circuit. However, for general circuits, directly applying their techniques does not work as we will discuss in Section 1.3. This leads to the following question:

Can we design perfectly secure MPC protocols for general circuits over a constant-size ring $\mathbb{Z}/q\mathbb{Z}$ whose asymptotic communication complexity scales as $O(n|C|)$?

Again, all current solutions have communication that scales as $O(n|C|)$ ring elements, but assuming that either the ring bit-size scales as $\log n$ or the underlying circuit is a SIMD circuit. For fixed-size rings (which is the natural setting in practice) and general circuits, ring extensions must be used, which leads to a communication of $O(n \log n|C|)$. Recall that this affects the very practically-motivated settings of binary computation (*i.e.* circuits over $\mathbb{Z}_2$), and also the `int32` and `int64` cases (*i.e.* circuits over $\mathbb{Z}/2^k\mathbb{Z}$ for $k = 32$ and $k = 64$).

1.1 Our Contribution

In this work we give an answer in the affirmative to the aforementioned question. We provide an MPC protocol with the following desirable features:

- Perfect security against an active adversary corrupting $t < n/3$ parties
- Securely computes circuits over $\mathbb{Z}/q\mathbb{Z}$ for *any constant* q
- The total number of elements in $\mathbb{Z}/q\mathbb{Z}$ communicated scales as $O(n|C|)$
- Guaranteed output delivery

As we have previously discussed, via CRT this task reduces to the task of computation over a ring $\mathbb{Z}/p^k\mathbb{Z}$ with constant p. For the sake of presentation we focus on $p = 2$, that is, computation over $\mathbb{Z}/2^k\mathbb{Z}$, but the ideas presented directly work for more general p.

Remark 1 (On security with abort). We note that even removing the G.O.D. condition, and settling with security with abort only, obtaining a protocol with the other properties is not simple, and is on its own a relevant and challenging open question. In this work we aim for the stronger notion of G.O.D.

Remark 2 (Cost of addition gates). In our work, (a linear amount of) communication is required for every *addition gate*, which is not the case if one settles for $O(n \log n|C|)$ complexity (where addition gates are for free in terms of communication). Looking ahead, this occurs due to a technique called *network routing*, originated in [GPS22,GPS21], which is used to exploit some notion of

packing while routing values through the circuit correctly. The work of [GPS22] (which is set in a different context than ours) also suffers from communication per addition gates, and avoiding this overhead in our work would likely lead to improvements to [GPS22]. If the circuit does not have substantially many more addition gates than multiplication gates (a reasonable setting in practice), then this condition becomes immaterial for our asymptotic $O(n|C|)$ claim.

1.2 Related Work

Perfectly secure MPC for $t < n/3$ with linear communication and G.O.D. has been studied in multiple works such as the one by Goyal, Liu, and Song [GLS19] and Beerliová-Trubíniová and Hirt [BTH08]. These are set specifically in the context of finite fields $\mathbb{F}_{p^d}$, where $p^d = \Omega(n)$. The work ofAbspoel et al. [ACD+19] generalizes this to the ring $\mathbb{Z}/p^k\mathbb{Z}$ by using Galois ring extensions of degree $d = \Omega(n)$. Unfortunately, as we have pointed out, these techniques are not suitable for *constant-sized* rings since they add an $\Omega(\log n)$ overhead.

If it is known that t is *far* from $n/3$, that is, $t < n(\frac{1}{3} - \epsilon)$ for any constant $\epsilon > 0$, then it is possible to obtain perfectly fully secure MPC with a communication of $O(|C|)$ (independent of n) for fields $\mathbb{F}_{p^d}$ with $p^d = \Omega(n)$, and $O(\log n|C|)$ for constant-sized fields [DIK10]. These techniques extend naturally to the case of $\mathbb{Z}/p^k\mathbb{Z}$ by using ring extensions as in [ACD+19], and to arbitrary $\mathbb{Z}/q\mathbb{Z}$ via CRT. This is better than our linear communication $O(n|C|)$, but it assumes the aforementioned gap $\epsilon > 0$. Our protocol works for the case $\epsilon = 0$: $t < n/3$, and n can be as small as $n = 3t + 1$.

In the $t < n/2$ case with *statistical* security, a similar situation exists: most protocols with G.O.D. that achieve linear communication complexity require a ring $\mathbb{Z}/p^k\mathbb{Z}$ with $p > n$ [BFO12], [GSZ20,ACD+19]. This state of affairs changed for the case of $\mathbb{F}_p$ and *security with abort* for *constant* p in the recent work of [PS21]. The same ideas in [PS21] extend naturally to $\mathbb{Z}/p^k\mathbb{Z}$ for constant p. However, it is not clear how to extend the ideas in [PS21] to G.O.D. since, in the $t < n/2$ regime, the techniques used to achieve full security are considerably much more complex than these for $t < n/3$ (which are already quite intricate). The core difference is that in $t < n/2$ setting the central idea to achieve G.O.D. , dispute control, differs from player elimination—in spite of sharing some similarities: unlike player elimination, in $t < n/2$ this pair cannot be removed because one may be inadvertently removing one of the $t + 1$ honest parties, and t honest parties alone cannot have enough joint information to finish the computation.

Finally, we note that if one only wants to achieve statistical security (rather than perfect security) in the G.O.D. setting for $t < n/3$, [IKP+16] has achieved $O(\text{poly}(\log(n))|C|)$ elements of communication if $t < n(1/2 - \epsilon)$, for any constant $\epsilon > 0$, which is even better than linear communication.

1.3 Overview of our Techniques

We begin by highlighting the difficulties that existing works face when considering MPC over constant-size rings. Perfectly secure MPC with linear communication (for non-constant fields) was first proposed in [BTH08], and it was later improved in [GLS19] to remove a term dependent on the circuit depth. These ideas can be generalized to rings such as $\mathbb{Z}/p^k\mathbb{Z}$ [ACD+19], again with the same complexity if $p = \Omega(n)$. This complexity comes from the use of Shamir secret-sharing, which requires enough interpolation points to operate, and hence it requires a ring of large enough characteristic. Shamir's is not the only linear secret-sharing scheme one could use, but it is unlikely there exist other schemes that somehow reduce this secret/share size requirement, since this is highly connected to the MDS conjecture (see for example [FR22, Lemma 1]). Since Shamir secret-sharing seems unavoidable, our core idea is to use the ring extensions needed for it, while *packing* multiple entries in a single ring extension secret-shared value, so that the cost per single share is ultimately linear. Ultimately, the challenges lie on efficiently making use of this packing, which is our focus in this overview.

As we have mentioned previously, here and for the rest of our paper we focus on the case $\mathbb{Z}/2^k\mathbb{Z}$. Furthermore, for the sake of this introduction only we focus on security with abort (which, as mentioned previously, is already a challenging task on its own). We highlight towards the end of the section how to tackle the G.O.D. case. Our work makes use of several techniques in the literature such as player elimination [BTH08], reverse multiplication-friendly embeddings (RMFEs) [CCXY18], network routing [GPS22,GPS21], among others. In this section we provide a

high level overview of how these ideas are put together in order to obtain our protocol. First, we introduce some notation, with further details given in Section 2 and beyond. We let $\mathsf{GR}(2^k, m)$ be a Galois ring extension of $\mathbb{Z}/2^k\mathbb{Z}$ of degree m, which can be seen as the ring of polynomials over $\mathbb{Z}/2^k\mathbb{Z}$ modulo an monic polynomial irreducible mod 2 of degree m (for more details see Section 2 or [Wan03]). We can perform Shamir secret-sharing over this ring if $m = \lceil \log n \rceil + 1$ [ACD+19], and we denote degree-d sharings of $x \in \mathsf{GR}(2^k, m)$ by $[x]_d$. An RMFE is a pair of $\mathbb{Z}/2^k\mathbb{Z}$-linear homomorphisms $(\phi : (\mathbb{Z}/2^k\mathbb{Z})^\ell \to \mathsf{GR}(2^k, m),\ \psi : \mathsf{GR}(2^k, m) \to (\mathbb{Z}/2^k\mathbb{Z})^\ell)$ satisfying $\psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})) = \boldsymbol{x} \star \boldsymbol{y}$ for every $\boldsymbol{x}, \boldsymbol{y} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. Such objects exist, with $\ell = \Theta(m)$ [CCXY18,ELXY23], and in fact we can also take them such that $\phi(\mathbf{1}) = 1$ [ELXY23]. This ensures that $\psi(\phi(\boldsymbol{x})) = \psi(\phi(\boldsymbol{x})\phi(\mathbf{1})) = \boldsymbol{x} \star \mathbf{1} = \boldsymbol{x}$.

Embedding using RMFEs. We may take as a starting point the approach in [CCXY18] in order to secret-share elements of $\mathbb{Z}/2^k\mathbb{Z}$ efficiently using Shamir secret-sharing via RMFEs. Recall that one (naive) way of secret-sharing an element $x \in \mathbb{Z}/2^k\mathbb{Z}$ using Shamir SS is to embed it in $\mathsf{GR}(2^k, m)$, and then secret-sharing over $\mathsf{GR}(2^k, m)$, which results in an undesired overhead of $m \approx \log n$. Instead, one may secret-share m elements simultaneously by interpreting them as an element of $\mathsf{GR}(2^k, m)$, removing the overhead of m in an amortized sense (*i.e.* after dividing by the number of secrets m), but unfortunately this approach does not interact well with the MPC setting since one cannot easily multiply elements this way.

RMFEs are introduced in [CCXY18] as a solution to this problem. Instead of secret-sharing m elements by thinking of them as an element of $\mathsf{GR}(2^k, m)$, ℓ elements $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ are shared together by first mapping them as $x = \phi(\boldsymbol{x}) \in \mathsf{GR}(2^k, m)$, and sharing $[x]_t$ instead. This has an overhead of m for ℓ secrets, which is $m/\ell = \Theta(1)$ amortized. Furthermore, the multiplicative property of RMFEs turns out to enable products on secret-shared data, making this embedding particularly suitable for MPC. In fact, this is used in [CCXY18] to remove the $\log n$ overhead of perfect security in the context in which the circuit C is structured as ℓ copies of the same function, run on possibly different inputs. Our goal however is to enable a more general class of circuits that have no specific structure in terms of their wiring, and hence this approach is insufficient.

To illustrate the main challenge, let us discuss how the approach from [CCXY18] works, for ℓ copies of the same circuit. The invariant the parties maintain is that, for every set of ℓ values $\boldsymbol{x} = (x_1, \ldots, x_\ell) \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ corresponding to the same wire across the ℓ copies, the parties have sharings $[\phi(\boldsymbol{x})]_t$. Given ℓ copies of a multiplication gate with inputs $\boldsymbol{x}$ and $\boldsymbol{y}$, the invariant is maintained by using a triple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{c} = \boldsymbol{a} \star \boldsymbol{b}$. Assuming the parties have $[\phi(\boldsymbol{x})]_t$ and $[\phi(\boldsymbol{y})]_t$, they can locally compute $[\phi(\boldsymbol{x}) + \phi(\boldsymbol{a})]_t$ and open this as $\phi(\boldsymbol{u})$, and similarly open $\phi(\boldsymbol{v})$ where $\boldsymbol{v} = \boldsymbol{y} + \boldsymbol{b}$. Crucially, since this is Shamir reconstruction, this can be done with error correction/detection, ensuring no errors are introduced. Now, the parties compute locally

$$[z]_t = \phi(\boldsymbol{u}) \cdot \phi(\boldsymbol{v}) - \phi(\boldsymbol{v}) \cdot [\phi(\boldsymbol{a})]_t - \phi(\boldsymbol{u}) \cdot [\phi(\boldsymbol{b})]_t + [\phi(\boldsymbol{c})]_t,$$

which satisfies $\psi(z) = \boldsymbol{x} \star \boldsymbol{y}$, thanks to the properties of RMFEs. Finally, the parties can execute a simple re-encoding protocol that applies $\phi \circ \psi : \mathsf{GR}(2^k, m) \to \mathsf{GR}(2^k, m)$ to $[z]_t$, yielding $[\phi(\boldsymbol{x} \star \boldsymbol{y})]_t$, hence preserving the desired invariant.

Now, if the circuit is non-SIMD, it could easily happen that, say, a value encoded in position 1 must be multiplied (or even added) with a value encoded in position 2. The properties of RMFEs only allow for the computation of $\boldsymbol{x} \star \boldsymbol{y} = (x_1 y_1, \ldots, x_\ell y_\ell)$, but they fall short if one somehow needs products that are "not aligned", such as $(x_1 y_2, x_2 y_1, x_3 y_3, \ldots)$. SIMD circuits do not present such misalignments, which is why the techniques in [CCXY18] work in that setting.

Network routing. To alleviate this issue, we may resort to network routing, a general technique introduced in [GPS22,GPS21] to ensure the sharings of output groups can be rearranged in such a way that they are "aligned" when fed as inputs to future groups of gates. In these works, such "wiring" issues appeared in the context of packed secret-sharing where, as in our setting, parties have shares whose underlying secrets are *vectors* that can somehow be added or multiplied component-wise, but cannot be re-routed easily. In our work we show that such techniques can also be used in our context, where the "packing" is done with RMFEs. For the sake of keeping this overview as lightweight as possible, we will not dive into the details on how this is done, and instead we will refer the reader to Sections 3.4 and E where details are provided. For now, it suffices

to say that there is an efficient method for the parties to obtain sharings $[\phi(\boldsymbol{x})]_t$ for every input group in a given layer, starting from packed sharings of every output group of the previous layers.

For simplicity in the exposition we will assume from now on that the circuit only has multiplication gates, without any addition operation. We discuss at the end of this overview why this is convenient, and how to handle the case of addition gates as well. Naturally, our main protocols include the general case.

On the dependency on circuit width. Using the network routing techniques mentioned above, together with the protocol sketched earlier, we would obtain the desired result: perfectly secure MPC for $t < n/3$ over $\mathbb{Z}/2^k\mathbb{Z}$ with G.O.D. and linear communication $O(n)$. However, this result hides an assumption on the circuit structure: it requires each layer to contain at least $\Omega(n\ell)$ multiplication gates. This is because every layer requires parallel reconstructions for each of its multiplication gates, and robust reconstruction of Shamir sharings with *linear* communication (*i.e.* instead of all parties sending their shares to each other, which would be quadratic) requires $\Omega(n)$ reconstructions to be done in parallel [DN07]. On top of this each such Shamir sharing "packs" $\ell \approx O(\log n)$ values, which results in a requirement of $n\ell$ secrets to be reconstructed (per layer!) to obtain the communication gains.

The above results in a total communication that is not $O(n|C|)$, but rather $O(n|C| + n^2\ell \cdot \mathsf{depth}(C))$. For circuits C such that $\mathsf{depth}(C) \gg |C|/n$ (*e.g.* "skinny" circuits), this extra term dominates communication. In contrast, for the case of MPC over $\mathbb{F}_p$ for large p, it is known that the $n^2\ell \cdot \mathsf{depth}(C)$ term can be eliminated, resulting in true linear communication $O(n|C|)$ [GLS19]. This leaves a gap between what we know over (large) fields (which can be generalized for $\mathbb{Z}/p^k\mathbb{Z}$ for large p), and the case of $\mathbb{Z}/2^k\mathbb{Z}$. What follows is dedicated to discuss how we address this complexity gap.

Computing the circuit optimistically with additive secret-sharing. We first reduce the term $n^2\ell \cdot \mathsf{depth}(C)$ to $n\ell \cdot \mathsf{depth}(C)$ via the following core idea—also used in [GLS19]: derive *additive* sharings from Shamir sharings and use the additive sharings to compute the circuit (which enables cheating but can be done with linear communication without any minimum batch size requirement), and only use the Shamir sharings for a final verification check (which involves reconstructing many robust sharings in parallel and hence can be done with linear communication complexity). For the optimistic computation of the circuit we will make use of additive secret-sharing, which we denote by $\langle x \rangle$ for $x \in \mathbb{Z}/2^k\mathbb{Z}$ (we also extend this notation naturally to vectors over $\mathbb{Z}/2^k\mathbb{Z}$). For input gates, each client having a group of inputs $\boldsymbol{x} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ secret-shares $[\phi(\boldsymbol{x})]_t$ towards the parties (the parties perform degree checks that ensure the degree is indeed $\leq t$), and then the parties locally derive $\langle \boldsymbol{x} \rangle$ from $[\phi(\boldsymbol{x})]_t$. This is done by first locally converting $[\phi(\boldsymbol{x})]_t$ to $\langle \phi(\boldsymbol{x}) \rangle$ (a standard procedure involving each party multiplying locally by certain Lagrange coefficients), followed by a *local* application of the mapping ψ, to obtain $\langle \psi(\phi(\boldsymbol{x})) = \boldsymbol{x} \rangle$. Now, the parties optimistically compute the circuit. For every pair of values to be multiplied $\langle x \rangle$ and $\langle y \rangle$, letting $i \in [\ell]$ be the index of this gate in its group, the parties can use the triple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ associated to this group to locally derive the additively shared triple $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$, which they can use to multiply $\langle x \rangle$ and $\langle y \rangle$: open $u_i = x + a_i$ and $v_i = y + b_i$ (with linear communication without any minimum batch size restriction), and compute $\langle xy \rangle = u_i \cdot v_i - v_i \cdot \langle a \rangle - u_i \cdot \langle b \rangle + \langle c \rangle$.

For the check, recall that the parties have Shamir sharings of $[\phi(\boldsymbol{x})]_t$ for every group $\boldsymbol{x}$ in the input layer. Consider a multiplication group in the first layer having as inputs $\boldsymbol{x}, \boldsymbol{y}$. Using network routing, the parties can obtain Shamir sharings $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$, so they can compute $[\phi(\boldsymbol{u})]_t = [\phi(\boldsymbol{x})]_t + [\phi(\boldsymbol{a})]_t$ and $[\phi(\boldsymbol{v})]_t = [\phi(\boldsymbol{y})]_t + [\phi(\boldsymbol{b})]_t$, hence obtaining robust versions of the reconstructed values $\boldsymbol{u}, \boldsymbol{v}$ when these gates were computed optimistically. Furthermore, the parties can compute locally $[z]_t = \phi(\boldsymbol{u}) \cdot \phi(\boldsymbol{v}) - \phi(\boldsymbol{v}) \cdot [\phi(\boldsymbol{a})]_t - \phi(\boldsymbol{u}) \cdot [\phi(\boldsymbol{b})]_t + [\phi(\boldsymbol{c})]_t$, and apply the re-encoding protocol from [CCXY18] to obtain $[\phi(\boldsymbol{x} \star \boldsymbol{y})]_t$. Doing this for all groups in the first layer ensures that the parties obtain robust sharings $[\phi(\boldsymbol{z})]_t$ of all output groups $\boldsymbol{z}$ in the first layer. This process can be iterated: the parties use network routing to obtain robust sharings of all input groups in the second layer, obtain robust sharings of all reconstructed values $\boldsymbol{u}$ and $\boldsymbol{v}$ during the optimistic computation of this layer, and then compute robust sharings of the outputs groups in this layer. This is repeated until the output layer is reached. Before reconstructing the outputs, however, the parties robustly reconstruct in parallel all Shamir sharings $[\boldsymbol{u}]_t$ and $[\boldsymbol{v}]_t$ corresponding

to *all* multiplication groups, cross checking with the values reconstructed optimistically.[5] The key point is that this robust reconstruction involves enough values as to be able to enjoy amortized linear communication complexity.

Packing across different multiplication layers. Using additive SS optimistically during the circuit computation shaves a factor of n in the term $n^2\ell \cdot \mathsf{depth}(C)$, but it still leaves us with a term $n\ell \cdot \mathsf{depth}(C)$, since we still need to RMFE-pack ℓ gates across the *same* layer. To address this we allow groups to contain gates spanning over potentially *different* layers, with the only restriction being that the set of groups must form a DAG (we say there is an edge from group A to group B if at least one gate in A connects to one gate in B; in Section 3.1 we show that such grouping can always be done for *any* circuit—without any width restrictions). For notational convenience we assume that the gates in every group are indexed in increasing topological order, meaning that if a gate indexed by i in a group depends on the output of another gate indexed by j in the same group, then $j < i$. As before, we still group each client's input wires into batches of size ℓ each,[6] and we also let each client secret-shares a group of inputs $\boldsymbol{x}$ as $[\phi(\boldsymbol{x})]_t$.

Careful observation reveals that such relaxation in grouping does not really affect the optimistic computation using additive SS: the parties can still derive additive SS triples $(\langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle)$ and use these to perform multiplications. However, complications will arise at the verification stage. To illustrate this, let us consider the first multiplication group in topological order, and let us denote its inputs as $\boldsymbol{x}$ and $\boldsymbol{y}$. If it was the case that all of the multiplications in the group belonged to the first layer, then the parties could perform network routing as before to obtain $[\phi(\boldsymbol{x})]_t$ and $[\phi(\boldsymbol{y})]_t$. Unfortunately, since we are packing across different layers, it can happen that, say, the left input x_i to the i-th gate is equal to the output $x_j y_j$ of the j-th gate in the same group, with $j < i$. This prevents us from determining $[\phi(\boldsymbol{x})]_t$ from the input layer alone.

Determining $[\phi(\boldsymbol{x})]_t$ for the first multiplication group. For illustration, let us keep our focus on the first multiplication group. Our goal is to show how the parties can obtain robust sharings $[\phi(\boldsymbol{x})]_t$, which, as illustrated above, cannot be derived from the input layer alone since the left input x_i to get i depends on the j-th output $x_j y_j$ of gate $j < i$. First, recall that the parties have opened $\langle \boldsymbol{u}\rangle = \langle \boldsymbol{x}\rangle + \langle \boldsymbol{a}\rangle$ and $\langle \boldsymbol{v}\rangle = \langle \boldsymbol{y}\rangle + \langle \boldsymbol{b}\rangle$, in the process of processing this group optimistically. However, due to the non-robustness of additive SS, these reconstructions may have resulted in $\boldsymbol{u}' = \boldsymbol{u} + \boldsymbol{\delta}$ and $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{\epsilon}$, for some possibly non-zero $\boldsymbol{\delta}$ and $\boldsymbol{\epsilon}$. Using these reconstructions, the parties can compute locally $[\phi(\boldsymbol{x}+\boldsymbol{\delta})]_t = \phi(\boldsymbol{u}') - [\phi(\boldsymbol{a})]_t$ and $[\phi(\boldsymbol{y}+\boldsymbol{\epsilon})]_t = \phi(\boldsymbol{v}') - [\phi(\boldsymbol{b})]_t$, which correspond to robust sharings of the inputs $\boldsymbol{x}$ and $\boldsymbol{y}$, but *potentially incorrect*. Now we let the parties execute *any* correct multiplication protocol (we borrow the protocol from [BTH08,ACD+19] for this purpose) to compute the product $[\phi(\boldsymbol{x}+\boldsymbol{\delta}) \cdot \phi(\boldsymbol{y}+\boldsymbol{\epsilon})]_t$, followed by re-encoding as in [CCXY18] (*i.e.* applying $\phi \circ \psi$) to obtain $[\phi((\boldsymbol{x}+\boldsymbol{\delta}) \star (\boldsymbol{y}+\boldsymbol{\epsilon}))]_t$.

For simplicity let us assume that i is the only index in this group whose corresponding gate depends on other outputs from the same group, with all the other gates receiving inputs directly from the input layer. In particular, both x_j and y_j come from the input layer. Recall that the goal is to obtain $[\phi(\boldsymbol{x})]_t$ using network routing. Since the parties have robust sharings of all groups in the input layer, they have almost all the pieces needed to obtain $[\phi(\boldsymbol{x})]_t$, with the only missing part being robust sharings that contain the output of the j-th gate, since this is needed for the i-th left input x_i. Our idea is to use, for the missing j-th output, the sharings $[\phi((\boldsymbol{x}+\boldsymbol{\delta}) \star (\boldsymbol{y}+\boldsymbol{\epsilon}))]_t$ computed above, which contain the j-th output $x_j y_j + \gamma$, where $\gamma = x_j\epsilon_j + y_j\delta_j + \delta_j\epsilon_j$, in the j-th entry. In other words, the parties perform network routing on the sharings from the input layer *and* the sharing $[\phi((\boldsymbol{x}+\boldsymbol{\delta}) \star (\boldsymbol{y}+\boldsymbol{\epsilon}))]_t$ to obtain $[\phi(\boldsymbol{x})]_t$. However, since the i-th entry corresponds to $x_i + \gamma$, the actual secret is $[\phi(\boldsymbol{x} + \gamma\boldsymbol{e}_i)]_t$. In other words, the parties do not obtain robust sharings of the *correct* $\boldsymbol{x}$, but instead, the i-th entry is shifted by γ.

This may raise a red flag at first sight: recall we are using the sharing $[\phi(\boldsymbol{x}+\gamma\boldsymbol{e}_i)]_t$ to verify the multiplications in the first group, in particular, verifying that $\langle \boldsymbol{u}\rangle = \langle \boldsymbol{x}\rangle + \langle \boldsymbol{a}\rangle$ was opened correctly. However, we are using an incorrect $[\phi(\boldsymbol{x}+\gamma\boldsymbol{e}_i)]_t$, so it may be the case that this somehow helps the

[5] As shown in [GLS19], certain care is needed to ensure that security is not broken by delaying verification. We omit these details in this overview.

[6] We assume each client has $\Omega(\ell) = \Omega(n)$ inputs. Otherwise there is a minor overhead due to packing, but this is only restricted to the input layer.

adversary reconstruct $\langle \boldsymbol{u} \rangle$ incorrectly. For example, the adversary may be able to reconstruct $\langle \boldsymbol{u} \rangle$ as $\boldsymbol{u} + \gamma \boldsymbol{e}_i$, which is consistent with the robust sharings held by the parties $[\phi(\boldsymbol{x} + \gamma \boldsymbol{e}_i) + \phi(\boldsymbol{a})]_t$, and hence the check will pass, in spite of $\langle \boldsymbol{u} \rangle$ being reconstructed incorrectly. Yet, observe that this is an attack only if γ is non-zero, for which it must be the case that either δ_j or ϵ_j is not zero; say for simplicity $\delta_j \neq 0$. Fortunately, this will be caught in the check: the parties have the sharings $[\phi(\boldsymbol{x} + \gamma \boldsymbol{e}_i) + \phi(\boldsymbol{a})]_t$, which are incorrect in position i but, crucially, are correct in position j, so the adversary will not be able to conceal the fact that the j-th entry was modified.

Determining $[\phi(\boldsymbol{x})]_t$ for every input group. The principle above applies more generally. After performing optimistic computation using additive secret-sharing, the parties perform the following for every multiplication group with inputs $\boldsymbol{x}, \boldsymbol{y}$. Let $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ be the Shamir triple associated to the group, and recall that the parties reconstructed (potentially incorrectly) $\boldsymbol{u}$ and $\boldsymbol{v}$ as part of the optimistic multiplications. The parties compute locally $[\phi(\boldsymbol{x})]_t = \phi(\boldsymbol{u}) - [\phi(\boldsymbol{a})]_t$ and $[\phi(\boldsymbol{y})]_t = \phi(\boldsymbol{v}) - [\phi(\boldsymbol{b})]_t$, and then they compute the product $[\phi(\boldsymbol{x} \star \boldsymbol{y})]_t$ using a secure multiplication protocol followed by re-encoding, as illustrated earlier. At this point, for every set of outputs $\boldsymbol{x}$ of a given group, the parties hold $[\phi(\boldsymbol{x})]_t$. Moreover, the following crucial property holds: if no cheating occurred up to (and including) the optimistic evaluation of the gate at index i, then $\boldsymbol{x}[i]$ holds the correct wire value.

Now, the parties apply network routing to map all the packed sharings of the output groups into packed sharings of input groups $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$ for every multiplication group with inputs $\boldsymbol{x}, \boldsymbol{y}$. Importantly, due to the property above, if no cheating has occurred prior to the computation of, say, the i-th gate in a group with inputs $\boldsymbol{x}, \boldsymbol{y}$, then we know that the sharings $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$ derived from the network routing satisfy that the i-th entries x_i and y_i are *correct*. This way, if cheating occurs for the first time in this gate, by reconstructing incorrect $u'_i = x_i + a_i + \delta_i$ and $v'_i = y_i + b_i + \epsilon_i$ (which may result in more errors in other entries of the vectors $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$, but only with indices $j > i$, not for $j \leq i$), the parties can use $[\phi(\boldsymbol{x})]_t, [\phi(\boldsymbol{y})]_t$ (together with the associated triple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$) to check the correctness of the reconstructed u'_i and v'_i.

Dealing with Addition Gates. Recall we assumed for simplicity that the circuit did not have any addition gates. We briefly comment how the general case is handled. First, both addition and multiplication gates are grouped in sets of ℓ gates each (where the gates within each group are of the same type). The optimistic computation phase remains the same: the parties handle addition gates by simply adding their (additive) shares together, locally. For the verification step, however, we need the parties to communicate for every group of addition gates. To see why this is the case, consider a group of addition gates with inputs $\langle \boldsymbol{x} \rangle, \langle \boldsymbol{y} \rangle$, and imagine that every output of these gates is later each fed to a multiplication gate. The parties can of course add locally $\langle \boldsymbol{x} \rangle, \langle \boldsymbol{y} \rangle$, but recall that for the network routing phase to work, we need the parties to have *packed Shamir* sharings of the outputs, like $[\phi(\boldsymbol{x} + \boldsymbol{y})]_t$, but it is not clear how they can obtain these from $\langle \boldsymbol{x} \rangle, \langle \boldsymbol{y} \rangle$ alone.

For the case of multiplication gates, this was achieved with the help of the (packed) triple that was used for the product. For the case of addition gates, we will use a similar idea: we make use of an *additive triple* $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$, ask the parties to open $\boldsymbol{u} \leftarrow \langle \boldsymbol{x} + \boldsymbol{a} \rangle$ and $\boldsymbol{v} \leftarrow \langle \boldsymbol{y} + \boldsymbol{b} \rangle$, and compute $[\phi(\boldsymbol{x} + \boldsymbol{y})]_t = (\boldsymbol{u} + \boldsymbol{v}) - [\phi(\boldsymbol{c})]_t$. This can also be seen as adding an extra step that first converts $\langle \boldsymbol{x} \rangle$ to $[\phi(\boldsymbol{x})]_t$ using the "double sharing" $(\langle \boldsymbol{a} \rangle, [\phi(\boldsymbol{a})]_t)$ (and similarly for $\boldsymbol{y}$), and then add these sharings together.

On Guaranteed Output Delivery. Finally, we comment on how the protocol sketched here is extended to G.O.D., without blowing communication. We use the player elimination framework from [BTH08], in which the parties not only perform a check but, in case of failure, identify a so-called *semi-corrupted pair* in the process, which is a pair of parties that is guaranteed to contain at least one corrupted party. At this point, the pair can be safely removed from the computation (which preserves the $t < n/3$ ratio), and the computation can be restarted. Restarting the computation many times can cause communication to blow up by a large factor. To address this, the circuit is split into *segments* of certain size, and the check described here is performed at the end of each such segment, rather than at the end of the whole circuit. Setting segment sizes appropriately reduces the size of the repeated computations, which keeps communication within $O(n|C|)$. There are

subtle issues, like part of the output of a group of a given segment being fed to the same segment, while some other part is fed to a future segment. This is mostly inconvenient notation-wise, but it does not add heavy technical complications.

To give a more complete picture, it remains to describe more clearly how the parties can identify a semi-corrupted pair during our check. Recall that our verification consists, in essence, of opening sharings of the form $[\phi(\boldsymbol{z})]_t$, and comparing against a previously opened set of values $\boldsymbol{z}'$. How should the parties react in case some mismatch is found? The core idea is to pinpoint to the party who announced an incorrect (additive) sharing in a first place. The main challenge with this is that, even though the parties have a robust version of the underlying *secrets* $\boldsymbol{z}$, they do not necessarily have a robust version of the additive sharings that each party should have sent when reconstructing $\boldsymbol{z}$, so it is not obvious how to identify which party sent an incorrect additive share. Fortunately, as it turns out, it is indeed possible to derive robust sharings that somehow commit the parties to the additive shares they should send at reconstruction. For this we introduce a notion of *extended* additive sharings, which expands additive SS with the necessary information to check whether a party sent a correct additive share. We provide details in Sections 3.2 and C in the Appendix.

Remark 3 (On sharings of zero). We remark that our overview here is a simplified version of our actual protocol, which must use of several other ingredients not discussed here. One of these is that, in several places, the parties need to re-randomize certain sharings using shares of zero, which is crucial for, among different purposes, preventing leakage of sensitive information when reconstructing optimistically, as in [GLS19].

1.4 Outline of the Document

We begin by presenting several important preliminaries in Section 2. This is followed by Section 3, which contains our protocols for optimistically evaluating a segment (Section 3.2), as well as verifying the computation is performed correctly (Section 3.5). This includes the network routing needed to compute robust sharings of groups in the circuit (Section 3.4), as well as our method to identify semi-corrupted parties once an attack has been detected (Section 3.6). We also discuss in detail how the circuit is partitioned into groups and segments (Section 3.1). Section 4 uses the building blocks from the previous sections to present our main MPC perfectly secure protocol with G.O.D. over constant-size rings, with linear communication complexity.

Our work makes use of several functionalities and protocols. To help the reader navigate, we provide in Section G in the Appendix a list with all of our functionalities and protocols, and their location within the text.

2 Preliminaries

In this work, we focus on functions that can be written as an arithmetic circuit C over the ring $\mathbb{Z}/2^k\mathbb{Z}$ with input, addition, multiplication and output gates. Let $|C|$ denote the size of the circuit C. We will make use of the *client-server* model for secure multiparty computation, in which clients can provide inputs and receive outputs to/from the servers, who are the parties who execute the actual MPC protocol. Note that, if every party plays a single client and a single server, this corresponds to a protocol in the standard MPC model. We assume that every pair of parties, either client and/or server, is connected via a secure (private and authentic) synchronous channel. We measure communication complexity as the total number of bits sent via private channels.[7]

Let c denote the number of clients, n denote the number of servers, and t denote the upper bound of the number of corrupted servers. In this work we focus on the 1/3-corruption setting, *i.e.* $3t + 1 = n$. In this work, we design an MPC protocol where all clients and servers compute the functionality $\mathcal{F}_{\mathsf{Main}}$ with perfect security. Our definition of perfect security is based on the standard simulation-based security which is shown in the work [Can00].

[7] Since we consider constant-sized rings, this is asymptotically the same as measuring the number of ring elements.

Functionality 1: $\mathcal{F}_{\mathsf{Main}}(C)$

1. Let x denote the input and C denote the circuit. $\mathcal{F}_{\mathsf{Main}}$ receives the input from all clients.
2. $\mathcal{F}_{\mathsf{Main}}$ computes $C(x)$ and distributes the output to all clients.

2.1 Party-Elimination Framework

We make use of the party elimination framework byHirt, Maurer, and Przydatek [HMP00], which constitutes a general strategy to achieve perfect security with G.O.D. with linear communication complexity. The basic idea is to let the parties perform checks that evaluate the correctness of the computation, identifying a pair of parties (with the help of BA for consensus) that contains at least one corrupted party in case of failure; such pair is referred to as a *semi-corrupted pair*. This pair of parties is then *eliminated* (*i.e.* removed from the computation), and the protocol is restarted. To avoid the overhead of re-executing as many times as potential eliminated pairs—which is upper bounded by t—the computation is divided into segments, and the check is performed at the end of each segment. This way, the extra cost of re-running is—in the worst case—t times the cost of each segment, so by keeping segments of appropriate size one can obtain efficient protocols with G.O.D.

We use $\mathcal{P}_{\mathtt{active}}$ to denote the set of parties which are active in the current segment, that is, that have not been eliminated. We use $\mathcal{C}_{\mathtt{active}} \subset \mathcal{P}_{\mathtt{active}}$ for the set of active corrupted parties. Let n' be the size of $\mathcal{P}_{\mathtt{active}}$. We use t' for the maximum possible number of the corrupted parties in $\mathcal{P}_{\mathtt{active}}$. Each time a semi-corrupted pair is identified, these two parties are removed from $\mathcal{P}_{\mathtt{active}}$ and hence $\mathcal{C}_{\mathtt{active}}$. It results in $n' := n' - 2$ and $t' := t' - 1$. Initially we have $n = n'$, $t = t'$. Let $T = n' - 2t'$. Therefore, T remains unchanged during the whole protocol.

2.2 Finite Rings

Basic notation. Let $\mathbb{Z}$ denote the ring of integers. For $q \in \mathbb{Z}$, let $q\mathbb{Z}$ denote the ideal $\{q \cdot n : q \in \mathbb{Z}\}$ and let $\mathbb{Z}/q\mathbb{Z}$ denote the quotient ring, which is the ring of integers modulo q. For a ring $\mathcal{S}$, let $\mathcal{S}[X]$ denote the ring of polynomials in the variable X with coefficients in $\mathcal{S}$. Also, let $\mathcal{S}^*$ denote the multiplicative subgroup of invertible elements in $\mathcal{S}$.

Galois Rings. We adopt the notion of *Galois rings* that contains the quotient ring $\mathbb{Z}/2^k\mathbb{Z}$ from [ACD+19].

Definition 1 (Galois Ring [ACD+19]). *A degree-d* Galois ring *of $\mathbb{Z}/2^k\mathbb{Z}$ is a ring of the form*

$$(\mathbb{Z}/2^k\mathbb{Z})[X]/g(X),$$

where h is a positive integer, and $g(X) \in (\mathbb{Z}/2^k\mathbb{Z})[X]$ is a non-constant degree-d polynomial such that its reduction modulo 2 is an irreducible polynomial in the field $\mathbb{F}_2[X]$. We use $\mathsf{GR}(2^k, d)$ to denote degree-d Galois ring *of $\mathbb{Z}/2^k\mathbb{Z}$.*

In order to interpolate polynomials in a Galois ring, we rely on the following lemma.

Lemma 1 ([ACD+19]). *Let $\mathsf{GR}(2^k, d)$ be a Galois ring with degree d. There exists a length 2^d sequence of distinct elements in $\mathsf{GR}(2^k, d)$ denoted by $\alpha_1, \ldots, \alpha_{2^d}$, such that for any $x_1, \ldots, x_{2^d} \in \mathsf{GR}(2^k, d)$, there exists a unique interpolating polynomial of degree at most $(2^d - 1)$ such that $f(\alpha_i) = x_i$ for all $i \in \{1, 2, \ldots d\}$.*

Using this lemma, we can define necessary components such as Shamir secret sharings and hyper-invertible matrices over Galois rings. In the following, we will use a Galois ring of $\mathbb{Z}/2^k\mathbb{Z}$ denoted by $\mathcal{R} := \mathsf{GR}(2^k, m)$. Note that the size of $\mathcal{R}$ is $2^{m \cdot k}$. We select m such that $2^m \geq 2n + 1$ so that it is possible to interpolate degree-$2n$ polynomials in $\mathcal{R}$.

2.3 Secret Sharing Schemes

Shamir Secret Sharing. We will use the standard Shamir secret sharing scheme [Sha79] in this work. For the Galois ring $\mathcal{R} = \mathsf{GR}(2^k, m)$, suppose $(\alpha_i)_{i=1}^n, \beta$ are $n+1$ distinct points, which can be used to interpolate polynomials according to Lemma 1. A degree-d Shamir sharing of $x \in \mathcal{R}$ among $n' \leq n$ parties is a vector $(s_1, s_2, \ldots, s_{n'}) \in \mathcal{R}^{n'}$ that satisfies the property that there exists a polynomial $f(\cdot) \in \mathcal{R}^{\leq d}[x]$ with $f(\beta) = x$ and $f(\alpha_i) = s_i, \forall i \in [n']$. The share held by party P_i is s_i. With any $(d+1)$ different shares of the same sharing the secret x can be reconstructed.

A degree-d Shamir sharing of $x \in \mathcal{R}$ is denoted as $[x]_d$. The following two properties hold for Shamir sharings:

- For all $x, y \in \mathcal{R}$, $[x+y]_d = [x]_d + [y]_d$.
- For all $x, y \in \mathcal{R}$ and for all d_1, d_2 subject to $d_1 + d_2 < n$, we have $[x \cdot y]_{d_1+d_2} = [x]_{d_1} \cdot [y]_{d_2}$.

Our protocol also rely heavily on the following property of Shamir sharings. Suppose after some party elimination steps we have n' parties where a maximum t' of them can be malicious.

Lemma 2 ([BTH08]). *Suppose n' parties share a degree-d Shamir sharing $[x]_d$, and at most t' of the shares may be incorrect.*

- *If $t' < (n'-d)/2$, then $[x]_d$ is correctable after receiving all the shares, e.g. by Berlekamp-Welch Algorithm.*
- *If $t' < n' - d$, then whether $[x]_d$ is inconsistent is detectable after receiving all the shares. This is due to the fact that two different degree-d polynomials agree on at most d points.*

2.4 Reverse Multiplication Friendly Embeddings

Definition 2 (RMFE over Ring [CRX21,ELXY23]). *Let ℓ, m, k be positive integers. Let $\mathcal{R} = \mathsf{GR}(2^k, m)$ denote the degree-m Galois ring of $\mathbb{Z}/2^k\mathbb{Z}$. A pair of mappings $(\phi : (\mathbb{Z}/2^k\mathbb{Z})^\ell \to \mathcal{R}, \psi : \mathcal{R} \to (\mathbb{Z}/2^k\mathbb{Z})^\ell)$ is called an $(\ell, m)_{2^k}$-reverse multiplication friendly embedding (RMFE) if, for all $\boldsymbol{x}, \boldsymbol{y} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$, it holds that*

$$\psi(\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})) = \boldsymbol{x} \star \boldsymbol{y}.$$

Without loss of generality we can assume that $\psi(\phi(\mathbf{1})) = 1$, which ensures $\psi(\phi(\boldsymbol{x})) = \boldsymbol{x}$ for all $\boldsymbol{x} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$.

Defining the $\mathbb{Z}/2^k\mathbb{Z}$-Linear Map $\mathsf{val}(\cdot)$ [PS21]. To compute the summation of all entries of $\psi(y)$ from $y \in \mathcal{R}$, we define an $\mathbb{Z}/2^k\mathbb{Z}$-linear map $\mathsf{val}(\cdot) : \mathcal{R} \to \mathbb{Z}/2^k\mathbb{Z}$ as follows:

- For an input y, suppose $\psi(y) = (y_1, y_2, \ldots, y_\ell)$.
- $\mathsf{val}(y)$ is defined to be $\sum_{i=1}^\ell y_i$.

Let $\boldsymbol{e}_i$ be a vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ such that all entries are 0 except that the i-th entry is 1, and let $\boldsymbol{x}$ be a vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ of which the i-th entry is x_i. According to the definition of RMFEs, we have $\boldsymbol{e}_i \star \boldsymbol{x} = \psi(\phi(\boldsymbol{e}_i) \cdot \phi(\boldsymbol{x}))$. Therefore, we can access x_i by computing $x_i = \mathsf{val}(\phi(\boldsymbol{e}_i) \cdot \phi(\boldsymbol{x}))$.

Existence of Constant Rate RMFEs over ring $\mathbb{Z}/2^k\mathbb{Z}$ [ACE+21]. In [ACE+21] it has been shown that constant rate RMFEs exist, as summarized in Theorem 1.

Theorem 1. *There exists a family of constant rate $(\ell, m)_{2^k}$-RMFE where $m = \Theta(\ell)$.*

In this work, we will use $(\ell, m)_{2^k}$-RMFE such that $m = O(\log n)$ and $\ell = O(\log n)$. The Galois ring $\mathcal{R} = \mathsf{GR}(2^k, m)$ satisfies $2^m \geq 2n + 1$.

2.5 Useful Building Blocks

Reconstructing Shamir Sharings. The functionality $\mathcal{F}_{\mathsf{OpenPub}}$ takes N degree-d ($d \leq t$) Shamir secret sharings over $\mathcal{R}$ as input, and it outputs the reconstructed secrets to all parties. We assume that for each input degree-d sharing, the shares of all active honest parties lie on a degree-d polynomial.[8] The full description of $\mathcal{F}_{\mathsf{OpenPub}}$ appears in Section A in the Appendix. An instantiation of this functionality for our ring case can be easily generalized from the field-case construction in [ACD+19], which has communication complexity of $O(N \cdot n + n^2)$ elements in $\mathcal{R}$ (*i.e.* $O(N \cdot n \cdot m + n^2 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$).

Secure Multiplication. The functionality $\mathcal{F}_{\mathsf{Mult}}$ takes two tuples of N degree-t Shamir sharings over $\mathcal{R}$ as input, which are denoted by $([x_1]_t, \ldots, [x_N]_t)$ and $([y_1]_t, \ldots, [y_N]_t)$. The output of $\mathcal{F}_{\mathsf{Mult}}$ is the tuple of degree-t Shamir sharings of the results $([x_1 \cdot y_1]_t, \ldots, [x_N \cdot y_N]_t)$. We assume that for each input degree-t Shamir sharing, the shares of all active honest parties lie on a degree-t polynomial. The full description of $\mathcal{F}_{\mathsf{Mult}}$ appears in Section A in the Appendix. An instantiation of this functionality for our ring case can be easily generalized from the field-case construction in [BTH08]. Also, another instantiation of this functionality is implied in [ACD+19]. The protocol generalized from [BTH08] has communication complexity of $O(N \cdot n + n^3)$ elements in $\mathcal{R}$ (*i.e.* $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$).

Performing Re-Encode. In our construction, we will need to transform a degree-t Shamir sharing over $\mathcal{R}$ from $[x]_t$ to $[\phi \circ \psi(x)]_t$ in order to evaluate multiplication gates in the circuit. This process is called *re-encode*. The functionality $\mathcal{F}_{\mathsf{ReEncode}}$ takes N degree-t Shamir sharings over $\mathcal{R}$ as input, which are denoted by $[x_1]_t, \ldots, [x_N]_t$. The output of $\mathcal{F}_{\mathsf{ReEncode}}$ are N degree-t Shamir sharings of the re-encoded result $[\phi \circ \psi(x_1)]_t, \ldots, [\phi \circ \psi(x_N)]_t$. We assume that for each input degree-t Shamir sharing, the shares of all active honest parties lie on a degree-t polynomial. The full description of $\mathcal{F}_{\mathsf{ReEncode}}$ appears in Section A in the Appendix. An instantiation of this functionality for our ring case can be easily generalized from the field-case construction in [CCXY18]. The instantiation has communication complexity of $O(N \cdot n + n^3)$ elements in $\mathcal{R}$ (*i.e.* $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$).

Verifying Consistency of Unreliable Broadcast Values. In our protocol, we will ask a dealer D to distribute several values that are supposed to be all the same, towards all parties. We do this over point-to-point channel to save the communication. The functionality $\mathcal{F}_{\mathsf{VerifyBC}}$ receives from all parties N such unreliable broadcast values dealt by a dealer D, and verifies whether all parties indeed received the same values. The output of this functionality to each party is either `consistent` or $(\texttt{inconsistent}, E)$, where E is a semi-corrupted pair of parties. The full description of $\mathcal{F}_{\mathsf{VerifyBC}}$ appears in Section A in the Appendix. An instantiation of this functionality for our ring case can be easily generalized from the field-case construction in [BTH08], which has communication complexity of $O(N \cdot n + n^2)$ elements in $\mathcal{R}$ (*i.e.* $O(N \cdot n \cdot m + n^2 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$).

Input Gates for Shamir Sharings. We introduce the functionality $\mathcal{F}_{\mathsf{InputShamir}}$, where a client C with N inputs in $\mathcal{R}$ shares its inputs to the active parties using Shamir secret sharing. The full description of $\mathcal{F}_{\mathsf{InputShamir}}$ appears in Section A in the Appendix. An instantiation of this functionality for our ring case can be easily generalized from the field-case construction in [BTH08]. The instantiation has a communication complexity of $O(N \cdot n + n^3)$ elements in $\mathcal{R}$ (*i.e.* $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$).

2.6 Preparing Correlated Randomness

Our protocol relies on different forms of correlated randomness shared by all parties, and these are prepared independently of the inputs of the clients. We give a brief description of the correlations we require below.

[8] If this is not the case, we ask the functionality to send the active honest parties' inputs to the adversary and allow the adversary to decide the output of active honest parties. Essentially, we give up the security if the shares of active honest parties do not lie on degree-d polynomials.

Random Shamir Sharings. The functionality $\mathcal{F}_{\mathsf{RandShamir}}$ enables all parties to prepare N random degree-t Shamir sharings in the form of $[\phi(\boldsymbol{r})]_t$, where $\boldsymbol{r}$ is a random vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$. The description and the instantiation of $\mathcal{F}_{\mathsf{RandShamir}}$ can be found in Section B.3 in the Appendix. The total communication complexity for the instantiation of $\mathcal{F}_{\mathsf{RandShamir}}$ to generate N random Shamir sharings is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Random Zero Additive Sharings. Once Beaver triples are prepared in the preprocessing phase, parties only need to do reconstructions in the online phase. To protect the shares held by honest parties, for each reconstruction, we will prepare a random additive sharing of 0 among the first $t+1$ parties. The functionality $\mathcal{F}_{\mathsf{RandZeroAdditive}}$ enables all parties to prepare N random zero additive sharings. The description and the instantiation of $\mathcal{F}_{\mathsf{RandZeroAdd}}$ can be found in Section B.3 in the Appendix. The total communication complexity for the instantiation of $\mathcal{F}_{\mathsf{RandZeroAdd}}$ to generate N zero additive sharings is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Random Parity Sharings. For an element $p \in \mathcal{R}$, we say that p is *parity element* if $\mathsf{val}(p) = 0$, and a parity sharing is a degree-t Shamir sharing of a parity element. When localizing a fault within a circuit segment, uniformly random parity sharings will be used as masks so that it is possible to check the correctoness of the reconstruction. The functionality $\mathcal{F}_{\mathsf{RandParity}}$ enables all parties to prepare N random parity sharings. The description and instantiation of $\mathcal{F}_{\mathsf{RandParity}}$ can be found in Section B.3 in the Appendix. The total communication complexity for the instantiation of $\mathcal{F}_{\mathsf{RandParity}}$ to generate N random parity sharings is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Beaver Triples. To evaluate addition gates and multiplication gates, all parties will prepare Beaver triples in the form of $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, where $\boldsymbol{a}+\boldsymbol{b} = \boldsymbol{c}$ when the Beaver triple is additive and $\boldsymbol{a} \star \boldsymbol{b} = \boldsymbol{c}$ when the Beaver triple is multiplicative. We introduce two functionalities $\mathcal{F}_{\mathsf{TripleAdd}}$ and $\mathcal{F}_{\mathsf{TripleMult}}$. $\mathcal{F}_{\mathsf{TripleAdd}}$ enables all parties to prepare N random additive Beaver triples, and $\mathcal{F}_{\mathsf{TripleMult}}$ enables all parties to prepare N random multiplicative Beaver triples. The descriptions and the instantiations of both $\mathcal{F}_{\mathsf{TripleAdd}}$ and $\mathcal{F}_{\mathsf{TripleMult}}$ can be found in Section B.3 in the Appendix. The total communication complexity for the instantiation of either $\mathcal{F}_{\mathsf{TripleAdd}}$ or $\mathcal{F}_{\mathsf{TripleMult}}$ to generate N Beaver Triples is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

3 Segment Evaluation and Verification

Our protocol first splits the circuit into *segments*, and then assigns gates of each type within a segment into *gate groups*. This is discussed in Section 3.1. Then, in Section 3.2 we show how the parties evaluate the gates of a given circuit *optimistically*, that is, without checking that the computation was carried out correctly. The verification is discussed in Section 3.3

3.1 Groups and Segments

We need to split the circuit into groups in certain way in order to apply packing effectively. We assume that the circuit C satisfies the following conditions:

1. **Circuit Segment Conditions:**
   - Because C is a Directed Acyclic Graph (DAG), there exists a topological ordering among all addition and multiplication gates. We require that each segment consists of addition and multiplication gates whose topological orders are consecutive.
   - The size of each segment should be $\Theta(n^2 \cdot m^2)$, except that the size of the last segment can be $O(n^2 \cdot m^2)$.
2. **Gate Number Conditions:**
   - In the input and output layers, the number of input gates belonging to each client and the number of output gates belonging to each client are multiples of ℓ.
   - The number of addition and multiplication gates within each circuit segment are multiples of ℓ.
3. **Gate Grouping Conditions:**

– During the computation, gates that have the same type (i.e., input gates belonging to the same client, output gates belonging to the same client, multiplication gates in the same circuit segment, addition gates in the same circuit segment) are organized into gate groups of size ℓ.
– For the output wires of each gate group, the number of times that those wires are used as input wires in other gates is a multiple of ℓ.

In Section E.1 in the Appendix we show that, if C does not satisfy these properties, then it can be transformed into a circuit C' that does satisfy the properties without affecting our linear communication claim. Based on the conditions above, we can split each segment into *gate groups* consisting of either ℓ multiplication gates (in which case the group is a *multiplication group*), or ℓ addition gates (in which case the group is an *addition group*). A set of ℓ wires corresponding to the left or right inputs of a given gate group is referred to as an *input group*, and *output groups* are defined similarly, but with output wires.

3.2 Segment Evaluation

The focus of this section is to show how the parties can evaluate optimistically a given segment $\mathtt{seg}$. The overall idea is to use additive secret-sharing with multiplication triples derived from packed triples. However, we consider an "enhanced" version of additive secret-sharing that allows for fault detection in case of cheating. This is described below.

Extended additive sharings. Let (ϕ, ψ) be an $(\ell, m)_2$-RMFE. Recall that n denotes the number of parties and $\phi : (\mathbb{Z}/2^k\mathbb{Z})^\ell \to \mathcal{R}$ is an $\mathbb{Z}/2^k\mathbb{Z}$-linear map. Also, $\mathcal{R} = \mathsf{GR}(2^k, m)$ such that $2^m \geq 2n + 1$. Therefore, Shamir secret sharing is well-defined in $\mathcal{R}$. In our construction, we will use ϕ to encode a vector of secrets $\boldsymbol{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(\ell)}) \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. All parties will hold a degree-t Shamir sharing of $\phi(\boldsymbol{x})$, denoted by $[\phi(\boldsymbol{x})]_t$. For $x \in \mathbb{Z}/2^k\mathbb{Z}$, we use $\langle x \rangle$ to denote an *additive sharing* of x among the first $t + 1$ parties in $\mathbb{Z}/2^k\mathbb{Z}$. Specifically, the additive sharing of x is $\langle x \rangle = (x_1, \ldots, x_{n'})$ where party P_i holds the share $x_i \in \mathbb{Z}/2^k\mathbb{Z}$ such that $\sum_{i=1}^{t+1} x_i = x$ and $x_{t+2}, \ldots, x_{n'}$ are all 0. Recall that $\psi : \mathcal{R} \to (\mathbb{Z}/2^k\mathbb{Z})^\ell$ and $\mathsf{val}(\cdot) : \mathcal{R} \to \mathbb{Z}/2^k\mathbb{Z}$ are both $\mathbb{Z}/2^k\mathbb{Z}$-linear. The parties have *extended additive sharings* of $x \in \mathbb{Z}/2^k\mathbb{Z}$, denoted by $(\!|x|\!)$, if the they have a degree-t shamir sharing $[y]_t$ in $\mathcal{R}$ such that $\mathsf{val}(y) = x$. We write $(\!|x|\!) := [y]_t$. It is clear that these sharings are additive.

We note that we can derive extended additive sharings of $x \in \mathbb{Z}/2^k\mathbb{Z}$ from Shamir sharings $[\phi(\boldsymbol{z})]_t$, where the j-th element of $\boldsymbol{z}$ is x. To see this, observe that, by the property of RMFE, we have that $\psi(\phi(\boldsymbol{e}_j) \cdot \phi(\boldsymbol{z})) = \boldsymbol{e}_j \star \boldsymbol{z}$. Therefore, $\mathsf{val}(\phi(\boldsymbol{e}_j) \cdot \phi(\boldsymbol{z})) = x$. To obtain $(\!|x|\!)$, all parties locally compute $(\!|x|\!) = \phi(\boldsymbol{e}_j) \cdot [\boldsymbol{z}]_t$. In addition, it is easy to obtain sharings $\langle x \rangle$ from $(\!|x|\!)$ by using Lagrange coefficients; we give the details in Section C in the Appendix, where we describe at length the notion of extended sharings, together with their properties.

Optimistically evaluating a segment. For a circuit segment $\mathtt{seg}$, we use Protocol $\Pi_{\mathsf{Eval}}(\mathtt{seg})$ to optimistically evaluate this segment. We evaluate its addition and multiplication gates using extended additive sharings and Beaver triples. A Beaver triple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ can be used to evaluate ℓ addition gates or ℓ multiplication gates. We assume that all parties have computed Shamir sharings of all gate group outputs of the previous circuit segments. This means that for an i-th element of any $[\phi(\boldsymbol{z})]_t$ used as a gate input in $\mathtt{seg}$, all parties can locally compute the extended additive sharing $(\!|x|\!) := \phi(\boldsymbol{e}_i) \cdot [\phi(\boldsymbol{z})]_t$.

For each gate with extended additive input sharings $(\!|x|\!), (\!|y|\!)$, we will use the Beaver triple associated to the gate group containing this gate. Suppose the gate is the j-th gate within the gate group, and suppose $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$ is the Beaver triple corresponding to the gate group. All parties compute the extended additive sharings $(\!|a_j|\!) := \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{a})]_t$, $(\!|b_j|\!) := \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{b})]_t$ and $(\!|c_j|\!) := \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{c})]_t$. Then all parties derive the additive sharings $\langle x \rangle$, $\langle y \rangle$, $\langle a_j \rangle$, $\langle b_j \rangle$ from the extended additive sharings using the method described previously.

The next step is reconstructing $\langle x \rangle + \langle a_j \rangle$ and $\langle y \rangle + \langle b_j \rangle$, for which a fixed dealer D will receive all shares, and then sends the reconstructed value to all parties. However, a subtle issue is that all parties must protect the redundancy in their sharings by preparing two random zero additive

sharing $\langle o_1 \rangle, \langle o_2 \rangle$, which can be done with Functionality $\mathcal{F}_{\mathsf{RandZeroAdd}}$. Then, all parties send their shares of $\langle x \rangle + \langle a_j \rangle + \langle o_1 \rangle$ and $\langle y \rangle + \langle b_j \rangle + \langle o_2 \rangle$ to the dealer D, who reconstructs $u := x + a_j$ and $v := y + b_j$, and sends the result to all other parties.

If the gate is an addition gate, all parties locally compute the output extended additive sharing

$$(\!|z|\!) := (u + v) \cdot \phi(\boldsymbol{e}_j) - (\!|c_j|\!).$$

If the gate is a multiplication gate, all parties locally compute the output extended additive sharing

$$(\!|z|\!) := (u \cdot v) \cdot \phi(\boldsymbol{e}_j) - u \cdot (\!|b_j|\!) - v \cdot (\!|a_j|\!) + (\!|c_j|\!).$$

We describe Π_{Eval} in full detail below, and we show that the comminication cost of Π_{Eval} is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Protocol 1: $\Pi_{\mathsf{Eval}}(\mathtt{seg})$

1. Suppose $\mathtt{seg}$ has $n_{\text{add}} \cdot \ell$ addition gates and $n_{\text{mult}} \cdot \ell$ multiplication gates. All parties select the active party with the smallest index as the dealer of this segment $\mathtt{seg}$. Let D denote the dealer.
2. All parties call $\mathcal{F}_{\mathsf{RandZeroAdd}}$ to prepare $2 \cdot (n_{\text{add}} + n_{\text{mult}}) \cdot \ell$ random additive zero sharings. All parties also receive a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
3. All parties call $\mathcal{F}_{\mathsf{TripleAdd}}(n_{\text{add}} \cdot \ell)$ to generate the additive Beaver triples for the segments. All parties receive a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$.
   All parties call $\mathcal{F}_{\mathsf{TripleMult}}(n_{\text{mult}} \cdot \ell)$ to generate the multiplicative Beaver triples for the segments. All parties receive a set of eliminated parties denoted by S_3. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_3$.
4. All parties locally get all the extended additive sharings for the gate inputs of $\mathtt{seg}$ that are collected from previous layers. For the gate input connected to the i-th wire of the Shamir sharing $[\phi(\boldsymbol{z})]_t$, all parties locally derive the extended additive sharing by $\phi(\boldsymbol{e}_i) \cdot [\phi(\boldsymbol{z})]_t$.
5. All parties evaluate the multiplication gates and addition gates within $\mathtt{seg}$ according to topological ordering. For each gate with input extended additive sharings $(\!|x|\!)$ and $(\!|y|\!)$, we suppose it corresponds to the j-th entry of the Beaver triple $([\phi(\boldsymbol{a})]_t, [\phi(\boldsymbol{b})]_t, [\phi(\boldsymbol{c})]_t)$, denoted by (a_j, b_j, c_j). All parties consume two unused random additive sharings prepared in Step 2, denoted by $\langle o_1 \rangle$ and $\langle o_2 \rangle$. Then all parties perform the following steps:
   (a) All parties locally derive the extended additive sharing for a_j, b_j, c_j with $(\!|a_j|\!) = \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{a})]_t$, $(\!|b_j|\!) = \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{b})]_t$ and $(\!|c_j|\!) = \phi(\boldsymbol{e}_j) \cdot [\phi(\boldsymbol{c})]_t$.
   (b) All parties locally computes $\langle x \rangle + \langle a_j \rangle + \langle o_1 \rangle$ and $\langle y \rangle + \langle b_j \rangle + \langle o_2 \rangle$, and send their shares to D.
   (c) D reconstructs $u := x + a_j$ and $v := y + b_j$. Then D sends u and v to all parties.
   (d) If the gate is an addition gate, all parties locally compute the output extended additive sharing
   $$(\!|z|\!) := (u + v) \cdot \phi(\boldsymbol{e}_j) - (\!|c_j|\!).$$
   If the gate is a multiplication gate, all parties locally compute the output extended additive sharing
   $$(\!|z|\!) := (u \cdot v) \cdot \phi(\boldsymbol{e}_j) - u \cdot (\!|b_j|\!) - v \cdot (\!|a_j|\!) + (\!|c_j|\!).$$
6. All parties output the eliminated set of parties $S := S_1 \cup S_2 \cup S_3$.

Cost of Π_{Eval}. Recall that each circuit segment has $O(n^2 \cdot m^2)$ gates, so we have $n_{\text{add}} = O(n^2 \cdot m^2/\ell)$ and $n_{\text{mult}} = O(n^2 \cdot m^2/\ell)$. It follows that the communication complexity of Step 2, Step 3, and Step 5 are all $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the communication complexity of Π_{Eval} is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

3.3 A First (Inefficient) Verification Protocol

For a given segment $\mathtt{seg}$, once the parties have executed $\Pi_{\mathsf{Eval}}(\mathtt{seg})$, they have obtained extended additive sharings of every intermediate wire value of the circuit. However, a corrupted party may have cheated during this protocol, perhaps by sending incorrect sharings to the dealer D, and/or

D itself sends incorrect reconstructions to the parties. The purpose of this section is discussing how the parties can check whether such cheating indeed took place or not. The parties do this before they proceed to the next segment evaluation. Our strategy, as outlined in Section 1.3, is to get Shamir sharings of all input groups in $\mathtt{seg}$, which will be used to verify that the openings are done correctly. These sharings are derived from Shamir sharings of the output groups in the previous segments, and possible of $\mathtt{seg}$ itself. In order to achieve this, we must apply *network routing* techniques, which are developed in the works of [GPS21,GPS22].

We begin by presenting a version of our verification protocol that does *not* yet satisfy the linear communication complexity claim, but is structurally very close to our actual protocol while requiring little preliminaries on network routing for a clear understanding.

Network routing. Consider a segment $\mathtt{seg}$, and suppose that the parties have sharings $[\phi(\boldsymbol{z})]_t$ for every output group $\boldsymbol{z}$ of each segment prior to $\mathtt{seg}$, and also of $\mathtt{seg}$ itself. Now, let $\boldsymbol{x}$ be an input group in $\mathtt{seg}$. Each entry x_i in $\boldsymbol{x}$ is the output of a previous gate, which appears in an output group of either $\mathtt{seg}$, or a prior segment. Network routing is a set of techniques that enables the parties to obtain, from the sharings of all previous groups $[\phi(\boldsymbol{z})]_t$, sharings $[\phi(\boldsymbol{x})]_t$ of the input group $\boldsymbol{x}$ in $\mathtt{seg}$. This is crucial for our verification protocol, and it is achieved by Protocol $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$ (Protocol 2 in p. 17). The construction of $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$ is a natural adaptation to the RMFE setting of the network routing techniques from [GPS21,GPS22], which are originally set in the packed secret-sharing context.

For our first verification protocol, we will use $\Pi_{\mathsf{NetworkRouting}}$ as a "black-box". Doing so results in a verification protocol with super-linear communication, stemming from the fact that all the calls to $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$ across all segments $\mathtt{seg}$ redo a lot of computation that can be done only once if one "opens the box". Later in the section we dig into the details of $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$, identifying these steps that are repeated across calls so that they are called only once, avoiding unnecessary repetition and hence achieving the desired linear communication complexity.

Fault detection. We need to introduce one more tool before we present our first (inefficient) verification protocol, Our verification ultimately boils down to ensuring that extended secret-shared values, that have been opened non-robustly using additive shares through a dealer, have actually been opened correctly. If this is not the case, the parties should be able to identify a semi-corrupted pair. This is achieved by means of a protocol $\Pi_{\mathsf{FaultDetection}}$ that takes as input an inconsistent pair of extended additive sharing $\langle\!\langle s \rangle\!\rangle$ and the masked additive sharing $\langle s \rangle + \langle o \rangle$ corresponding to it. At a high level, in this protocol the dealer will open the shares of the extended additive sharing and the masked additive sharing to find out a dispute between parties. We refer the readers to Section 3.6 for more details.

Inefficient verification. Consider a segment $\mathtt{seg}$. Let $([\phi(\boldsymbol{a}^{(i)})]_t, [\phi(\boldsymbol{b}^{(i)})]_t, [\phi(\boldsymbol{c}^{(i)})]_t)$ denote the Beaver triple corresponding to the i-th gate group in $\mathtt{seg}$, and let $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}^{(i)}$ denote the left and right inputs of the gate group. Note that before the verification of $\mathtt{seg}$, each party locally holds the values $d_j^{(i)} = x_j^{(i)} + a_j^{(i)}$ and $e_j^{(i)} = y_j^{(i)} + b_j^{(i)}$ sent by D for $i \in [N], j \in [\ell]$. All parties first check the consistency of the values sent by D by calling $\mathcal{F}_{\mathsf{VerifyBC}}$. If the values are consistent, all parties can get "temporary" input group sharings $[\phi(\boldsymbol{x})]_t$ and $[\phi(\boldsymbol{y})]_t$ from with $\boldsymbol{d}^{(i)}$, $\boldsymbol{e}^{(i)}$ and the Beaver triple $([\phi(\boldsymbol{a}^{(i)})]_t, [\phi(\boldsymbol{b}^{(i)})]_t, [\phi(\boldsymbol{c}^{(i)})]_t)$, and they use $\mathcal{F}_{\mathsf{Mult}}$ and $\mathcal{F}_{\mathsf{ReEncode}}$ to obtain $[\phi(\boldsymbol{x} \star \boldsymbol{y})]_t$.

After getting all output Shamir sharings in $\mathtt{seg}$, the parties use $\Pi_{\mathsf{NetworkRouting}}$ to obtain Shamir sharings of all the input groups $[\phi(\boldsymbol{x})]_t$ and $[\phi(\boldsymbol{y})]_t$. The parties robustly reconstruct $[\phi(\boldsymbol{x} + \boldsymbol{a})]_t$ and $[\phi(\boldsymbol{y} + \boldsymbol{b})]_t$ using $\mathcal{F}_{\mathsf{OpenPub}}$, and then they compare these outputs with the values reconstructed in the evaluation phase $\boldsymbol{d}^{(i)}$ and $\boldsymbol{e}^{(i)}$. If all entries are consistent, the evaluation of $\mathtt{seg}$ is correct. Otherwise, all parties can locate the first inconsistent $x_j^{(i)} + a_j^{(i)}$ or $y_j^{(i)} + b_j^{(i)}$. Then they eliminate a set of semi-corrupted parties, and re-evaluate the circuit segment afterwards.

The steps of the verification protocol are the following. We reiterate that this does not have linear communication complexity, but only because of the calls to $\Pi_{\mathsf{NetworkRouting}}$. This is addressed in Section 3.5 after we "open the box" of network routing in Section 3.4

1. Suppose $\mathtt{seg}$ has N gate groups. For the i-th gate group within $\mathtt{seg}$ that corresponds to the Beaver triple $([\phi(\boldsymbol{a}^{(i)})]_t, [\phi(\boldsymbol{b}^{(i)})]_t, [\phi(\boldsymbol{c}^{(i)})]_t)$ and has inputs $\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ All parties locally compute $\phi(\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)})$ for all $i \in [N]$.

2. All parties call $\mathcal{F}_{\mathsf{VerifyBC}}$ with inputs $\{\phi(\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)})\}_{i\in[N]} \cup \{\phi(\boldsymbol{y}^{(i)}+\boldsymbol{b}^{(i)})\}_{i\in[N]}$ and D. If the result is a semi-honest pair $\{P_{j_1}, P_{j_2}\}$, all parties take it as output and halt. Otherwise, run the following steps.
3. All parties locally compute $[\phi(\boldsymbol{x}^{(i)})]_t = \phi(\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)}) - [\phi(\boldsymbol{a}^{(i)})]_t$ and $[\phi(\boldsymbol{y}^{(i)})]_t = \phi(\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}) - [\phi(\boldsymbol{b}^{(i)})]_t$ for all $i \in [N]$.
4. For each additive gate group with input $[\phi(\boldsymbol{x}^{(i)})]_t$ and $[\phi(\boldsymbol{y}^{(i)})]_t$, all parties locally compute the output sharing
$$[\phi(\boldsymbol{z}^{(i)})]_t = [\phi(\boldsymbol{x}^{(i)})]_t + [\phi(\boldsymbol{y}^{(i)})]_t.$$
For all the multiplicative gate groups, suppose their indices form the set I_{mult}. All parties call $\mathcal{F}_{\mathsf{Mult}}$ with input $([\phi(\boldsymbol{x}^{(i)})]_t)_{i\in I_{\text{mult}}}$ and $([\phi(\boldsymbol{y}^{(i)})]_t)_{i\in I_{\text{mult}}}$, and get output $([w^{(i)}]_t)_{i\in I_{\text{mult}}}$ and a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - S_1$.
Then all the parties call the functionality $\mathcal{F}_{\mathsf{ReEncode}}$ with input $([w^{(i)}]_t)_{i\in I_{\text{mult}}}$ and get the output sharings $([\phi(\boldsymbol{z}^{(i)})]_t)_{i\in I_{\text{mult}}}$ and a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - S_2$.
5. All parties run the protocol $\Pi_{\mathsf{NetworkRouting}}(\texttt{seg})$ to get all Shamir sharings of $\texttt{seg}$'s gate group inputs, denoted by $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t\}_{i\in[N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t\}_{i\in[N]}$. All parties also receive a set of eliminated parties denoted by S_3. All parties update $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - S_3$.
6. All parties call $\mathcal{F}_{\mathsf{OpenPub}}$ to reconstruct the Shamir sharings $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t + [\phi(\boldsymbol{a}^{(i)})]_t\}_{i\in[N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t + [\phi(\boldsymbol{b}^{(i)})]_t\}_{i\in[N]}$, and get $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ for all $i \in [N]$.
7. Each party locally compares $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ with $\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}$ and compares $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ with $\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}$ for all $i \in [N]$. If there are any differences, all parties do the following: let $S := S_0 \cup S_1 \cup S_2 \cup S_3$. If $S \neq \emptyset$, all parties output S and $\texttt{incorrect}$ and halts. Otherwise, all parties can select a wire with inconsistent value that has the smallest topological order, denoted by $x^{(i_0)}_{j_0} + a^{(i_0)}_{j_0}$ or $y^{(i_0)}_{j_0} + b^{(i_0)}_{j_0}$. Let $\langle s\rangle + \langle o\rangle$ denote its corresponding additive sharing for reconstruction in the protocol Π_{Eval}, and let $(\!|s|\!)$ denote its extended additive sharing. Then all parties run the protocol $\Pi_{\mathsf{FaultDetection}}(D, \langle s\rangle + \langle o\rangle, (\!|s|\!))$, and get a set of eliminated parties S_4 as output. All parties update $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - S_4$, and output $S \cup S_4$ and $\texttt{incorrect}$.

3.4 Details on Network Routing

In this section we describe in detail how network routing works in order to identify the pieces that can be re-used from one call to $\Pi_{\mathsf{NetworkRouting}}$ to the next, and then we present our actual verification protocol with linear communication in Section 3.5.

Fan-Out Operations. The first ingredient of network routing is a functionality $\mathcal{F}_{\mathsf{FanOut}}$, which we describe in detail as Functionality 18 in Section E.4 in the Appendix. $\mathcal{F}_{\mathsf{FanOut}}$ takes as input a list of sharings $[\phi(\boldsymbol{z}^{(1)})]_t, \ldots, [\phi(\boldsymbol{z}^{(N)})]_t$, and also $n^{(i)}_j \in \mathbb{Z}^+$ for every $i \in [N]$ and $j \in [\ell]$ where ℓ divides $\sum_{j=1}^{\ell} n^{(i)}_j$. The functionality outputs, for each $i \in [N]$, sharings $[\phi(\boldsymbol{x}^{(i)}_j)]_t$ for $j \in [m^{(i)}]$ where $m^{(i)} := \left(\sum_{j=1}^{\ell} n^{(i)}_j\right)/\ell$, such that each $z^{(i)}_j$ appears exactly $n^{(i)}_j$ times in $[\boldsymbol{x}^{(i)}_1 \| \cdots \| \boldsymbol{x}^{(i)}_{m^{(i)}}]$. Jumping ahead, fan-out will be used to copy each output wire as many times as it is used subsequently in the circuit.

The protocol Π_{FanOut} that implements $\mathcal{F}_{\mathsf{FanOut}}$ can be found in Section E.4 in the Appendix, and its communication complexity of the protocol to generate a total of M fan-out sharings is $O(M \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Secret Collection. The second crucial operation that is needed in network routing is, given a series of Shamir sharings $([\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t)$, obtain another set of sharings $([\phi(\boldsymbol{y}_1)]_t, \ldots, [\phi(\boldsymbol{y}_N)]_t)$, where $(\boldsymbol{y}_1 \| \cdots \| \boldsymbol{y}_N)$ is a permutation of $(\boldsymbol{x}_1 \| \cdots \| \boldsymbol{x}_N)$. We refer to this operation as *secret collection*. To gain some intuition on how this helps in network routing, suppose that $\mathcal{F}_{\mathsf{FanOut}}$ has been applied to all wires in the circuit, copying them as many times as they appear in future gates, and let $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ be the all the vectors output by $\mathcal{F}_{\mathsf{FanOut}}$. Let $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ consist of all input groups to all circuit segments, and also to the output layer. We have then that $\boldsymbol{y}$ is a *permutation* of $\boldsymbol{x}$. Using secret collection, the parties can obtain sharings of each input group $[\phi(\boldsymbol{y}_i)]_t$, which is precisely what is needed for network routing.

The following theorem from [GPS21] is useful for implementing this secret collection functionality.

Theorem 2 ([GPS21]). *Suppose $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ satisfy $P \cdot \boldsymbol{x} = \boldsymbol{y}$ where P is a permutation matrix. There exists two sets of permutations $p_1, \ldots, p_N$ and $q_1, \ldots, q_N$ that permute vectors in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$, such that after applying p_i to $\boldsymbol{x}_i$ and q_j to $\boldsymbol{y}_j$ for all $i, j \in [N]$, the following property holds for an arbitrary $q_h \cdot \boldsymbol{y}_h$:*

- *Suppose that $q_h \cdot \boldsymbol{y}_h = (y'_1, \ldots, y'_\ell)$, and that y'_w corresponds to the $u_{h,w}$-th entry in $p_{v_{h,w}} \cdot \boldsymbol{x}_{v_{h,w}}$, for $w \in [\ell]$. Then, $(u_{h,1}, \ldots, u_{h,\ell})$ forms a permutation of $(1, 2, \ldots, \ell)$*

Following this theorem, we can obtain $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N)$ by first permuting each $[\phi(\boldsymbol{x}_i)]_t$ as $[\phi(p_i \cdot \boldsymbol{x}_i)]_t$. Then, for each $h \in [N]$ parties compute locally $\sum_{w=1}^{\ell} \phi(\boldsymbol{e}_w) \cdot [\phi(p_{v_{h,w}} \cdot \boldsymbol{x}_{v_{h,w}})]_t$, which creates a vector whose w-th entry is the w-th entry of $p_{v_{h,w}} \cdot \boldsymbol{x}_{v_{h,w}}$. Thanks to the theorem, this is precisely $q_h \cdot \boldsymbol{y}_h$, so the parties can obtain $[\phi(q_h \cdot \boldsymbol{y}_h)]_t$ by applying $\mathcal{F}_{\mathsf{ReEncode}}$. Finally, to obtain the desired $[\phi(\boldsymbol{y}_h)]_t$ for each $h \in [N]$, the parties can apply the inverse permutation q_h^{-1} of q_h to the sharing $[\phi(q_h \cdot \boldsymbol{y}_h)]_t$.

The permutations above are done with a functionality $\mathcal{F}_{\mathsf{Permute}}$, which we define and instantiate with Protocol Π_{Permute} in Section E.4 in the Appendix, involving a communication complexity of $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. The aggregation $\sum_{w=1}^{\ell} \phi(\boldsymbol{e}_w) \cdot [\phi(p_{v_{h,w}} \cdot \boldsymbol{x}_{v_{h,w}})]_t$ followed by the re-encoding with $\mathcal{F}_{\mathsf{ReEncode}}$ is abstracted as a functionality $\mathcal{F}_{\mathsf{Collect}}$, which we implement with a protocol Π_{Collect} in Section E.4 in the Appendix. The communication complexity of Π_{Collect} is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Network Routing for a Circuit Segment. Consider a segment $\mathtt{seg}$. We can finally describe Protocol $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$, which computes sharings $[\phi(\boldsymbol{x})]_t$ for every input group $\boldsymbol{x}$ of the segment $\mathtt{seg}$. In a nutshell, the protocol proceeds exactly as sketched above: (1) $\mathcal{F}_{\mathsf{FanOut}}$ is used to copy wires, and (2) Secret collection is performed by calling $\mathcal{F}_{\mathsf{Permute}} \to \mathcal{F}_{\mathsf{Collect}} \to \mathcal{F}_{\mathsf{Permute}}$. However, for our "non-black-box" optimization, suppose that $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg}')$ was called for the segment $\mathtt{seg}'$ that goes right before $\mathtt{seg}$. In this case, $\mathcal{F}_{\mathsf{FanOut}}$ has been performed on all output groups prior to segment $\mathtt{seg}$, and therefore we only need to do it for these output groups in $\mathtt{seg}$. Even more, since $\Pi_{\mathsf{NetworkRouting}}$ is called to verify the inputs of $\mathtt{seg}$, we only need to use $\mathcal{F}_{\mathsf{FanOut}}$ in these output wires of $\mathtt{seg}$ that are fed as inputs to $\mathtt{seg}$ itself. Later, if the check passes, then we apply $\mathcal{F}_{\mathsf{FanOut}}$ to the remaining wires. A similar optimization happens with the first call to $\mathcal{F}_{\mathsf{Permute}}$, which is done on the output groups.

The description of $\Pi_{\mathsf{NetworkRouting}}$ appears in Protocol 2. If $\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(N)}$ are the output groups of $\mathtt{seg}$, we denote by $\tilde{n}_j^{(i)}$ for $j \in [\ell], i \in [N]$ the number of times that wire $z_j^{(i)}$ is used *inside* $\mathtt{seg}$ itself. Protocol $\Pi_{\mathsf{NetworkRouting}}$ assumes that the protocol has been called for the previous segment, so these are the only remaining copies needed for getting the input groups of $\mathtt{seg}$. After this, the $\mathcal{F}_{\mathsf{Permute}} \to \mathcal{F}_{\mathsf{Collect}} \to \mathcal{F}_{\mathsf{Permute}}$ sequence is applied.

Protocol 2: $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$

1. All parties call $\mathcal{F}_{\mathsf{FanOut}}$ on all the output sharings of $\mathtt{seg}$ and $\{\tilde{n}_j^{(i)}\}_{j \in [\ell], i \in [N]}$, where the j-th wire of $\boldsymbol{z}^{(i)}$ is copied $\tilde{n}_j^{(i)}$ times (this is the number of times this wire is used in $\mathtt{seg}$ itself). All parties receive the fan-out sharings which are used for this segment's gate inputs, and a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
2. All parties call $\mathcal{F}_{\mathsf{Permute}}$ with the fan-out sharings and the desired permutations as input. All parties receive the permuted fan-out Shamir secret sharings, and a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$.
3. All parties call $\mathcal{F}_{\mathsf{Collect}}$ to get the collected Shamir sharings of $\mathtt{seg}$'s gate group inputs. All parties also receive a set of eliminated parties denoted by S_3. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_3$.
4. All parties call $\mathcal{F}_{\mathsf{Permute}}$ with the collected Shamir sharings as input, and get all Shamir sharings of $\mathtt{seg}$'s gate group inputs, denoted by $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t\}_{i \in [N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t\}_{i \in [N]}$. All parties also receive a set of eliminated parties denoted by S_4.
5. All parties output $S := S_1 \cup S_2 \cup S_3 \cup S_4$ and all the input Shamir sharings $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t\}_{i \in [N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t\}_{i \in [N]}$.

3.5 Efficient Verification

Now we are ready to "patch" the verification protocol from Section 3.3 to get linear communication. The full protocol, Π_{Verify}, is given in Section D in the Appendix, and here we only discuss the core

differences with respect to the protocol from before. As before, we do call $\Pi_{\mathsf{NetworkRouting}}$ in step 5, except that this time we take into account the fact that $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ have been called for all previous segments, and hence only need to be computed for the current segment (as described in $\Pi_{\mathsf{NetworkRouting}}$). Second, recall that the $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ calls in $\Pi_{\mathsf{NetworkRouting}}$ are only performed to the output groups needed for the input groups in the current segment `seg`. If step 7 succeeds, then the parties need to apply $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ to the remaining output groups in `seg` in preparation to the call to $\Pi_{\mathsf{NetworkRouting}}$ for the next segment.

Protocol Π_{Verify} is given in Section D in the Appendix. In that section we analyze its communication, verifying that it indeed grows linearly with n.

3.6 Fault Localization

Finally, we have focused so far in how the parties detect that cheating occurred, but we have not discussed how to react to that, identifying a semi-corrupted pair so that the segment can be re-run. Recall that a parity element $p \in \mathcal{R}$ is an element that satisfies $\mathsf{val}(p) = 0$. To mask the shares of extended additive sharing $(\!|s|\!)$ when the dealer D opens all shares, all parties will prepare a random degree-t shamir sharing of a random parity element, denoted by $[p]_t$. Then all parties will send their shares of $(\!|s|\!) + [p]_t$ to D.

We note that it is enough for D to localize a semi-corrupted pair just by opening the two sharings $(\!|s|\!) + [p]_t$ and $\langle s\rangle + \langle o\rangle$, so we introduce the following steps to let all parties disclose to D the randomness masks that they have distributed and received. We first observer that, due to the way all parties prepare random sharings (see Section B.2 in the Appendix), $\langle o\rangle$ can be written as $\langle o\rangle = \sum_{i=0}^{n'}\langle o_i\rangle$, and $[p]_t$ can be written as $[p]_t = \sum_{i=0}^{n'}[p_i]_t$, where $\langle o_i\rangle$ is the zero additive sharing distributed by P_i and $[p_i]_t$ is the parity sharing distributed by the party P_i. Each parity sharing $[p_i]_t$ corresponds to an additive sharing whose secret is 0, denoted by $\langle o'_i\rangle$. Let $\langle o'\rangle := \sum_{i=1}^{n'}\langle o'_i\rangle$. Note that D can locally compute $\langle o\rangle - \langle o'\rangle$, and $\langle o\rangle - \langle o'\rangle = \sum_{i=1}^{n'}\langle o_i\rangle - \langle o'_i\rangle$.

Following the idea in [BFO12], in order to protect the shares of $\langle o_i\rangle - \langle o'_i\rangle$ (which may leak information), each party P_i distributes another additive zero sharing $\langle o''_i\rangle$ as a mask. Then P_i reveals to D all the shares of $\langle o_i\rangle - \langle o'_i\rangle + \langle o''_i\rangle$, and also the share of $\langle o_j\rangle - \langle o'_j\rangle + \langle o''_j\rangle$ that P_i received from another party P_j. Given this information, D is able to identify disputes between parties. We summarize the full protocol $\Pi_{\mathsf{FaultDetection}}$ as follows. Note that the dealer D has the shares of $\langle s\rangle + \langle o\rangle$ sent by all parties at the beginning of the protocol, and $\langle s\rangle$ is transformed from $(\!|s|\!)$.

Protocol 3: $\Pi_{\mathsf{FaultDetection}}(D, \langle s\rangle + \langle o\rangle, (\!|s|\!))$

1. Each party P_i prepares a random additive sharing of zero denoted by $\langle o''_i\rangle$, and distribute the shares of $\langle o''_i\rangle$ to all other parties. Then all parties locally compute $\langle o''\rangle := \sum_{i=1}^{n'}\langle o''_i\rangle$.
2. All parties send their shares of $\langle o''\rangle$ to D. Then D broadcasts whether $\langle o''\rangle$ is a valid additive sharing of 0.
3. If D broadcasts that $\langle o''\rangle$ is not valid, all parties set $\langle \hat{o}\rangle := \sum_{i=1}^{n'}\langle o''_i\rangle$ and let $\langle \hat{o}_i\rangle := \langle o''_i\rangle$ for all $i \in [n']$.
4. Otherwise, all parties call $\mathcal{F}_{\mathsf{RandParity}}$ to prepare a random parity sharing $[p]_t$. If all parties receive a non-empty set of eliminated parties denoted by S, all parties take S as output as halt. Otherwise, all parties send their shares of $(\!|s|\!) + [p]_t$ to D. We note that $[p]_t$ can be written as $[p]_t = \sum_{i=1}^{n'}[p_i]_t$, and $[p_i]_t$ is distributed by the party P_i. Let $\langle o'\rangle$ denote the additive sharing transformed from $[p]_t$, and let $\langle o'_i\rangle$ denote the additive sharing transformed from $[p_i]_t$. All parties set $\langle \hat{o}\rangle := \langle o\rangle - \langle o'\rangle + \langle o''\rangle$ and let $\langle \hat{o}_i\rangle := \langle o_i\rangle - \langle o'_i\rangle + \langle o''_i\rangle$ for all $i \in [n']$.
5. Let $\hat{o}_i^{(j)}$ denote the share of $\langle \hat{o}_i\rangle$ sent by P_i to P_j. Let $(\hat{o}_j^{(i)})'$ denote the share of $\langle \hat{o}_j\rangle$ received by P_i from P_j. For all $i \in [n']$ and for all $j \in [n']$, P_i sends $\hat{o}_i^{(j)}$ and $(\hat{o}_j^{(i)})'$ to D.
6. Let $\hat{o}^{(j)}$ denote the share of $\langle \hat{o}\rangle$ held by P_j. If for some i, $\langle \hat{o}_i\rangle$ that P_i claims to have sent is not a valid additive sharing of 0, or if $\hat{o}^{(i)} \neq \sum_{j=1}^{n'}\hat{o}_j^{(i)}$, D broadcasts the message $(P_i, \texttt{incorrect})$. Otherwise, if for some j_1 and j_2 ($j_1 \neq j_2$), the share $\hat{o}_{j_1}^{(j_2)}$ that P_{j_1} claims to have sent to P_{j_2} is inconsistent to $(\hat{o}_{j_1}^{(j_2)})'$ that P_{j_2} claims to have received from P_{j_1}, D broadcasts the message $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \texttt{inconsistent})$.

7. If $(P_i, \texttt{incorrect})$ is broadcast by D, all partied set the eliminating set $E := \{D, P_i\}$. Otherwise, if $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \texttt{inconsistent})$ is broadcast by D, P_{j_1} and P_{j_2} will broadcast if they agree with this message. If P_{j_1} does not agree, all parties set the eliminating set $E := \{D, P_{j_1}\}$; otherwise if P_{j_2} does not agree, all parties set the eliminating set $E := \{D, P_{j_2}\}$; otherwise all parties set the eliminating set $E := \{P_{j_1}, P_{j_2}\}$. All parties take E as output.

Lemma 3 (Correctness of $\Pi_{\mathsf{FaultDetection}}$). *If the secrets of $\langle s \rangle + \langle o \rangle$ and $(\!|s|\!) + [p]_t$ are not consistent, and $\langle s \rangle$ is derived from $(\!|s|\!)$, then Protocol 3 can find a semi-corrupted pair E that satisfies $E \cap \mathcal{C}_{\mathtt{active}} \neq \emptyset$ when the protocol does not halt on Step 4.*

The proof of Lemma 3 can be found in Section F.1 in the Appendix.

Cost of $\Pi_{\mathsf{FaultDetection}}$. The cost of Step 1 is $O(n^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ for each party to share $\langle o_i'' \rangle$. The cost of Step 2 is $O(n)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ for all parties to send their shares of $\langle o'' \rangle$ to D, and $O(n^2)$ bits for D to broadcast if $\langle o'' \rangle$ is valid. The cost of Step 4 is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ for calling $\mathcal{F}_{\mathsf{RandParity}}$. The cost of Step 5 is $O(n^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ for each party to send $\langle \hat{o}_i \rangle$ that they sent and received to D. The cost of Step 6 is bounded by $O(n^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ for broadcasting $\hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})'$, plus $O(n^2 \log n)$ bits for broadcasting P_i or P_{j_1}, P_{j_2}. The cost of Step 7 is bounded by $O(n^2)$ bits. Recall that $m = O(\log n)$. Therefore, the overall cost of this protocol is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

4 Main Protocol

In the previous section we saw how to evaluate each segment in the circuit, and how to check if the execution was correct, identifying a semi-corrupt pair if this was not the case. In this section we show how to put together these protocols in order to evaluate the entire circuit with G.O.D., essentially by making use of the player elimination framework byBeerliová-Trubíniová and Hirt [BTH08], in which the parties that remain after a semi-corrupted pair is removed re-run the failed segment. This is described in Section 4.3. No MPC protocol would be complete without describing how input and output gates are handled. This is explained in Sections 4.1 and 4.2, respectively.

4.1 Input Gates

Since we are in the client-server model, all the inputs belong to the clients. Recall that we assume that the number of inputs for each client is a multiple of ℓ. In this part, we introduce a protocol Π_{Input}, which enables all client to share their inputs to all parties, and then properly performs fan-out and permutations to input sharings to prepare them for later use. We describe the protocol Π_{Input} below.

Protocol 4: Π_{Input}

1. For each client Client, suppose its inputs are $\{x_i^{(1)}, \ldots, x_i^{(\ell)}\}_{i=1}^N$. All parties and Client perform the following steps:
   (a) Let $\boldsymbol{x}_i := (x_i^{(1)}, \ldots, x_i^{(\ell)}) \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. Client locally computes $\phi(\boldsymbol{x}_i)$ for all $i \in [N]$.
   (b) All parties call $\mathcal{F}_{\mathsf{InputShamir}}$ with the inputs $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$ from Client. All parties get the output $[y_1]_t, \ldots, [y_N]_t$ and a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
   (c) All parties call $\mathcal{F}_{\mathsf{RandShamir}}(N)$ to prepare N random degree-t shamir secret sharings denoted by $[\phi(\boldsymbol{r}_1)]_t, \ldots, [\phi(\boldsymbol{r}_N)]_t$. All parties also receive a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$.
   (d) All parties call $\mathcal{F}_{\mathsf{OpenPub}}(N)$ to reconstruct $[y_1]_t + [\phi(\boldsymbol{r}_1)]_t, \ldots, [y_N]_t + [\phi(\boldsymbol{r}_N)]_t$ and get the secrets $y_1 + \phi(\boldsymbol{r}_1), \ldots, y_N + \phi(\boldsymbol{r}_N)$.
   (e) For all $i \in [N]$, All parties locally check if $\phi \circ \psi(y_i + \phi(\boldsymbol{r}_i)) = y_i + \phi(\boldsymbol{r}_i)$. If the equation does not hold, all parties set $[y_i]_t$ to $[0]_0$. Note that after this step, each sharing $[y_i]_t$ can be written as $[y_i]_t = [\phi(\boldsymbol{x}_i')]_t$, where $\boldsymbol{x}_i' = \psi(y_i)$.
   (f) All parties take $[\phi(\boldsymbol{x}_1')]_t, \ldots, [\phi(\boldsymbol{x}_N')]_t$ as the shared inputs of Client.

2. All parties call $\mathcal{F}_{\mathsf{FanOut}}$ with the input sharings from all clients, and get the set of eliminated parties denoted by S_3 and the resulting fan-out sharings. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_3$.
3. All parties call $\mathcal{F}_{\mathsf{Permute}}$ to permute all the fan-out sharings with the desired permutations. All parties get the set of eliminated parties denoted by S_4, and the resulting permuted sharings. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_4$.
4. All parties output the resulting permuted sharings in the previous step.

Cost of Π_{Input}. To get input from a client with $N \cdot \ell$ inputs in $\mathbb{Z}/2^k\mathbb{Z}$, the communication complexity is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. The communication complexity of Step 2 is $O(M \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$, and the communication complexity of Step 3 is $O(M \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Since M is bounded by $O(|C'|/\ell)$, and $m, \ell = O(\log n)$, the total communication complexity of Π_{Input} is $O(|C| \cdot n + c \cdot n^3 \cdot m^2 + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

4.2 Output Gates

We introduce the protocol Π_{Output} that reconstructs all outputs towards all clients. In this protocol, all parties first performs network routing to generate the output Shamir sharings, and then they send the shares of the output sharing to each client. The description of Π_{Output} appears below.

Protocol 5: Π_{Output}

1. All parties call $\mathcal{F}_{\mathsf{Collect}}$ to collect secrets for Shamir sharings of the output layer, and all parties receive a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
2. All parties call $\mathcal{F}_{\mathsf{Permute}}$ with the collected Shamir sharings and the desired permutations as input. All parties get all output Shamir sharings as output, and get a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$.
3. For each client Client that has output Shamir sharings $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$, all parties send their shares of $[\phi(\boldsymbol{z}_i)]_t$ to Client for all $i \in [N]$. Then Client reconstructs the secrets $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N$.

Cost of Π_{Output}. In Step 1, the communication complexity is bounded by calling $\mathcal{F}_{\mathsf{Collect}}$ that outputs $|C'|/\ell$ sharings, so the communication complexity is bounded by $O((|C| + \ell \cdot c) \cdot n \cdot m/\ell + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. In step 2, the communication complexity is bounded by $O(n \cdot |C'|) = O(|C| \cdot n + n \cdot c \cdot \ell)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Since m, ℓ are both $O(\log n)$, the total communication complexity of Π_{Output} is $O(|C| \cdot n + c \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

4.3 Main Protocol

Given the above protocols, the main protocol that implements the ideal functionality $\mathcal{F}_{\mathsf{Main}}$ follows in a straightforward way. The main protocol is introduced in Π_{Main}.

Protocol 6: $\Pi_{\mathsf{Main}}(C)$

1. Let C denote the circuit. All parties transform C to C'. All parties agree on the gate grouping, and they order of the circuit segments according to topological ordering. All parties set $\mathcal{P}_{\mathtt{active}} := \mathcal{P}$.
2. All parties run the protocol Π_{Input}.
3. All parties evaluate the circuit segments according to their ordering. For each circuit segment denoted by $\mathtt{seg}$:
   (a) All parties run the protocol $\Pi_{\mathsf{Eval}}(\mathtt{seg})$. All parties get the set of eliminated parties denoted by S.
   (b) All parties run the protocol $\Pi_{\mathsf{Verify}}(\mathtt{seg}, S)$. If the output is $\mathtt{incorrect}$, all parties repeat step 3.(a) and 3.(b) to evaluate $\mathtt{seg}$. Otherwise, all parties continue to evaluate the next circuit segment.
4. All parties run the protocol Π_{Output}.

Cost of Π_{Main}. In Step 3, each time Π_{Eval} or Π_{Verify} outputs a semi-honest pair of parties, at least one corrupted party is removed from $\mathcal{P}_{\mathtt{active}}$, so Π_{Eval} and Π_{Verify} will be repeated at most $t = O(n)$ times. Note that each time Π_{Eval} or Π_{Verify} is repeated, the communication complexity is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Overall, this introduces an overhead of $O(n^4 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$ to the total communication complexity.

In Step 3, when no circuit segment is repeated, the total communication complexity of Π_{Eval} and Π_{Verify} is

$$\sum_{\mathtt{seg}} \big(O(n^3 \cdot m^2) + O(M_{\mathtt{seg}} \cdot n \cdot m)\big) = O(|C'| \cdot n + n^3 \cdot m^2 + (\sum_{\mathtt{seg}} M_{\mathtt{seg}}) \cdot n \cdot m).$$

We note that $\sum_{\mathtt{seg}} M_{\mathtt{seg}}$ is bounded by $O(|C'|/\ell)$. So the total communication complexity of Step 3 when no circuit segment is repeated is $O((|C'|/\ell) \cdot n \cdot m + |C'| \cdot n + n^3 \cdot m^2)$.

Recall that $m, \ell = O(\log n)$. Therefore, the overall communication complexity of Π_{Main} is

$$O(|C| \cdot n + c \cdot n^3 \cdot \log^2 n + n^4 \cdot \log^2 n)$$

elements in $\mathbb{Z}/2^k\mathbb{Z}$. Here $|C|$ is the size of the circuit in *both* addition and multiplication gates. Notice that this is *linear*, as desired.

Lemma 4. *Protocol* Π_{Main} *securely computes* $\mathcal{F}_{\mathsf{Main}}$ *in the* ($\mathcal{F}_{\mathsf{InputShamir}}$, $\mathcal{F}_{\mathsf{RandShamir}}$, $\mathcal{F}_{\mathsf{RandZeroAdd}}$, $\mathcal{F}_{\mathsf{RandParity}}$, $\mathcal{F}_{\mathsf{OpenPub}}$, $\mathcal{F}_{\mathsf{ReEncode}}$, $\mathcal{F}_{\mathsf{Mult}}$, $\mathcal{F}_{\mathsf{TripleAdd}}$, $\mathcal{F}_{\mathsf{TripleMult}}$, $\mathcal{F}_{\mathsf{VerifyBC}}$, $\mathcal{F}_{\mathsf{FanOut}}$, $\mathcal{F}_{\mathsf{Permute}}$, $\mathcal{F}_{\mathsf{Collect}}$)*-hybrid model with perfect security against a fully malicious adversary who controls* $t < n/3$ *parties.*

The proof of Lemma 4 can be found in Section F.3 in the Appendix. This leads to the following Theorem, which is the main result of our work.

Theorem 3. *In the client-server model, let* c *denote the number of clients, and* $n = 3t + 1$ *denote the number of parties (servers). Let* k *be a constant positive integer and let* $\mathbb{Z}/2^k\mathbb{Z}$ *be a finite ring of constant size. For an arithmetic circuit* C *over* $\mathbb{Z}/2^k\mathbb{Z}$*, let* $|C|$ *denote the size of the circuit. There exists an information-theoretic MPC protocol which securely computes the arithmetic circuit with perfect security in the presence of a fully malicious adversary controlling up to* c *clients and* t *parties. The communication complexity of this protocol is* $O(|C| \cdot n + \mathsf{poly}(c, n))$ *elements in* $\mathbb{Z}/2^k\mathbb{Z}$.

Acknowledgments

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful. 2024 JP Morgan Chase & Co. All rights reserved.

Y. Song was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003.

References

ACD+19. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, and Chen Yuan. Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 471–501, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

ACD+20. Mark Abspoel, Ronald Cramer, Ivan Damgård, Daniel Escudero, Matthieu Rambaud, Chaoping Xing, and Chen Yuan. Asymptotically good multiplicative LSSS over Galois rings and applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 151–180, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

ACE+21. Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 222–244, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

BFO12. Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Heidelberg, Germany.

BGP92. Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. *Computer Science: Research and Applications*, pages 313–321, 1992.

BTH08. Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

CCXY18. Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 395–426, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

CRX21. Ronald Cramer, Matthieu Rambaud, and Chaoping Xing. Asymptotically-good arithmetic secret sharing over $\mathbb{Z}/p^\ell\mathbb{Z}$ with strong multiplication and its applications to efficient MPC. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 656–686, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

CW92. Brian A Coan and Jennifer L Welch. Modular construction of a byzantine agreement protocol with optimal message bit complexity. *Information and Computation*, 97(1):61–85, 1992.

DEF+19. Ivan Damgård, Daniel Escudero, Tore Kasper Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure MPC over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy*, pages 1102–1120, San Francisco, CA, USA, May 19–23, 2019. IEEE Computer Society Press.

DIK10. Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

DN07. Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

ELXY23. Daniel Escudero, Hongqing Liu, Chaoping Xing, and Chen Yuan. Degree-d reverse multiplication-friendly embeddings: Constructions and applications. *Asiacrypt*, 2023.

FR22. Thibauld Feneuil and Matthieu Rivain. Threshold linear secret sharing to the rescue of MPC-in-the-head. Cryptology ePrint Archive, Report 2022/1407, 2022. https://eprint.iacr.org/2022/1407.

FY92. Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

GLS19. Vipul Goyal, Yanyi Liu, and Yifan Song. Communication-efficient unconditional MPC with guaranteed output delivery. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 85–114, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany.

GPS21. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Unconditional communication-efficient MPC via hall's marriage theorem. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 275–304, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

GPS22. Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. Sharing transformation and dishonest majority MPC with packed secret sharing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 3–32, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

GSZ20. Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 618–646, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

HMP00. Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. Efficient secure multi-party computation. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

IKP+16. Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 430–458, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

PS21. Antigoni Polychroniadou and Yifan Song. Constant-overhead unconditionally secure multiparty computation over binary fields. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 812–841, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Wan03. Zhe-Xian Wan. *Lectures on finite fields and Galois rings*. World Scientific Publishing Company, 2003.

Appendix

A Full Description of Building Block Functionalities

Full Description of $\mathcal{F}_{\mathsf{OpenPub}}$. The full description of $\mathcal{F}_{\mathsf{OpenPub}}$ appears in Functionality 2.

Functionality 2: $\mathcal{F}_{\mathsf{OpenPub}}(N)$

1. Let $[x_1]_d, [x_2]_d, \ldots, [x_N]_d$ denote the input sharings.
2. For all $i \in [N]$, $\mathcal{F}_{\mathsf{OpenPub}}$ receives from the active honest parties their shares of $[x_i]_d$. Then $\mathcal{F}_{\mathsf{OpenPub}}$ reconstructs the secrets x_i. $\mathcal{F}_{\mathsf{OpenPub}}$ further computes the shares of $[x_i]_d$ held by active corrupted parties, and sends the shares to the adversary.
3. For all $i \in [N]$, $\mathcal{F}_{\mathsf{OpenPub}}$ sends the reconstructed x_i to all active parties.

Full Description of $\mathcal{F}_{\mathsf{Mult}}$. The full description of $\mathcal{F}_{\mathsf{Mult}}$ appears in Functionality 3.

Functionality 3: $\mathcal{F}_{\mathsf{Mult}}(N)$

1. Let $[x_1]_t, \ldots, [x_N]_t$ and $[y_1]_t, \ldots, [y_N]_t$ denote the input sharings. $\mathcal{F}_{\mathsf{Mult}}$ receives from active honest parties their shares of $[x_1]_t, \ldots, [x_N]_t$ and $[y_1]_t, \ldots, [y_N]_t$. Then $\mathcal{F}_{\mathsf{Mult}}$ reconstructs the secrets $x_1, \ldots, x_N$ and $y_1, \ldots, y_N$. $\mathcal{F}_{\mathsf{Mult}}$ further computes the shares of $[x_1]_t, \ldots, [x_N]_t$ and $[y_1]_t, \ldots, [y_N]_t$ held by active corrupted parties and send the shares to the adversary.
2. $\mathcal{F}_{\mathsf{Mult}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ S such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{Mult}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{Mult}}$ then sends S to all active honest parties.
3. $\mathcal{F}_{\mathsf{Mult}}$ computes $x_1 \cdot y_1, \ldots, x_N \cdot y_N$, and then $\mathcal{F}_{\mathsf{Mult}}$ receives from the adversary the shares of active corrupted parties of $[x_1 \cdot y_1]_t, \ldots, [x_N \cdot y_N]_t$.
4. $\mathcal{F}_{\mathsf{Mult}}$ samples the whole sharings $[x_1 \cdot y_1]_t, \ldots, [x_N \cdot y_N]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{Mult}}$ sends P_h's shares of $[x_1 \cdot y_1]_t, \ldots, [x_N \cdot y_N]_t$ to P_h.

Full Description of $\mathcal{F}_{\mathsf{ReEncode}}$. The full description of $\mathcal{F}_{\mathsf{ReEncode}}$ appears in Functionality 4.

Functionality 4: $\mathcal{F}_{\mathsf{ReEncode}}(N)$

1. Let $[x_1]_t, \ldots, [x_N]_t$ denote the input sharings. $\mathcal{F}_{\mathsf{ReEncode}}$ receives from active honest parties their shares of $[x_1]_t, \ldots, [x_N]_t$. Then $\mathcal{F}_{\mathsf{ReEncode}}$ reconstructs the secrets $x_1, \ldots, x_N$. $\mathcal{F}_{\mathsf{ReEncode}}$ further computes the shares of $[x_1]_t, \ldots, [x_N]_t$ held by active corrupted parties and send the shares to the adversary.
2. $\mathcal{F}_{\mathsf{ReEncode}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ S such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{ReEncode}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{ReEncode}}$ then sends S to all active honest parties.
3. $\mathcal{F}_{\mathsf{ReEncode}}$ computes $\phi \circ \psi(x_1), \ldots, \phi \circ \psi(x_N)$, and then $\mathcal{F}_{\mathsf{ReEncode}}$ receives from the adversary the shares of active corrupted parties of $[\phi \circ \psi(x_1)]_t, \ldots, [\phi \circ \psi(x_N)]_t$.
4. $\mathcal{F}_{\mathsf{ReEncode}}$ samples the whole sharings $[\phi \circ \psi(x_1)]_t, \ldots, [\phi \circ \psi(x_N)]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{ReEncode}}$ sends P_h's shares of $[\phi \circ \psi(x_1)]_t, \ldots, [\phi \circ \psi(x_N)]_t$ to P_h.

Full Description of $\mathcal{F}_{\mathsf{VerifyBC}}$. The full description of $\mathcal{F}_{\mathsf{VerifyBC}}$ appears in Functionality 5.

Functionality 5: $\mathcal{F}_{\mathsf{VerifyBC}}(N)$

1. Let D be the suspicious dealer and let N be the total number of ring elements to verify. Let $x^{(1)}, \ldots, x^{(N)}$ denote these elements.
2. $\mathcal{F}_{\mathsf{VerifyBC}}$ receives from active honest parties the values of $x^{(1)}, \ldots, x^{(N)}$, and send them to the adversary.
3. $\mathcal{F}_{\mathsf{VerifyBC}}$ receives from the adversary one of the following:
   - Default string only if active honest parties shares are all consistent;

- A semi-corrupted pair $\{P_{j_1}, P_{j_2}\}$ where $\{P_{j_1}, P_{j_2}\} \cap \mathcal{C}_{\texttt{active}} \neq \emptyset$.

4. Based on what it has received, $\mathcal{F}_{\mathsf{VerifyBC}}$ does one of the following:
   - On receiving default string, $\mathcal{F}_{\mathsf{VerifyBC}}$ sends `consistent` to active honest parties;
   - On receiving a semi-corrupted pair, $\mathcal{F}_{\mathsf{VerifyBC}}$ sends $(\texttt{inconsistent}, \{P_{j_1}, P_{j_2}\})$ to active honest parties.

Full Description of $\mathcal{F}_{\mathsf{InputShamir}}$. The full description of $\mathcal{F}_{\mathsf{InputShamir}}$ appears in Functionality 6.

Functionality 6: $\mathcal{F}_{\mathsf{InputShamir}}(N)$

1. Let $x_1, \ldots, x_N \in \mathcal{R}$ denote the inputs of the client denoted by Client.
2. $\mathcal{F}_{\mathsf{InputShamir}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\texttt{active}}$ S such that $|S|/2 \leq |S \cap \mathcal{C}_{\texttt{active}}|$. $\mathcal{F}_{\mathsf{InputShamir}}$ updates $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - S$. $\mathcal{F}_{\mathsf{InputShamir}}$ then sends S to all active honest parties.
3. $\mathcal{F}_{\mathsf{InputShamir}}$ receives from the adversary one of the following:
   - The shares of active corrupted parties of $[x_1]_t, \ldots [x_N]_t$.
   - If Client is corrupted, the adversary can choose to change the inputs to $x'_1, \ldots, x'_N$. Then the adversary sends the shares of active corrupted parties of $[x'_1]_t, \ldots [x'_N]_t$
4. Based on what it has received, $\mathcal{F}_{\mathsf{InputShamir}}$ does one of the following:
   - On receiving the shares of active corrupted parties of $[x_1]_t, \ldots [x_N]_t$, $\mathcal{F}_{\mathsf{InputShamir}}$ samples a random degree-t Shamir secret sharing $[x_1]_t, \ldots [x_N]_t$ so that they are compatible with the active corrupted parties' shares for all $i \in [N]$. For each active honest party P_h, $\mathcal{F}_{\mathsf{InputShamir}}$ sends P_h's shares of $[x_1]_t, \ldots [x_N]_t$ to P_h.
   - On receiving a new set of inputs $x'_1, \ldots, x'_N$ and the shares of active corrupted parties of $[x'_1]_t, \ldots [x'_N]_t$, $\mathcal{F}_{\mathsf{InputShamir}}$ samples a random degree-t Shamir secret sharing $[x'_1]_t, \ldots [x'_N]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{InputShamir}}$ sends P_h's shares of $[x'_1]_t, \ldots [x'_N]_t$ to P_h for all $i \in [N]$.

B Preparing Random Sharings

B.1 Preliminaries

We first introduce several important tools to help us instantiate our protocols that prepare different random secret sharings.

Byzantine Agreement. Byzantine agreement (BA) is a protocol that takes a bit from each party as input and enables all honest parties to reach a consensus on a single bit. In the case that all parties hold the same bit as input then the output bit coincides precisely with this input bit. In our protocol, we use BA in a few places, like letting all parties reach a binary consensus on whether an error happened. We also use BA to let one party broadcast one bit to all other parties consistently, meaning all parties receive the same bit, which is the bit the party sent if this party is honest. This is easily instantiable from BA by letting the party send the bit to all other parties, followed by BA to reach a consensus on the bit received.

With $t < n/3$, both bit consensus and broadcast can be achieved by a perfect byzantine agreement protocol communicating $O(n^2)$ bits [BGP92,CW92]. This quadratic complexity is fine in our context since we only use this primitive a small number of times so that it does not affect our overall linear communication.

Packed Shamir Secret Sharings. We will use the packed Shamir secret sharing scheme [FY92] in this work to prepare randomness for network routing. For the Galois ring $\mathcal{R} = \mathsf{GR}(2^k, m)$, suppose $(\alpha_i)_{i=1}^n, (\beta_j)_{j=1}^n$ are $2n$ distinct points to interpolate polynomials according to Lemma 1.

For all $1 \leq r \leq n$, a degree-d ($d \geq r-1$) packed Shamir sharing of $\boldsymbol{x} \in \mathcal{R}^r$ is a vector $(s_1, s_2, \ldots, s_n) \in \mathcal{R}^n$ that satisfies the property that there exists a polynomial $f(\cdot) \in \mathcal{R}^{\leq d}[x]$ with $f(\beta_j) = x_j, \forall j \in [r]$ and $f(\alpha_i) = s_i, \forall i \in [n]$. The share held by party P_i is s_i. With any $(d+1)$

different shares of the same sharing, we can reconstruct the secret $\boldsymbol{x}$. When $r = 1$, it corresponds to the standard Shamir sharing.

A degree-d packed Shamir sharing for the vector $\boldsymbol{x} \in \mathcal{R}^r$ is denoted as $[\boldsymbol{x}]_d$, and a degree-d standard Shamir sharing for the element $y \in \mathcal{R}$ is denoted as $[y]_d$. For packed Shamir sharings, the following two properties hold:

- For all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{R}^r$, $[\boldsymbol{x} + \boldsymbol{y}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$.
- Let $\star$ denote the coordinate-wise multiplication, then for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{R}^r$ and for all $d_1, d_2 \geq r-1$ subject to $d_1 + d_2 < n$, we have $[\boldsymbol{x} \star \boldsymbol{y}]_{d_1+d_2} = [\boldsymbol{x}]_{d_1} \cdot [\boldsymbol{y}]_{d_2}$.

We note that Lemma 2 also holds for packed Shamir secret sharings.

Abstract General Linear Secret Sharing Schemes. We adopt the idea of an abstract definition of a general linear secret sharing scheme (GLSSS) in [ACD+20]. We borrow the following notations from [ACD+20].

For two non-empty sets U and $\mathcal{I}$, $U^{\mathcal{I}}$ represents the indexed Cartesian product $\prod_{i \in \mathcal{I}} U$. For some non-empty $A \subset \mathcal{I}$, the natural projection π_A maps one tuple $u = (u_i)_{i \in \mathcal{I}} \in U^{\mathcal{I}}$ to the tuple $(u_i)_{i \in A} \in U^A$. We also suppose that $\mathcal{S}$ is a finite quotient ring.

We use the definition of arithmetic secret sharing scheme from [ACD+20].

Definition 3 (Abstract $\mathcal{S}$-GLSSS [ACD+20]). *The syntax of an $\mathcal{S}$-arithmetic secret sharing scheme Σ consists of the following data:*

- *A set of parties $\mathcal{I} = \{1, \ldots, n\}$.*
- *A secret space $Z = \mathcal{S}^r$. r is also denoted as the number of secrets packed within Σ.*
- *A share space $U = \mathcal{S}^\ell$. ℓ is also denoted as the share size.*
- *A sharing space $C \subset U^{\mathcal{I}}$, where $U^{\mathcal{I}}$ denotes the intended Cartesian product $\prod_{i \in \mathcal{I}} U$.*
- *An injective $\mathcal{S}$-module homomorphism: $\mathtt{share} : Z \times \mathcal{S}^k \to C$, that maps a secret $\boldsymbol{x} \in Z$ and a random tape $\boldsymbol{\rho} \in \mathcal{S}^k$ to a sharing $[\![\boldsymbol{x}]\!] \in C$. $\mathtt{share}$ is also denoted as the sharing map of Σ.*
- *A surjective $\mathcal{S}$-module homomorphism: $\mathtt{rec} : C \to Z$, which takes as input a sharing $[\![x]\!] \in C$ and outputs a secret $\boldsymbol{x} \in Z$. rec is also denoted as the reconstruction map of Σ.*

The scheme Σ satisfies that for all $\boldsymbol{x} \in Z$ and $\rho \in R^k$, $\mathtt{rec}(\mathtt{share}(\boldsymbol{x}, \boldsymbol{\rho})) = \boldsymbol{x}$. We may refer to Σ as the 6-tuple $(n, Z, U, C, \mathtt{share}, \mathtt{rec})$.

Definition 4 (Privacy Set and Reconstruction Set [CCXY18]). *Suppose $A \subset \mathcal{I}$ is non-empty. We say A is a* privacy set *if for all $\boldsymbol{x}_0, \boldsymbol{x}_1 \in Z$, and for all vector $\boldsymbol{v} \in U^A$,*

$$\Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_0, \boldsymbol{\rho})) = \boldsymbol{v}] = \Pr_{\boldsymbol{\rho}}[\pi_A(\mathtt{share}(\boldsymbol{x}_0, \boldsymbol{\rho})) = \boldsymbol{v}].$$

We say A is a reconstruction set *if there is an $\mathcal{S}$-module homomorphism $\mathtt{rec}_A : \pi_A(C) \to Z$, such that for all $[\![\boldsymbol{x}]\!] \in C$,*

$$\mathtt{rec}_A(\pi_A([\![\boldsymbol{x}]\!])) = \mathtt{rec}([\![\boldsymbol{x}]\!]).$$

Intuitively, for a privacy set A, the shares of all parties in A are independent of the secret; for a reconstruction set A, the shares of all parties in A fully determine the secret.

In [CCXY18], the interleaved GLSSS $\Sigma^{\times m}$ is defined as an n-party secret sharing scheme which corresponds to m Σ-sharings. We introduce the *tensoring-up lemma* from [CCXY18], which can be easily adapted to our ring case:

Proposition 1 (Tensoring-up Lemma [CCXY18]). *Let $\mathcal{R}$ be a degree-m Galois ring containing the quotient ring $\mathcal{S}$ and let Σ be a $\mathcal{S}$-GLSSS. Then the m-fold interleaved $\mathcal{S}$-GLSSS $\Sigma^{\times m}$ is naturally viewed as an $\mathcal{R}$-GLSSS, compatible with its $\mathcal{S}$-linearity.*

Suppose $[\![x]\!]$ is a sharing in Σ, with this proposition we can define the linear operation for every $\lambda \in \mathcal{R}$ such that for all $[\![\boldsymbol{x}]\!] = ([\![x_1]\!], \ldots, [\![x_m]\!]) \in \Sigma^{\times m}$ we have:

- $\lambda \cdot [\![\boldsymbol{x}]\!] = (\lambda \cdot [\![x_1]\!], \ldots, \lambda \cdot [\![x_m]\!])$ for all $\lambda \in \mathcal{S}$,
- $\lambda_1 \cdot [\![\boldsymbol{x}]\!] + \lambda_2 \cdot [\![\boldsymbol{x}]\!] = (\lambda_1 + \lambda_2) \cdot [\![\boldsymbol{x}]\!]$ for all $\lambda_1, \lambda_2 \in \mathcal{R}$ and
- $\lambda_1 \cdot (\lambda_2 \cdot [\![\boldsymbol{x}]\!]) = (\lambda_1 \cdot \lambda_2) \cdot [\![\boldsymbol{x}]\!]$ for all $\lambda_1, \lambda_2 \in \mathcal{R}$.

An Example of a GLSSS Using The Tensoring-up Lemma. We will use the standard Shamir secret sharing scheme as an example of a GLSSS and show how to use the tensoring-up lemma. For ring $\mathcal{S}$ of size $|\mathcal{S}| \geq n+1$, we may define a secret sharing scheme Σ which takes $x \in \mathcal{S}$ as input and outputs $[x]_t$, a degree-t Shamir sharing. The secret space and the share space of Σ are $\mathcal{S}$. According to the Lagrange interpolation, the secret x can be written as a $\mathcal{S}$-linear combination of all the shares. Therefore, the defining map of Σ is $\mathcal{S}$-linear. Therefore, Σ is a $\mathcal{S}$-GLSSS.

A sharing $[\![\boldsymbol{x}]\!] = ([\![x_1]\!], \ldots, [\![x_m]\!]) \in \Sigma^{\times m}$ is a vector of m sharings in Σ. Let $\mathcal{R}$ be a degree-m Galois ring of $\mathcal{S}$. According to the tensoring-up lemma $\Sigma^{\times m}$ is a $\mathcal{R}$-GLSSS. Therefore $\mathcal{R}$-linear operations can be performed to the sharings in $\Sigma^{\times m}$.

Hyper-Invertible Matrices. We adopt the definition of hyper-invertible matrices from [BTH08].

Definition 5. *An r-by-c matrix $\mathcal{M}$ over the Galois ring $\mathcal{R}$ is hyper-invertible if for any index sets $R \subseteq \{1, 2, \ldots, r\}$ and $C \subseteq \{1, 2 \cdots, c\}$ with $|R| = |C| > 0$, the matrix $\mathcal{M}_R^C$ is invertible, where $\mathcal{M}_R$ denotes the matrix consisting of the rows $i \in R$ of $\mathcal{M}$ and $\mathcal{M}^C$ denotes the matrix consisting of the columns $j \in C$ of $\mathcal{M}$, and $\mathcal{M}_R^C = (\mathcal{M}_R)^C$.*

We note that an n-by-n hyper-invertible matrix exists if the degree of the Galois ring $\mathcal{R} = \mathsf{GR}(2^k, m)$ satisfies $2^m \geq 2n + 1$ [ACD+19]. In our work, we will use n-by-n hyper-invertible matrices over $\mathcal{R}$.

B.2 Preparing Random Sharings for $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS

In this section, we present the protocol for preparing random sharings for a given general $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS scheme, denoted by Σ. Our protocol is adapted from [PS21], while we introduce party elimination framework to ensure perfect security. Let $[\![x]\!]$ denote a sharing in Σ of secret x. For a set $A \subset \mathcal{I}$, recall that $\pi_A([\![x]\!])$ refers to the share of $[\![x]\!]$ held by parties in A. We assume that Σ satisfies the following property:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i \in A}$ for the parties in A, let

$$\Sigma(A, (a_i)_{i \in A}) := \{[\![x]\!] \mid [\![x]\!] \in \Sigma \text{ and } \pi_A([\![x]\!]) = (a_i)_{i \in A}\}.$$

  Then, there is an efficient algorithm that outputs either that $\Sigma(A, (a_i)_{i \in A}) = \emptyset$ or a random sharing $[\![x]\!]$ in $\Sigma(A, (a_i)_{i \in A})$.

The functionality $\mathcal{F}_{\mathsf{Rand}}$ is described in Functionality 7. Essentially, $\mathcal{F}_{\mathsf{Rand}}$ allows the adversary to specify the shares held by active corrupted parties or to generate a valid semi-corrupted pair. In the first case, based on these shares $\mathcal{F}_{\mathsf{Rand}}$ generates a random sharing in Σ and distributes the shares to active honest parties. Note that when the set of active corrupted parties is a privacy set, the secret is independent of the shares chosen by the adversary. In the second case, the semi-corrupted pair will be eliminated from the active parties.

Functionality 7: $\mathcal{F}_{\mathsf{Rand}}(N)$

1. $\mathcal{F}_{\mathsf{Rand}}$ receives from the adversary one of the following:
   - The shares of active corrupted parties $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ where $\Sigma(\mathcal{C}_{\mathtt{active}}, (s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}) \neq \emptyset$ for all $j \in [N]$;
   - A semi-corrupted pair $\{P_{j_1}, P_{j_2}\}$ where $\{P_{j_1}, P_{j_2}\} \cap \mathcal{C}_{\mathtt{active}} \neq \emptyset$.
2. Based on what it has received, $\mathcal{F}_{\mathsf{Rand}}$ does one of the following:
   - On receiving $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ for all $j \in [N]$, $\mathcal{F}_{\mathsf{Rand}}$ randomly samples $[\![r]\!] \in \Sigma(\mathcal{C}_{\mathtt{active}}, (s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}})$ for all $j \in [N]$. For each honest party $P_i \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{Rand}}$ sends the i-th share of $[\![r^{(j)}]\!]$ to P_i for all $j \in [N]$.
   - On receiving $\{P_{j_1}, P_{j_2}\}$, $\mathcal{F}_{\mathsf{Rand}}$ sends $\{P_{j_1}, P_{j_2}\}$ to all active honest parties.

We follow the idea in [BTH08] to design a protocol that prepares random sharings in Σ. To sum up, firstly each party generates a random sharing in $\Sigma^{\times m}$. Each sharing $[\![\boldsymbol{x}]\!]$ in $\Sigma^{\times m}$ is a vector of m sharings denoted by $([\![x_1]\!], [\![x_2]\!], \ldots, [\![x_m]\!])$, and according to Proposition 1 $\Sigma^{\times m}$ is a

$\mathcal{R}$-GLSSS. Secondly, all parties apply a hyper-invertible matrix over $\mathcal{R}$ to transform the sharings for fault detection. Next, some selected parties will verify that a subset of the transformed sharings have the correct form. Finally, all parties take the other transformed sharings as the output.

Protocol 7: $\Pi_{\mathsf{Rand}}(N)$

1. All parties agree on a hyper-invertible matrix $\mathcal{M}$ over $\mathcal{R}$ of size n'-by-n'.
2. Each party in $\mathcal{P}_{\mathtt{active}}$ uniformly randomly samples $\lceil \frac{N}{m \cdot T} \rceil$ different sharings in $\Sigma^{\times m}$. Let $[\![\boldsymbol{s}_i^{(j)}]\!]$ denote the j-th sharing sampled by P_i. Then each party in $\mathcal{P}_{\mathtt{active}}$ distributes the shares to all the other parties in $\mathcal{P}_{\mathtt{active}}$.
3. For all $j \in \{1, 2, \ldots, \lceil \frac{N}{m \cdot T} \rceil\}$, all parties in $\mathcal{P}_{\mathtt{active}}$ locally compute
$$([\![\boldsymbol{r}_1^{(j)}]\!], [\![\boldsymbol{r}_2^{(j)}]\!], \ldots, [\![\boldsymbol{r}_{n'}^{(j)}]\!]) = \mathcal{M} \cdot ([\![\boldsymbol{s}_1^{(j)}]\!], [\![\boldsymbol{s}_2^{(j)}]\!], \ldots, [\![\boldsymbol{s}_{n'}^{(j)}]\!])$$
4. For all $j \in \{1, 2, \ldots, \lceil \frac{N}{m \cdot T} \rceil\}$, and for $\ell \in \{T+1, \ldots, n'\}$, all parties in $\mathcal{P}_{\mathtt{active}}$ send their shares of $[\![\boldsymbol{r}_\ell^{(j)}]\!]$ to P_h. P_h checks if the sharing $[\![\boldsymbol{r}_\ell^{(j)}]\!]$ is a valid sharing in $\Sigma^{\times m}$. If not, P_h sets its happy-bit to `unhappy`.
5. **Fault Detection Phase**:
   (a) All players broadcast their happy-bits using the byzantine agreement protocol.
   (b) For each party, if it receives at least one `unhappy` then sets his or her happy-bit to `unhappy`.
   (c) All players run a consensus protocol on their happy-bits. If the consensus is `happy`, all parties output the output of π and terminate the procedure. Otherwise, they proceed to the following steps.
6. **Fault Localization Phase**:
   (a) All players agree the party with the smallest index in $\mathcal{P}_A$ as the dealer D. All other players send all their generated values and communication to D.
   (b) On receiving all the information, D simulates π and the fault detection phase itself. D either prepares the message $(P_i, \mathtt{corrupt})$ if P_i failed to follow the protocol, or the message $(P_{j_1}, P_{j_2}, \mathtt{idx}, \mathtt{b}, \mathtt{b}', \mathtt{disputed})$ if the $\mathtt{idx}$-th bit $\mathtt{b}$ that P_{j_1} sends to P_{j_2} is inconsistent with P_{j_2}'s bit $\mathtt{b}'$ that P_{j_2} claims to have received. Then D broadcasts the prepared message to all players.
   (c) If $(P_i, \mathtt{corrupt})$ is received, all players set the eliminating set $E = \{D, P_i\}$.
   Otherwise, if $(P_{j_1}, P_{j_2}, \mathtt{idx}, \mathtt{b}, \mathtt{b}', \mathtt{disputed})$ is received, P_{j_1} and P_{j_2} will broadcast if they agree with this message. If P_{j_1} does not agree, all players set the eliminating set $E = \{D, P_{j_1}\}$; otherwise if P_{j_2} does not agree, all players set the eliminating set $E = \{D, P_{j_2}\}$; otherwise all players set the eliminating set $E = \{P_{j_1}, P_{j_2}\}$.
   (d) All players update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - E$ and halt.
7. For all $j \in \{1, 2, \ldots, \lceil \frac{N}{m \cdot T} \rceil\}$ and for all $i \in [T]$, all active parties separate the sharing $[\![\boldsymbol{r}_i^{(j)}]\!]$ into m sharings in Σ, and take them as outputs.

Lemma 5. *In Step 4 of Protocol 7, either at least one active honest party has* `unhappy` *happy-bit or all active corrupted parties has dealt consistent shares to active honest parties in $\Sigma^{\times m}$ in Step 2 of Π_{Rand}, i.e. $\Sigma(\mathcal{H}_{\mathtt{active}}, ([\![\boldsymbol{s}_\ell^{(j)}]\!]_i)_{i \in \mathcal{H}_{\mathtt{active}}}) \neq \emptyset$ where $[\![\boldsymbol{s}_\ell^{(j)}]\!]_i$ denotes party P_i's share of the sharing $[\![\boldsymbol{s}_\ell^{(j)}]\!]$.*

Proof. If all actives honest parties are happy, then at least t' active parties have verified that $[\![\boldsymbol{r}_\ell^{(j)}]\!]$ is valid for all $j \in \lceil \frac{N}{m \cdot T} \rceil$, and we suppose without loss of generality that $[\![\boldsymbol{r}_\ell^{(j)}]\!]$ is valid for $\ell \in \{T+1, \ldots, T+t'\}$. And also we suppose without loss of generality that $[\![\boldsymbol{s}_i^{(j)}]\!]$ is valid for $i \in \{1, \ldots, n'-t'\}$. Then we have that
$$([\![\boldsymbol{r}_{T+1}^{(j)}]\!], \ldots, [\![\boldsymbol{r}_{T+t'}^{(j)}]\!]) = \mathcal{M}_{\mathcal{H}_{\mathtt{active}}} \cdot ([\![\boldsymbol{s}_1^{(j)}]\!], \ldots, [\![\boldsymbol{s}_{n'-t'}^{(j)}]\!]) + \mathcal{M}_{\mathcal{C}_{\mathtt{active}}} \cdot ([\![\boldsymbol{s}_{n'-t'+1}^{(j)}]\!], \ldots, [\![\boldsymbol{s}_{n'}^{(j)}]\!]),$$
where $\mathcal{M}_{\mathcal{H}_{\mathtt{active}}} = \mathcal{M}_{\{T+1,\ldots,T+t'\}}^{\{1,\ldots,n'-t'\}}$ and $\mathcal{M}_{\mathcal{C}_{\mathtt{active}}} = \mathcal{M}_{\{T+1,\ldots,T+t'\}}^{\{n'-t'+1,\ldots,n'\}}$. Because $\mathcal{M}$ is a hyper-invertible matrix, $\mathcal{M}_{\mathcal{C}}$ is an invertible matrix. Therefore, the shares of $[\![\boldsymbol{s}_{n'-t'+1}^{(j)}]\!], \ldots, [\![\boldsymbol{s}_{n'}^{(j)}]\!]$ held by active honest parties are also consistent.

It follows Lemma 5 that if all active honest parties are happy then the shares of $[\![\boldsymbol{r}_\ell^{(j)}]\!]$ held by active honest parties are consistent for all $\ell \in \{1, \ldots, n'\}$, and all active honest parties hold $\lceil N/m \rceil$ consistent sharings in $\Sigma^{\times m}$.

Cost of Protocol 7. Suppose the share size of a sharing in Σ is sh ring elements in $\mathbb{Z}/2^k\mathbb{Z}$. Then the share size of a sharing in $\Sigma^{\times m}$ is $m \cdot \mathsf{sh}$ ring elements in $\mathbb{Z}/2^k\mathbb{Z}$. It follows that the communication complexity of $\Pi_{\mathsf{Rand}}(N)$ is $O(N \cdot n \cdot \mathsf{sh} + n^2 \cdot m \cdot \mathsf{sh})$.

Lemma 6. *The protocol Π_{Rand} computes $\mathcal{F}_{\mathsf{Rand}}$ with perfect security when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. Recall that $\mathcal{C}_{\mathtt{active}}$ denotes the active corrupted parties and that $\mathcal{H}_{\mathtt{active}}$ denotes the active honest parties.

Simulation for Π_{Rand}. $\mathtt{Sim}$ simulates the protocol Π_{Rand} as follows:

1. In Step 2, for each honest party, $\mathtt{Sim}$ generates random shares of active corrupted parties in $\Sigma^{\times m}$. Then $\mathtt{Sim}$ sends these shares of active corrupted parties on behalf of active honest parties to $\mathcal{A}$. For each corrupted party, $\mathtt{Sim}$ receives the active honest parties' shares of the active corrupted parties' sharings from $\mathcal{A}$. Suppose that the sharings generated by active corrupted parties are $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i \in \mathcal{C}_{\mathtt{active}}}$ for all $j \in \{1, 2, \ldots, \lceil \frac{N}{m \cdot T} \rceil\}$. In this step, $\mathtt{Sim}$ only receives from $\mathcal{A}$ the shares of active honest parties of these shares.
2. In Step 4, to simulate the happy-bit, $\mathtt{Sim}$ computes the shares held by active honest parties for each sharing generated by the corrupted party. Next, $\mathtt{Sim}$ can compute the active honest parties' shares according to $\mathcal{M}^{\mathcal{C}_{\mathtt{active}}}_{\{T+1,\ldots,n'\}} \cdot (\llbracket \boldsymbol{s}_{n'-t'+1}^{(j)} \rrbracket, \ldots, \llbracket \boldsymbol{s}_{n'}^{(j)} \rrbracket)$. Suppose the result sharing is denoted by $(\llbracket \boldsymbol{w}_{T+1}^{(j)} \rrbracket, \ldots, \llbracket \boldsymbol{w}_{n'}^{(j)} \rrbracket)$. Then $\mathtt{Sim}$ receives from $\mathcal{A}$ the shares of $(\llbracket \boldsymbol{w}_{T+1}^{(j)} \rrbracket, \ldots, \llbracket \boldsymbol{w}_{n'}^{(j)} \rrbracket)$ that the active corrupted parties send to the active honest parties. For $\ell \in \{T+1, \ldots, n'\}$, if P_h is an honest party, $\mathtt{Sim}$ checks if the sharing $\llbracket \boldsymbol{w}_{\ell}^{(j)} \rrbracket$ is a valid sharing in $\Sigma^{\times m}$. If not, $\mathtt{Sim}$ sets P_h's happy-bit to be $\mathtt{unhappy}$ on behalf of P_h.
   If P_h is corrupted, $\mathtt{Sim}$ does nothing.
3. In Fault Detection Phase, $\mathtt{Sim}$ faithfully follows the byzantine agreement protocol using active honest parties' happy-bits.
4. If proceeds to Fault Localization Phase, $\mathtt{Sim}$ randomly samples the sharings distributed by active honest parties in Step 2 of Π_{Rand} so that the sampled sharings are consistent with the shares of active corrupted parties. Then $\mathtt{Sim}$ faithfully follows the protocol.
5. In Step 7, $\mathtt{Sim}$ compute the shares of $\llbracket \boldsymbol{r}_i^{(j)} \rrbracket$ held by active corrupted parties for all $i \in [T]$ and $j \in [\lceil \frac{N}{m \cdot T} \rceil]$, and $\mathtt{Sim}$ sends the shares of the first N sharings held by active corrupted parties to $\mathcal{F}_{\mathsf{Rand}}$.

Hybrid Arguments. We show that $\mathtt{Sim}$ perfectly simulates the behaviors of honest parties.

Hybrid$_0$: The execution in the real world.

Hybrid$_1$: In Step 2, for each $i \in \mathcal{H}_{\mathtt{active}}$, instead of receiving the entire share from P_i, $\mathtt{Sim}$ only samples the active corrupted parties' shares. This will not change the distribution of the shares held by active honest parties. The distribution is identical to the distribution of **Hybrid$_0$**.

Hybrid$_2$: In Step 4, $\mathtt{Sim}$ simulates the happy-bits of active honest parties. Because the happy-bit of some honest parties only depends on the consistency of the active corrupted parties' shares, $\mathtt{Sim}$ can perfectly simulate active honest parties' happy-bits. The distribution is identical to the distribution of **Hybrid$_1$**.

Hybrid$_3$: In Fault Detection Phase, $\mathtt{Sim}$ simulates the active honest parties' behavior using the simulated active honest parties' happy-bits instead. Because the two sets of active honest parties' happy-bits are identical, the distribution of this hybrid is identical to the distribution of **Hybrid$_2$**.

Hybrid$_4$: $\mathtt{Sim}$ simulates Fault Localization Phase by sampling random sharings that are compatible with active corrupted parties shares for active honest parties in Step 2. Note that the $\Sigma^{\times m}$ sharings sampled by $\mathtt{Sim}$ have the same distribution as the actual sharings sampled by active honest parties. Also note that it will not affect the output distribution, since the semi-corrupted pair is only related to the corrupted parties' messages. Therefore, the distribution of this hybrid is identical to the distribution of **Hybrid$_3$**.

Hybrid$_5$: According to Lemma 5, if $\mathtt{Sim}$ proceeds to Step 7 in Π_{Rand} then all active honest parties have been dealt consistent sharings from active corrupted parties in Step 2. With these sharings, $\mathcal{A}$ locally holds the shares of $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i \in \mathcal{C}_{\mathtt{active}}}$ that are consistent with the shares of active honest

parties. Specifically, active corrupted parties hold consistent $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{C}_{\mathtt{active}}}$ for all $j \in [\lceil \frac{N}{m\cdot T} \rceil]$, where $\llbracket \boldsymbol{s}_i^{(j)} \rrbracket \in \Sigma(\mathcal{H}_{\mathtt{active}}, (s_i^{(j)})_{i\in\mathcal{H}_{\mathtt{active}}})$ for $i \in \mathcal{C}_{\mathtt{active}}$. We have

$$(\llbracket \boldsymbol{r}_i^{(j)} \rrbracket)_{i=1}^{T} = \mathcal{M}_{[T]} \cdot (\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i=1}^{n'} = \mathcal{M}_{[T]}^{\mathcal{H}_{\mathtt{active}}} \cdot (\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{H}_{\mathtt{active}}} + \mathcal{M}_{[T]}^{\mathcal{C}_{\mathtt{active}}} \cdot (\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{C}_{\mathtt{active}}}.$$

$\mathtt{Sim}$ computes the active corrupted parties' shares of $(\llbracket \boldsymbol{r}_i^{(j)} \rrbracket)_{i=1}^{T}$ using this equation and gets $(r_i^{(j)})_{r\in\mathcal{C}_{\mathtt{active}}}$ for all $j \in [\lceil \frac{N}{m\cdot T} \rceil]$. $\mathtt{Sim}$ sends $(r_i^{(j)})_{r\in\mathcal{C}_{\mathtt{active}}}$ for all $j \in [\lceil \frac{N}{m\cdot T} \rceil]$ to $\mathcal{F}_{\mathsf{Rand}}$.

According to the property of Σ, the shares of $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{H}_{\mathtt{active}}}$ held by active corrupted parties are independent of the sharings $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{C}_{\mathtt{active}}}$, and are also independent of the shares of $(\llbracket \boldsymbol{s}_i^{(j)} \rrbracket)_{i\in\mathcal{H}_{\mathtt{active}}}$ held by active honest parties. Because $|\mathcal{H}_{\mathtt{active}}| \geq T$, $(\llbracket \boldsymbol{r}_i^{(j)} \rrbracket)_{i=1}^{T}$ corresponds to $(\llbracket \boldsymbol{r}_i^{(j)} \rrbracket)_{i\in\mathcal{H}_{\mathtt{active}}}$ and has no correlation. It follows that the active honest parties' shares of $(\llbracket \boldsymbol{r}_i^{(j)} \rrbracket)_{i=1}^{T}$ are independent of the view of $\mathcal{A}$ and are uniformly random. Therefore, the distribution of this hybrid is identical to the distribution of $\mathbf{Hybrid_4}$.

Note that $\mathbf{Hybrid_5}$ is the execution in the ideal world.

B.3 Instantiating Functionalities

A Robust Version of $\mathcal{F}_{\mathsf{Rand}}$. To instantiate functionalities, we first define a functionality $\mathcal{F}_{\mathsf{RandRobust}}$ to prepare random sharings for a $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS Σ with guaranteed output delivery. The description of $\mathcal{F}_{\mathsf{RandRobust}}$ appears in Functionality 8.

Functionality 8: $\mathcal{F}_{\mathsf{RandRobust}}(N)$

1. $\mathcal{F}_{\mathsf{RandRobust}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{RandRobust}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{RandRobust}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{RandRobust}}$ receives from the adversary the shares of active corrupted parties $(s_i^{(j)})_{i\in\mathcal{C}_{\mathtt{active}}}$ where $\Sigma(\mathcal{C}_{\mathtt{active}}, (s_i^{(j)})_{i\in\mathcal{C}_{\mathtt{active}}}) \neq \emptyset$ for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{RandRobust}}$ randomly samples $\llbracket r \rrbracket \in \Sigma(\mathcal{C}_{\mathtt{active}}, (s_i^{(j)})_{i\in\mathcal{C}_{\mathtt{active}}})$ for all $j \in [N]$. For each honest party $P_h \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandRobust}}$ sends the h-th share of $\llbracket r^{(j)} \rrbracket$ to P_h for all $j \in [N]$.

We show the protocol that implements $\mathcal{F}_{\mathsf{RandRobust}}$ as follows.

Protocol 8: $\Pi_{\mathsf{RandRobust}}(N)$

1. All parties set $S := \emptyset$.
2. All parties call $\mathcal{F}_{\mathsf{Rand}}(N)$. If the result is N random Σ sharings $\{\llbracket r^{(i)} \rrbracket\}_{i=1}^{N}$, all parties output these sharings and S. Otherwise, if the result is a semi-corrupted pair E, all parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - E$ and $S := S \cup E$, and then repeat this step.
3. All parties output S.

Suppose the share size of a sharing in Σ is sh ring elements in $\mathbb{Z}/2^k\mathbb{Z}$. The cost of this protocol is $O(N \cdot n \cdot \mathsf{sh} + n^3 \cdot m \cdot \mathsf{sh})$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 7. *The protocol $\Pi_{\mathsf{RandRobust}}$ computes $\mathcal{F}_{\mathsf{RandRobust}}$ with perfect security in the $\mathcal{F}_{\mathsf{Rand}}$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 2, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{Rand}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

Random Shamir Secret Sharings. The functionality $\mathcal{F}_{\mathsf{RandShamir}}$ enables all parties to prepare N random degree-t Shamir sharings in the form of $[\phi(\boldsymbol{r})]_t$, where $\boldsymbol{r}$ is a random vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$. The description of $\mathcal{F}_{\mathsf{RandShamir}}$ appears in Functionality 9.

Functionality 9: $\mathcal{F}_{\mathsf{RandShamir}}(N)$

1. $\mathcal{F}_{\mathsf{RandShamir}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{RandShamir}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{RandShamir}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{RandShamir}}$ receives from the adversary the shares of active corrupted parties $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ where $s_i^{(j)} \in \mathcal{R}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{RandShamir}}$ randomly samples N vectors $\boldsymbol{r}^{(j)}$ in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ for all $j \in [N]$. Then $\mathcal{F}_{\mathsf{RandShamir}}$ generates N degree-t Shamir sharings $([(\phi(\boldsymbol{r}^{(j)}))]_t)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(s_i^{(j)})_{j=1}^N$. For each $P_h \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandShamir}}$ sends the i-th share of $[(\phi(\boldsymbol{r}^{(j)}))]_t$ to P_h for all $j \in [N]$.

The instantiation of $\mathcal{F}_{\mathsf{RandShamir}}$ can be found in the next part where we describe the instantiation of the functionality that prepares random packed Shamir sharings.

Random Packed Shamir secret sharings. The functionality $\mathcal{F}_{\mathsf{RandShamir}}$ enables all parties to prepare N random degree-t Shamir sharings in the form of $[\phi(\boldsymbol{r})]_t$, where $\boldsymbol{r}$ is a random vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$. The description of $\mathcal{F}_{\mathsf{RandShamir}}$ appears in Functionality 9. We introduce the functionality to let all parties prepare random degree-d packed Shamir sharings in the form of $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$, where $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_h$ are h random vectors in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$. The description of $\mathcal{F}_{\mathsf{RandShPacked}}$ appears in Functionality 10.

Functionality 10: $\mathcal{F}_{\mathsf{RandShPacked}}(d, h, N)$

1. $\mathcal{F}_{\mathsf{RandShPacked}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ S such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{RandShPacked}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{RandShPacked}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{RandShPacked}}$ receives from the adversary the shares of active corrupted parties $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ where $s_i^{(j)} \in \mathcal{R}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{RandShPacked}}$ randomly samples $N \cdot h$ vectors $\boldsymbol{r}_q^{(j)}$ in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ for $q \in [h]$ and $j \in [N]$. Then $\mathcal{F}_{\mathsf{RandShPacked}}$ generates N degree-d Shamir sharings $([(\phi(\boldsymbol{r}_1^{(j)}), \phi(\boldsymbol{r}_2^{(j)}), \ldots, \phi(\boldsymbol{r}_h^{(j)}))]_d)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(s_i^{(j)})_{j=1}^N$. For each $P_i \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandShPacked}}$ sends the i-th share of $[(\phi(\boldsymbol{r}_1^{(j)}), \phi(\boldsymbol{r}_2^{(j)}), \ldots, \phi(\boldsymbol{r}_h^{(j)}))]_d$ to P_i for all $j \in [N]$.

We use $\mathcal{F}_{\mathsf{RandRobust}}$ to instantiate $\mathcal{F}_{\mathsf{RandShPacked}}$. Consider a secret sharing scheme Σ which takes h vectors $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_h \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ as input and outputs a degree-d packed Shamir sharing of $(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))$ in $\mathcal{R}$, i.e. $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$. Because packed Shamir sharings are linear in $\mathcal{R}$ and ϕ is $\mathbb{Z}/2^k\mathbb{Z}$-linear, Σ is an $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS. To use $\mathcal{F}_{\mathsf{RandRobust}}$ to prepare random sharings in Σ, we need to show that:

– Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i \in A}$ for parties in A, there exists an efficient algorithm which outputs either $\Sigma(A, (a_i)_{i \in A}) = \emptyset$ or a random sharing $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d \in \Sigma(A, (a_i)_{i \in A})$.

Depending on the size of A, there are three cases:

– If $|A| \geq d + 1$, by the property of packed Shamir secret sharing scheme, the set of shares $\{a_i\}_{i \in A}$ can fully determine the whole sharing if exists. The algorithm checks whether the shares and the secrets lie on a polynomial $f(\cdot)$ of degree at most d. If not, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Otherwise, the algorithm checks whether the secret of this Shamir sharing, denoted by $w_1, \ldots, w_\ell$, is in the image of ϕ. This can be done by checking whether $\phi(\psi(w_i)) = w_i$ holds for all $i = 1, \ldots, \ell$. If not, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Otherwise, the algorithm outputs the Shamir sharing determined by $\{a_i\}_{i \in A}$.
– If $d - h + 2 \leq |A| \leq d$, the set of shares $\{a_i\}_{i \in A}$ may not be independent because of the structure of the secret space. Recall that the secret space is $Z \subset \mathcal{R}^h$, and Z is $\mathcal{R}$-linear. Suppose Z is defined by the linear mapping $B : (\mathbb{Z}/2^k\mathbb{Z})^{\ell \cdot h} \to (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h}$. Since packed Shamir secret sharing scheme is also $\mathcal{R}$-linear, given the shares $\{a_i\}_{i \in A}$ the secret space can be defined as the image of the mapping $\mathcal{L} : (\mathbb{Z}/2^k\mathbb{Z})^{(d-|A|+2) \cdot h} \to (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h}$, and $\mathcal{L}$ can be written in the

form that $\mathcal{L}(\boldsymbol{a}) = C \cdot \boldsymbol{a} + \boldsymbol{b}$, where the matrix $C \in (\mathbb{Z}/2^k\mathbb{Z})^{(m\cdot h)\times((d-|A|+2)\cdot h)}$ and the vector $\boldsymbol{b} \in (\mathbb{Z}/2^k\mathbb{Z})^{m\cdot h}$ are determined by the shares $\{a_i\}_{i\in A}$. It is clear that $\Sigma(A,(a_i)_{i\in A}) \neq \emptyset$ if and only if $B \cdot \boldsymbol{x} = C \cdot \boldsymbol{y} + \boldsymbol{b}$ has solution. This can be determined easily by solving if the following linear equation has solutions

$$(B, C) \cdot \begin{pmatrix} -\boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{b},$$

and this can be done by solving the null space of the matrix (B, C). If the null space is empty, the algorithm outputs $\Sigma(A,(a_i)_{i\in A}) = \emptyset$. Otherwise, the algorithm picks random $(-\boldsymbol{x}_0, \boldsymbol{y}_0)^T$ in the null space. Suppose the secrets corresponding to the sampled vector $\boldsymbol{y}_0$ are $\phi(\boldsymbol{r}_1), \ldots, \phi(\boldsymbol{r}_\ell)$. Then, based on the secrets $\phi(\boldsymbol{r}_1), \ldots, \phi(\boldsymbol{r}_\ell)$ and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the whole packed Shamir sharing $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$ and outputs $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$.

– If $|A| \leq d - h + 1$, by the property of packed Shamir secret sharing scheme the set of shares $\{a_i\}_{i\in A}$ are independent, so $\Sigma(A,(a_i)_{i\in A}) \neq \emptyset$. The algorithm randomly samples $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_\ell \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ and randomly samples $(d+1-h-|A|)$ elements in $\mathcal{R}$ as the shares of the first $(d+1-h-|A|)$ parties in $\mathcal{I}\backslash A$. Then, based on the secrets $\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h)$, the shares of the first $(d+1-h-|A|)$ parties in $\mathcal{I}\backslash A$ and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the whole packed Shamir sharing $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$ and outputs $[(\phi(\boldsymbol{r}_1), \phi(\boldsymbol{r}_2), \ldots, \phi(\boldsymbol{r}_h))]_d$.

Therefore, $\mathcal{F}_{\mathsf{RandShPacked}}$ can be instantiated by $\mathcal{F}_{\mathsf{RandRobust}}$ with the secret sharing scheme Σ defined above. Note that the share size of Σ is $\mathsf{sh} = m$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. It follows that when using $\Pi_{\mathsf{Rand}}(N)$ to instantiate $\mathcal{F}_{\mathsf{RandRobust}}$ the total communication complexity to generate N random sharings in Σ is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

We note that $\mathcal{F}_{\mathsf{RandShamir}}$ is a special case of this functionality when $d = t$ and $h = 1$, and therefore $\mathcal{F}_{\mathsf{RandShamir}}$ can be instantiated by $\mathcal{F}_{\mathsf{RandRobust}}$. Also note that all parties can run the protocol Π_{Rand} to prepare N random packed Shamir sharings.

Random Zero Additive Sharings. The description of $\mathcal{F}_{\mathsf{RandZeroAdd}}$ appears in Functionality 11.

Functionality 11: $\mathcal{F}_{\mathsf{RandZeroAdd}}(N)$

1. $\mathcal{F}_{\mathsf{RandZeroAdd}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathsf{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathsf{active}}|$. $\mathcal{F}_{\mathsf{RandZeroAdd}}$ updates $\mathcal{P}_{\mathsf{active}} := \mathcal{P}_{\mathsf{active}} - S$. $\mathcal{F}_{\mathsf{RandZeroAdd}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{RandZeroAdd}}$ receives from the adversary the shares of active corrupted parties $(s_i^{(j)})_{i\in\mathcal{C}_{\mathsf{active}}\cap\{1,2,\ldots,t+1\}}$ where $s_i^{(j)} \in \mathbb{Z}/2^k\mathbb{Z}$ for all $i \in \mathcal{C}_{\mathsf{active}} \cap \{1, 2, \ldots, t+1\}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{RandZeroAdd}}$ randomly samples N additive sharings $(\langle o_j\rangle)_{j=1}^N$ such that $o_j = 0$ for all $j \in [N]$. For each $P_h \in \mathcal{H}_{\mathsf{active}} \cap \{1, 2, \ldots, t+1\}$, $\mathcal{F}_{\mathsf{RandZeroAdd}}$ sends the h-th share of $\langle o_j\rangle$ to P_h for all $j \in [N]$.

We will use $\mathcal{F}_{\mathsf{RandRobust}}$ to instantiate $\mathcal{F}_{\mathsf{RandZeroAdd}}$. Consider a secret sharing scheme Σ which outputs an additive sharing of 0 in $\mathbb{Z}/2^k\mathbb{Z}$ among the first $t+1$ parties, and the shares of the remaining parties are fixed to be 0. It is shown in [PS21] that the following properties holds for Σ:

– Σ is an $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS.
– Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, there exists an efficient algorithm which outputs either $\Sigma(A,(a_i)_{i\in A}) = \emptyset$ or a random sharing $\langle o\rangle \in \Sigma(A,(a_i)_{i\in A})$.

Therefore, $\mathcal{F}_{\mathsf{RandZeroAdd}}$ can be instantiated by $\mathcal{F}_{\mathsf{RandRobust}}$ with the secret sharing scheme Σ defined above. Note that the share size of Σ is $\mathsf{sh} = 1$ element in $\mathbb{Z}/2^k\mathbb{Z}$. It follows that when using $\Pi_{\mathsf{Rand}}(N)$ to instantiate $\mathcal{F}_{\mathsf{RandRobust}}$ the total communication complexity to generate N random sharings in Σ is $O(N \cdot n + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Random Zero Packed Shamir Sharings. When we prepare randomness needed for sharing transformation, all parties need to prepare random degree-d packed Shamir sharings whose secrets are 0. More specifically, all parties need to prepare sharings in the form of $([(o_1^{(j)}, o_2^{(j)}, \ldots, o_h^{(j)})]_d)_{j=1}^N$,

where o_i is an element in $\mathcal{R}$ that satisfies $\psi(o_i) = \mathbf{0} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ for all $i \in [h]$ and for all $j \in [N]$. We add constraints to d and h such that $d-h+1 \geq t'$ and $d \leq n'-1$. The description of $\mathcal{F}_{\mathsf{RandZeroSh}}$ appears in Functionality 12.

Functionality 12: $\mathcal{F}_{\mathsf{RandZeroSh}}(d, h, N)$

1. $\mathcal{F}_{\mathsf{RandZeroSh}}$ receives from the adversary one of the following:
   - The shares of active corrupted parties $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ where $s_i^{(j)} \in \mathcal{R}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$;
   - A semi-corrupted pair $\{P_{j_1}, P_{j_2}\}$ where $\{P_{j_1}, P_{j_2}\} \cap \mathcal{C}_{\mathtt{active}} \neq \emptyset$.
2. Based on what it has received, $\mathcal{F}_{\mathsf{RandZeroSh}}$ does one of the following:
   - On receiving $(s_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ for all $j \in [N]$, $\mathcal{F}_{\mathsf{RandZeroSh}}$ generates N degree-d Shamir sharings $([(o_1^{(j)}, o_2^{(j)}, \ldots, o_h^{(j)})]_d)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(s_i^{(j)})_{j=1}^N$. For each $P_i \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandZeroSh}}$ sends the i-th share of $[(o_1^{(j)}, o_2^{(j)}, \ldots, o_h^{(j)})]_d$ to P_i for all $j \in [N]$.
   - On receiving $\{P_{j_1}, P_{j_2}\}$, $\mathcal{F}_{\mathsf{RandZeroSh}}$ sends $\{P_{j_1}, P_{j_2}\}$ to all active honest parties.

We use $\mathcal{F}_{\mathsf{RandRobust}}$ to instantiate $\mathcal{F}_{\mathsf{RandZeroSh}}$. Consider a secret sharing scheme Σ which outputs a degree-d packed Shamir sharing of $(o_1, o_2, \ldots, o_h)$ in $\mathcal{R}$, i.e. $[(o_1, o_2, \ldots, o_h)]_d$. Let V denote the set of all elements o in $\mathcal{R}$ such that $\psi(o) = \mathbf{0} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. Recall that $\psi(\cdot)$ is $\mathbb{Z}/2^k\mathbb{Z}$-linear. Therefore, for all $o_1, o_2 \in V$ and for all $\alpha, \beta \in \mathbb{Z}/2^k\mathbb{Z}$, we have

$$\psi(\alpha \cdot o_1 + \beta \cdot o_2) = \alpha \cdot \psi(o_1) + \beta \cdot \psi(o_2) = \mathbf{0}$$

which implies that $\alpha \cdot o_1 + \beta \cdot o_2 \in V$. Therefore, V is a $\mathbb{Z}/2^k\mathbb{Z}$-subspace. Since packed Shamir sharings are linear in $\mathcal{R}$, Σ is an $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS.

Also, there exists an efficient algorithm to sample a uniformly random element in V. Let $\mathbf{1}$ be a vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ such that all entries are $1 \in \mathbb{Z}/2^k\mathbb{Z}$. The algorithm random samples $x \in \mathcal{R}$ and computes $\boldsymbol{y} := \psi(x)$. Then, the algorithm outputs $x - \phi(\mathbf{1}) \cdot \phi(\boldsymbol{y})$. Note that $\psi(x - \phi(\mathbf{1}) \cdot \phi(\boldsymbol{y})) = \psi(x) - \mathbf{1} \star \boldsymbol{y} = \mathbf{0}$. Therefore, $x - \phi(\mathbf{1}) \cdot \phi(\boldsymbol{y}) \in V$. Also note that for all $o \in V$, o will be output by the algorithm only when x satisfies that $o + \phi(\mathbf{1}) \cdot \phi(\boldsymbol{y}) = x$ for some $\boldsymbol{y} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. Because x is uniformly random in $\mathcal{R}$, the probability that the algorithm outputs o is $|(\mathbb{Z}/2^k\mathbb{Z})^\ell|/|\mathcal{R}|$. Therefore, the algorithm outputs a uniformly random element in V.

To use $\mathcal{F}_{\mathsf{RandRobust}}$ to prepare random sharings in Σ, we need to show that:

- Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i \in A}$ for parties in A, there exists an efficient algorithm which outputs either $\Sigma(A, (a_i)_{i \in A}) = \emptyset$ or a random sharing $[(o_1, o_2, \ldots, o_h)]_d \in \Sigma(A, (a_i)_{i \in A})$.

To this property, depending on the size of A, there are two cases:

- If $|A| \geq d + 1$, by the property of packed Shamir secret sharing scheme, the set of shares $\{a_i\}_{i \in A}$ can fully determine the whole sharing if exists. The algorithm checks whether the shares and the secrets lie on a polynomial $f(\cdot)$ of degree at most d. If not, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Otherwise, the algorithm outputs the Shamir sharing determined by $\{a_i\}_{i \in A}$.
- If $d-h+2 \leq |A| \leq d$, the set of shares $\{a_i\}_{i \in A}$ may not be independent because of the structure of the secret space. Recall that the secret space is $V \subset \mathcal{R}^h$, and V is $\mathcal{R}$-linear. Suppose V is defined by the linear mapping $B : (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h} \to (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h}$. Since packed Shamir secret sharing scheme is also $\mathcal{R}$-linear, given the shares $\{a_i\}_{i \in A}$ the secret space can be defined as the image of the mapping $\mathcal{L} : (\mathbb{Z}/2^k\mathbb{Z})^{(d-|A|+2) \cdot h} \to (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h}$, and $\mathcal{L}$ can be written in the form that $\mathcal{L}(\boldsymbol{a}) = C \cdot \boldsymbol{a} + \boldsymbol{b}$, where the matrix $C \in (\mathbb{Z}/2^k\mathbb{Z})^{(m \cdot h) \times ((d-|A|+2) \cdot h)}$ and the vector $\boldsymbol{b} \in (\mathbb{Z}/2^k\mathbb{Z})^{m \cdot h}$ are determined by the shares $\{a_i\}_{i \in A}$. It is clear that $\Sigma(A, (a_i)_{i \in A}) \neq \emptyset$ if and only if $B \cdot \boldsymbol{x} = C \cdot \boldsymbol{y} + \boldsymbol{b}$ has solution. This can be determined easily by solving if the following linear equation has solutions

  $$(B, C) \cdot \begin{pmatrix} -\boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{b},$$

  and this can be done by solving the null space of the matrix (B, C). If the null space is empty, the algorithm outputs $\Sigma(A, (a_i)_{i \in A}) = \emptyset$. Otherwise, the algorithm picks random $(-\boldsymbol{x}_0, \boldsymbol{y}_0)^T$ in the null space. Suppose the secrets corresponding to the sampled vector $\boldsymbol{y}_0$ are $o_1, o_2, \ldots, o_h$. Then, based on the secrets $o_1, o_2, \ldots, o_h$ and the shares $\{a_i\}_{i \in A}$, the algorithm reconstructs the whole packed Shamir sharing $[(o_1, o_2, \ldots, o_h)]_d$ and outputs $[(o_1, o_2, \ldots, o_h)]_d$.

– If $|A| \leq d-h+1$, by the property of packed Shamir secret sharing scheme, the set of shares $\{a_i\}_{i\in A}$ is independent, so $\Sigma(A,(a_i)_{i\in A}) \neq \emptyset$. The algorithm randomly samples $(d+1-h-|A|)$ elements in $\mathcal{R}$ as the shares of the first $(d+1-h-|A|)$ parties in $\mathcal{I}\backslash A$. Then, based on the secrets $o_1,\ldots,o_h$, the shares of the first $(d+1-h-|A|)$ parties in $\mathcal{I}\backslash A$ and the shares $\{a_i\}_{i\in A}$, the algorithm reconstructs the whole packed Shamir sharing $[(o_1,o_2,\ldots,o_h)]_d$ and outputs $[(o_1,o_2,\ldots,o_h)]_d$.

Therefore, $\mathcal{F}_{\mathsf{RandZeroSh}}$ can be instantiated by $\mathcal{F}_{\mathsf{RandRobust}}$ with the secret sharing scheme Σ defined above. Note that the share size of Σ is $\mathsf{sh} = m$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. It follows that when using $\Pi_{\mathsf{RandRobust}}(N)$ to instantiate $\mathcal{F}_{\mathsf{RandRobust}}$ the total communication complexity to generate N random sharings in Σ is $O(N\cdot n\cdot m + n^2\cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Random Parity Sharings. The functionality $\mathcal{F}_{\mathsf{RandParity}}$ that prepares random parity sharings is shown in Functionality 13.

Functionality 13: $\mathcal{F}_{\mathsf{RandParity}}(N)$

1. $\mathcal{F}_{\mathsf{RandParity}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S\cap\mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{RandParity}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{RandParity}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{RandParity}}$ receives from the adversary the shares of active corrupted parties $(u_i^{(j)})_{i\in\mathcal{C}_{\mathtt{active}}}$ where $s_i^{(j)} \in \mathcal{R}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{RandParity}}$ randomly samples N parity elements $p_1,\ldots,p_N \in \mathcal{R}$. Then $\mathcal{F}_{\mathsf{RandParity}}$ generates N degree-t Shamir sharing $([p_j]_t)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(u_i^{(j)})_{j=1}^N$. For each $P_h \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandParity}}$ sends the i-th share of $[p_j]_t$ to P_h for all $j \in [N]$.

Next we show that we can use Π_{Rand} to instantiate $\mathcal{F}_{\mathsf{RandRobust}}$. Consider a secret sharing scheme Σ which takes a parity element p and outputs a Shamir sharing of p in $\mathcal{R}$.

It is shown in [PS21] that the following properties hold for Σ:

– Σ is an $\mathbb{Z}/2^k\mathbb{Z}$-GLSSS.
– Given a set $A \subset \mathcal{I}$ and a set of shares $\{a_i\}_{i\in A}$ for parties in A, there exists an efficient algorithm which outputs either $\Sigma(A,(a_i)_{i\in A}) = \emptyset$ or a random sharing $[p]_t \in \Sigma(A,(a_i)_{i\in A})$.

Therefore, $\mathcal{F}_{\mathsf{RandParity}}$ can be instantiated by $\mathcal{F}_{\mathsf{RandRobust}}$ with the secret sharing scheme Σ defined above. Note that the share size of Σ is $\mathsf{sh} = m$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. It follows that when using $\Pi_{\mathsf{Rand}}(N)$ to instantiate $\mathcal{F}_{\mathsf{RandRobust}}$ the total communication complexity to generate N random sharings in Σ is $O(N\cdot n\cdot m + n^2\cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Random Additive Beaver Triples. The functionality $\mathcal{F}_{\mathsf{TripleAdd}}$ that prepares random additive Beaver triples is shown in Functionality 14.

Functionality 14: $\mathcal{F}_{\mathsf{TripleAdd}}(N)$

1. $\mathcal{F}_{\mathsf{TripleAdd}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S\cap\mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{TripleAdd}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{TripleAdd}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{TripleAdd}}$ receives from the adversary the shares of active corrupted parties $(u_i^{(j)}, v_i^{(j)}, w_i^{(j)})_{i\in\mathcal{C}_{\mathtt{active}}}$ where $u_i^{(j)}, v_i^{(j)}, w_i^{(j)} \in \mathcal{R}$ and $w_i^{(j)} = u_i^{(j)} + v_i^{(j)}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{TripleAdd}}$ randomly samples N random triples $(\boldsymbol{a}_1,\boldsymbol{b}_1,\boldsymbol{c}_1),\ldots,(\boldsymbol{a}_N,\boldsymbol{b}_N,\boldsymbol{c}_N)$, where each triple entry is in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ and each triple $(\boldsymbol{a}_j,\boldsymbol{b}_j,\boldsymbol{c}_j)$ satisfies $\boldsymbol{a}_j+\boldsymbol{b}_j=\boldsymbol{c}_j$. Then $\mathcal{F}_{\mathsf{TripleAdd}}$ generates $3N$ degree-t Shamir sharing $([\phi(\boldsymbol{a}_j)]_t,[\phi(\boldsymbol{b}_j)]_t,[\phi(\boldsymbol{c}_j)]_t)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(u_i^{(j)}, v_i^{(j)}, w_i^{(j)})_{j=1}^N$. For each $P_i \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{TripleAdd}}$ sends the i-th share of the sharings in the triple $([\phi(\boldsymbol{a}_j)]_t,[\phi(\boldsymbol{b}_j)]_t,[\phi(\boldsymbol{c}_j)]_t)$ to P_i for all $j \in [N]$.

The protocol $\Pi_{\mathsf{TripleAdd}}$ realizes $\mathcal{F}_{\mathsf{TripleAdd}}$. The description of $\Pi_{\mathsf{TripleAdd}}$ appears in Protocol 9.

Protocol 9: $\Pi_{\mathsf{TripleAdd}}(N)$

1. All parties invoke $\mathcal{F}_{\mathsf{RandShamir}}(2N)$ to prepare $2N$ regular degree-t Shamir secret sharings in the form of $([\phi(\boldsymbol{a}_i)]_t, [\phi(\boldsymbol{b}_i)]_t)_{i=1}^N$. All parties also receive a set of eliminated parties denoted by S. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$.
2. All parties locally compute $[\phi(\boldsymbol{c}_i)]_t = [\phi(\boldsymbol{a}_i)]_t + [\phi(\boldsymbol{b}_i)]_t$ fro all $i \in [N]$. All parties output S and $([\phi(\boldsymbol{a}_i)]_t, [\phi(\boldsymbol{b}_i)]_t, [\phi(\boldsymbol{c}_i)]_t)_{i=1}^N$.

The communication complexity of calling $\mathcal{F}_{\mathsf{RandShamir}}(2N)$ is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the communication complexity of $\Pi_{\mathsf{TripleAdd}}(N)$ is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 8. *The protocol $\Pi_{\mathsf{TripleAdd}}$ computes $\mathcal{F}_{\mathsf{TripleAdd}}$ with perfect security in the $\mathcal{F}_{\mathsf{RandShamir}}$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 1, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{RandShamir}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

Random Multiplicative Beaver Triples. The description of $\mathcal{F}_{\mathsf{TripleMult}}$ appears in Functionality 15.

Functionality 15: $\mathcal{F}_{\mathsf{TripleMult}}(N)$

1. $\mathcal{F}_{\mathsf{TripleMult}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{TripleMult}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{TripleMult}}$ then sends S to all active honest parties.
2. $\mathcal{F}_{\mathsf{TripleMult}}$ receives from the adversary the shares of active corrupted parties $(u_i^{(j)}, v_i^{(j)}, w_i^{(j)})_{i \in \mathcal{C}_{\mathtt{active}}}$ where $u_i^{(j)}, v_i^{(j)}, w_i^{(j)} \in \mathcal{R}$ for all $i \in \mathcal{C}_{\mathtt{active}}$ and for all $j \in [N]$.
3. $\mathcal{F}_{\mathsf{TripleMult}}$ randomly samples N random triples $(\boldsymbol{a}_1, \boldsymbol{b}_1, \boldsymbol{c}_1), \ldots, (\boldsymbol{a}_N, \boldsymbol{b}_N, \boldsymbol{c}_N)$, where each triple entry is in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ and each triple $(\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{c}_j)$ satisfies $\boldsymbol{a}_j \star \boldsymbol{b}_j = \boldsymbol{c}_j$. Then $\mathcal{F}_{\mathsf{TripleMult}}$ generates $3N$ degree-t Shamir sharing $([\phi(\boldsymbol{a}_j)]_t, [\phi(\boldsymbol{b}_j)]_t, [\phi(\boldsymbol{c}_j)]_t)_{j=1}^N$ such that the shares of $P_i \in \mathcal{C}_{\mathtt{active}}$ are $(u_i^{(j)}, v_i^{(j)}, w_i^{(j)})_{j=1}^N$. For each $P_h \in \mathcal{H}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{TripleMult}}$ sends the ℓ-th share of the sharings in the triple $([\phi(\boldsymbol{a}_j)]_t, [\phi(\boldsymbol{b}_j)]_t, [\phi(\boldsymbol{c}_j)]_t)$ to P_h for all $j \in [N]$.

The protocol $\Pi_{\mathsf{TripleMult}}$ realizes $\mathcal{F}_{\mathsf{TripleMult}}$. The description of $\Pi_{\mathsf{TripleMult}}$ appears in Protocol 10.

Protocol 10: $\Pi_{\mathsf{TripleMult}}(N)$

1. All parties invoke $\mathcal{F}_{\mathsf{RandShamir}}(2N)$ to prepare $2N$ regular degree-t Shamir secret sharings in the form of $([\phi(\boldsymbol{a}_i)]_t, [\phi(\boldsymbol{b}_i)]_t)_{i=1}^N$. All parties also receive a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
2. All parties invoke $\mathcal{F}_{\mathsf{Mult}}(N)$ with input $([\phi(\boldsymbol{a}_i)]_t)_{i=1}^N$ and $([\phi(\boldsymbol{b}_i)]_t)_{i=1}^N$. They get the output $([z_i]_t)_{i=1}^N$ and a set of eliminated parties denoted by S_2. All parties set $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$. We have for all $i \in [N]$, $z_i = \phi(\boldsymbol{a}_i) \cdot \phi(\boldsymbol{b}_i)$.
3. All parites invoke $\mathcal{F}_{\mathsf{ReEncode}}(N)$ with inputs $([z_i]_t)_{i=1}^N$. They get the output $([\phi \circ \psi(z_i)]_t)_{i=1}^N$ and a set of eliminated parties denoted by S_3. All parties set $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_3$. Let $\boldsymbol{c}_i$ denote $\psi(z_i)$ for all $i \in [N]$. All parties take $S := S_1 \cup S_2 \cup S_3$ and $([\phi(\boldsymbol{a}_i)]_t, [\phi(\boldsymbol{b}_i)]_t, [\phi(\boldsymbol{c}_i)]_t)_{i=1}^N$ as output.

The communication complexity of calling $\mathcal{F}_{\mathsf{RandShamir}}(2N)$ is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. The communication complexity of the instantiation of $\mathcal{F}_{\mathsf{Mult}}(N)$ is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$, and the communication complexity of the instantiation of $\mathcal{F}_{\mathsf{ReEncode}}(N)$ is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the communication complexity of $\Pi_{\mathsf{TripleMult}}(N)$ is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 9. *The protocol $\Pi_{\mathsf{TripleMult}}$ computes $\mathcal{F}_{\mathsf{TripleMult}}$ with perfect security in the ($\mathcal{F}_{\mathsf{RandShamir}}$, $\mathcal{F}_{\mathsf{Mult}}$, $\mathcal{F}_{\mathsf{ReEncode}}$)-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 1, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{RandShamir}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

C Extended Additive Secret Sharing

Let (ϕ, ψ) be an $(\ell, m)_2$-RMFE. Recall that n denotes the number of parties and $\phi : (\mathbb{Z}/2^k\mathbb{Z})^\ell \to \mathcal{R}$ is an $\mathbb{Z}/2^k\mathbb{Z}$-linear map. Also, $|\mathcal{R}| = 2^m \geq 2n + 1$. Therefore, the Shamir secret sharing scheme is well-defined in $\mathcal{R}$. In our construction, we will use ϕ to encode a vector of secrets $\boldsymbol{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(\ell)}) \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$. All parties will hold a degree-t Shamir sharing of $\phi(\boldsymbol{x})$, denoted by $[\phi(\boldsymbol{x})]_t$.

Defining Additive Sharings and Extended Additive Sharings. For $x \in \mathbb{Z}/2^k\mathbb{Z}$, we use $\langle x \rangle$ to denote an *additive sharing* of x among the first $t+1$ parties in $\mathbb{Z}/2^k\mathbb{Z}$. Specifically, the additive sharing of x is $\langle x \rangle = (x_1, \ldots, x_{n'})$ where party P_i holds the share $x_i \in \mathbb{Z}/2^k\mathbb{Z}$ such that $\sum_{i=1}^{t+1} x_i = x$ and $x_{t+2}, \ldots, x_{n'}$ are all 0.

Recall that $\psi : \mathcal{R} \to (\mathbb{Z}/2^k\mathbb{Z})^\ell$ and $\mathsf{val}(\cdot) : \mathcal{R} \to \mathbb{Z}/2^k\mathbb{Z}$ are both $\mathbb{Z}/2^k\mathbb{Z}$-linear. We define that a sharing $(\!|x|\!)$ is an *extended additive sharing* of $x \in \mathbb{Z}/2^k\mathbb{Z}$ if $(\!|x|\!)$'s shares form a degree-t shamir sharing $[y]_t$ in $\mathcal{R}$. The secret y satisfies $\mathsf{val}(y) = x$. We write $(\!|x|\!) := [y]_t$. It is clear that extended additive sharings are additive.

Generating Extended Additive Sharings. We generate generate the extended additive sharing of $x \in \mathbb{Z}/2^k\mathbb{Z}$ from the Shamir sharing $[\phi(\boldsymbol{z})]_t$. Suppose the j-th element of $\boldsymbol{z}$ is x. By the property of RMFE, we have $\psi(\phi(\boldsymbol{e}_j) \cdot \phi(\boldsymbol{z})) = \boldsymbol{e}_j \cdot \boldsymbol{z}$. Therefore, $\mathsf{val}(\phi(\boldsymbol{e}_j) \cdot \phi(\boldsymbol{z})) = x$. To obtain $(\!|x|\!)$, all parties locally compute $(\!|x|\!) = \phi(\boldsymbol{e}_j) \cdot [\boldsymbol{z}]_t$.

Converting Extended Additive Sharings to Additive Sharings. Now we show how to obtain $\langle x \rangle$ from $(\!|x|\!)$. We suppose that $(\!|x|\!)$ is equivalent to the Shamir sharing $[y]_t$.

Recall that the degree-t Shamir sharing $[y]_t$ corresponds to a degree-t polynomial $f(\cdot) \in \mathcal{R}[X]$ such that $f(\alpha_i)$ is the share of the i-th party P_i and $f(0) = y$, where $\alpha_1, \ldots, \alpha_n$ are distinct non-zero elements in $\mathcal{R}$. In particular, relying on Lagrange interpolation, $f(0)$ can be written as a linear combination of the first $t+1$ shares. For $i \in [t+1]$, let $c_i = \prod_{j \neq i, j \in [t+1]} \frac{\alpha_j}{\alpha_j - \alpha_i}$. We have $f(0) = \sum_{i=1}^{t+1} c_i f(\alpha_i)$. Therefore, the Shamir sharing $[y]_t$ can be locally converted to an additive sharing of x among the first $t+1$ parties by letting $P_i (i \in [t+1])$ taking $\mathsf{val}(c_i \cdot f(\alpha_i))$ as its share, while for $i \notin [t+1]$ the party P_i takes 0 as its share. Note that

$$\sum_{i=1}^{t+1} \mathsf{val}(c_i \cdot f(\alpha_i)) = \mathsf{val}(\sum_{i=1}^{t+1} c_i \cdot f(\alpha_i)) = \mathsf{val}(y) = x,$$

this is indeed an additive sharing of x.

D Segment Verification Protocol

Protocol 11: $\Pi_{\mathsf{Verify}}(\mathtt{seg}, S_0)$

1. Suppose $\mathtt{seg}$ has N gate groups. For the i-th gate group within $\mathtt{seg}$ that corresponds to the Beaver triple $([\phi(\boldsymbol{a}^{(i)})]_t, [\phi(\boldsymbol{b}^{(i)})]_t, [\phi(\boldsymbol{c}^{(i)})]_t)$ and has inputs $\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(i)} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ All parties locally computes $\phi(\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)})$ for all $i \in [N]$.
2. All parties call $\mathcal{F}_{\mathsf{VerifyBC}}$ with inputs $\{\phi(\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)})\}_{i \in [N]} \cup \{\phi(\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)})\}_{i \in [N]}$ and D. If the result is a semi-honest pair $\{P_{j_1}, P_{j_2}\}$, all parties take it as output and halt. Otherwise, run the following steps.
3. All parties locally compute $[\phi(\boldsymbol{x}^{(i)})]_t = \phi(\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}) - [\phi(\boldsymbol{a}^{(i)})]_t$ and $[\phi(\boldsymbol{y}^{(i)})]_t = \phi(\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}) - [\phi(\boldsymbol{b}^{(i)})]_t$ for all $i \in [N]$.

4. For each additive gate group with input $[\phi(\boldsymbol{x}^{(i)})]_t$ and $[\phi(\boldsymbol{y}^{(i)})]_t$, all parties locally compute the output sharing
$$[\phi(\boldsymbol{z}^{(i)})]_t = [\phi(\boldsymbol{x}^{(i)})]_t + [\phi(\boldsymbol{y}^{(i)})]_t.$$
For all the multiplicative gate groups, suppose their indices form the set I_{mult}. All parties call $\mathcal{F}_{\mathsf{Mult}}$ with input $([\phi(\boldsymbol{x}^{(i)})]_t)_{i\in I_{\text{mult}}}$ and $([\phi(\boldsymbol{y}^{(i)})]_t)_{i\in I_{\text{mult}}}$, and get output $([w^{(i)}]_t)_{i\in I_{\text{mult}}}$ and a set of eliminated parties denoted by S_1. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_1$.
Then all the parties call the functionality $\mathcal{F}_{\mathsf{ReEncode}}$ with input $([w^{(i)}]_t)_{i\in I_{\text{mult}}}$ and get the output sharings $([\phi(\boldsymbol{z}^{(i)})]_t)_{i\in I_{\text{mult}}}$ and a set of eliminated parties denoted by S_2. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_2$.
5. All parties run the protocol $\Pi_{\mathsf{NetworkRouting}}(\mathtt{seg})$. Note that during fan-out the j-th wire of $[\phi(\boldsymbol{z}^{(i)})]_t$ is copied $\tilde{n}_j^{(i)}$ times. All parties get all Shamir sharings of $\mathtt{seg}$'s gate group inputs, denoted by $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t\}_{i\in[N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t\}_{i\in[N]}$. All parties also receive a set of eliminated parties denoted by S_3. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_3$.
6. All parties call $\mathcal{F}_{\mathsf{OpenPub}}$ to reconstruct the Shamir sharings $\{[\phi(\tilde{\boldsymbol{x}}^{(i)})]_t + [\phi(\boldsymbol{a}^{(i)})]_t\}_{i\in[N]}$ and $\{[\phi(\tilde{\boldsymbol{y}}^{(i)})]_t + [\phi(\boldsymbol{b}^{(i)})]_t\}_{i\in[N]}$, and get $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ for all $i \in [N]$.
7. Each party locally compares $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ with $\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}$ and compares $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ with $\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}$ for all $i \in [N]$. Based on whether there are any differences, all parties do the following:
   (a) If for all $i \in [N]$, $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)} = \boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)} = \boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}$, all parties call $\mathcal{F}_{\mathsf{FanOut}}$ with the output sharings $\{[\phi(\boldsymbol{z}^{(i)})]_t\}_{i\in[N]}$ and $\{(n_j^{(i)} - \tilde{n}_j^{(i)})\}_{i\in[N],j\in[\ell]}$, where the j-th wire of $[\phi(\boldsymbol{z}^{(i)})]_t$ is copied $n_j^{(i)} - \tilde{n}_j^{(i)}$ times. All parties receive the fan-out sharings, and a set of eliminated parties denoted by S_4. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_4$.
   Then all parties run $\mathcal{F}_{\mathsf{Permute}}$ with the fan-out sharings and the desired permutations as input. All parties receive the permuted fan-out sharings, and a set of eliminated parties denoted by S_5. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_5$.
   All parties output $\mathtt{correct}$.
   (b) Otherwise, let $S := S_0 \cup S_1 \cup S_2 \cup S_3$. If $S \neq \emptyset$, all parties output S and $\mathtt{incorrect}$ and halts. Otherwise, all parties can select a wire with an inconsistent value that has the smallest topological order, denoted by $x_{j_0}^{(i_0)} + a_{j_0}^{(i_0)}$ or $y_{j_0}^{(i_0)} + b_{j_0}^{(i_0)}$. Let $\langle s\rangle + \langle o\rangle$ denote its corresponding additive sharing for reconstruction in the protocol Π_{Eval}, and let $(\!|s|\!)$ denote its extended additive sharing. Then all parties run the protocol $\Pi_{\mathsf{FaultDetection}}(D, \langle s\rangle + \langle o\rangle, (\!|s|\!))$, and get a set of eliminated parties S_4 as output. All parties update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S_4$, and output $S \cup S_4$ and $\mathtt{incorrect}$.

Lemma 10 (Correctness of Π_{Verify}). *In Π_{Verify}, either all active honest parties have got the correct output Shamir sharings, or all parties output a set of eliminated parties where at least half of the parties in the set are corrupted.*

The proof of Lemma 10 can be found in Section F.2.

Cost of Π_{Verify}. The communication complexity of Step 2 for calling $\mathcal{F}_{\mathsf{VerifyBC}}$ is $O(N\cdot n\cdot m+n^2\cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. The communication complexity of Step 4 for calling $\mathcal{F}_{\mathsf{Mult}}$ and $\mathcal{F}_{\mathsf{ReEncode}}$ is bounded by $O(N\cdot n\cdot m+n^3\cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. The communication complexity of Step 5 for running $\Pi_{\mathsf{NetworkRouting}}$ is $O(N\cdot n\cdot m+n^2\cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$, since all parties generate $\sum_{i\in[N],j\in[\ell]}\tilde{n}_j^{(i)}/\ell \le 3N$ fan-out sharings in this step. The communication complexity of Step 6 for calling $\mathcal{F}_{\mathsf{OpenPub}}$ is $O(N\cdot n\cdot m+n^2\cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Note that $N = O(n^2\cdot m)$ according to Theorem 4. Therefore, the total communication complexity before Step 7 is bounded by $O(n^3\cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

For Step 7, will discuss the cost of Π_{Verify} in two different scenarios based on whether the verification passes.

– If the verification passes, all parties will run $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ that output
$$M_{\mathtt{seg}} := \frac{\sum_{i\in[N],j\in[\ell]} n_j^{(i)} - \tilde{n}_j^{(i)}}{\ell}$$
Shamir sharings. Then the communication complexity of Step 7.(a) is $O(M_{\mathtt{seg}}\cdot n\cdot m+n^3\cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the total communication complexity in this case is $O(M_{\mathtt{seg}}\cdot n\cdot m+n^3\cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

– If the verification does not pass, all parties will run $\Pi_{\mathsf{FaultDetection}}$. And the communication complexity of running $\Pi_{\mathsf{FaultDetection}}$ is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the total communication complexity in this case is $O(n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

E Details on Network Routing

E.1 Circuit Transformation

The following theorem shows that we can safely assume that the circuit at hand satisfies the grouping properties for network routing.

Theorem 4 (Circuit Transformation). *Given an arithmetic circuit C over $\mathbb{Z}/2^k\mathbb{Z}$ with input coming from c clients, there exists an efficient algorithm which takes C as input and outputs an arithmetic circuit C' over $\mathbb{Z}/2^k\mathbb{Z}$ with the following properties:*

– *For all input x, $C(x) = C'(x)$.*
– *C' satisfies the Circuit Segment Conditions, the Gate Number Conditions, and the Gate Grouping Conditions which are stated previously.*
– *Circuit Size: $|C'| = O(|C| + \ell \cdot c)$, where c is the number of clients that provide inputs.*

Proof. Let $\mathsf{Client}_1, \mathsf{Client}_2, \ldots, \mathsf{Client}_c$ denote the clients that provide inputs. We show the algorithm to transform the circuit C' from C as follows:

– **Step 1:** We start by creating a virtual client Client_0. We will insert two new types of gates: input gates belonging to Client_0 and output gates belonging to Client_0. The input gates belonging to Client_0 are used to provide constant values for the computation. The output gates belonging to Client_0 are used to collect the wires that will not be output to any clients. Initially, for each input wire carrying a constant value in C, we insert an input gate belonging to Client_0 for this wire. Without loss of generality, we assume that each input of Client_0 is only used once in the circuit. To see this, an input of Client_0 is known to all parties, and if this input is used more than once, we can simply insert multiple input gates that take the same value. Note that the same argument does not work for other clients since it allows a client to use different values for its input wires in C' which should have the same value in C.
This step does not affect the circuit size.
– **Step 2:** Order all the addition gates and multiplication gates in C according to their topological ordering. We divide $n \cdot \ell$ consecutive gates into one circuit segment. For each circuit segment, let M_i and A_i denote the number of multiplication gates and the number of addition gates in it. We insert $\lceil \frac{M_i}{\ell} \rceil \cdot \ell - M_i$ dummy multiplication gates and $\lceil \frac{A_i}{\ell} \rceil \cdot \ell - A_i$ dummy addition gates in this circuit segment. Each of these new dummy gates takes two inputs from Client_0 and outputs to Client_0 as well. The inputs are set to be 0. In this way, we insert 2 input gates belonging to Client_0 and one output gate belonging to Client_0 for each of the new dummy multiplication gates and dummy addition gates. Then we organize ℓ consecutive addition gates into one addition gate group, and organize ℓ consecutive multiplication gates into one multiplication gate group. After this step, the circuit is divided into circuit segments where each circuit segment has size $\Theta(n^2 \cdot m^2)$ except that the last circuit segment has size $O(n^2 \cdot m^2)$. Also, the multiplication gates and the addition gates within each circuit are organized into gate groups of size ℓ. Note that this step increases the circuit size by $O(|C|/n)$.
– **Step 3:** For each client Client_i ($i = 0, 1, \ldots, c$), let I_i denote the number of input gates belonging to Client_i. Then we insert $\lceil \frac{I_i}{\ell} \rceil \cdot \ell - I_i$ dummy input gates belonging to Client_i, which will take 0 as input. We also insert the same number of dummy output gates belonging to Client_0, where each dummy output gate directly takes the input of one dummy input gate as output. Then we divide the input gates belonging to Client_i into groups of ℓ, and Client_i has $\lceil \frac{I_i}{\ell} \rceil$ input gate groups.
After this step, the number of input gates belonging to each Client_i is a multiple of ℓ. Note that this step increases the circuit size by $O(\ell \cdot c)$.
– **Step 4:** For each gate group (i.e. input gate group, multiplication gate group, or addition gate group), let $(w_1, w_2, \ldots, w_\ell)$ denote the output wires, and $(n_1, n_2, \ldots, n_\ell)$ denote the number of times that each output wire is used. Note that an output wire in a multiplication or addition

gate group can be the input wire of some gate in the same gate group because we do not require that the gates in a multiplication or addition gate group should belong to the same circuit layer. Let $n_0 = \lceil \frac{\sum_{i=1}^{\ell} n_i}{\ell} \rceil \cdot \ell - \sum_{i=1}^{\ell} n_i$. We insert n_0 dummy output gates belonging to Client_0, and these output gates take w_1 as the input wire.
After this step, the number of times that $w_1, w_2, \ldots, w_\ell$ are used as input wires in other gates is a multiple of ℓ. Note that each wire w_j is used at least once, so this step increases the circuit size by at most a factor of 2.

– **Step 5:** For each client Client_i $(i = 0, 1, \ldots, c)$, let O_i denote the number of output gates belonging to Client_i. We insert $\lceil \frac{O_i}{\ell} \rceil \cdot \ell - O_i$ input gates belonging to Client_0, which take 0 as input. We also insert the same number of output gates belonging to Client_i which take these inputs as output. Then we divide the output gates belonging to each client Client_i into gate groups of size ℓ, and we also divide the added input gates for Client_0 into gate groups of size ℓ. After this step, the number of output gates belonging to each Client_i is a multiple of ℓ. Also, after this step, the number of input gates belonging to Client_0 remains a multiple of ℓ. To see this, recall that after Step 2 and Step 3, the total number of output wires of all clients is a multiple of ℓ, and the number of output wires of Client_0 is a multiple of ℓ. In addition, since all the added dummy input gates for Client_0 in this step will be used only once, the number of times that the output wires of each gate group formed by these input gates are used is a multiple of ℓ. Note that this step increases the circuit size by $O(\ell \cdot c)$.

Next, we show that the circuit C' after the transformation has the desired properties:

– First we show $C(x) = C'(x)$ for all input x. Note that we do not change the original gates and wires in C. Therefore, it is sufficient to show that the gates and wires we add in the transformation do not change the functionality. For the input layer, we create several new input gates belonging to each Client_i $(i \geq 1)$. These new gates directly connect to the output gates of Client_0, which means that these values are never used in the computation. For the output layer, we create several new output gates belonging to each Client_i $(i \geq 1)$. These new gates directly take the value 0 from the input gates of Client_0, which do not affect the final result. Therefore, $C(x) = C'(x)$ for all input x.
– Next we show C' satisfies the Circuit Segment Conditions, the Gate Number Conditions, and the Gate Grouping Conditions.
  • After Step 2, the size of each segment except the last segment is $\Theta(n^2 \cdot m^2)$, and the size of the last segment is $O(n^2 \cdot m^2)$. Also, we divide the segment so that each segment contains gates of consecutive topological order. Therefore, C' satisfies the Circuit Segment Conditions.
  • After Step 2, the number of addition gates and the number of multiplication gates within each circuit segment are multiples of ℓ. After Step 1, Step 3, and Step 5, the number of input gates and the number of output gates belonging to each client are multiples of ℓ. Therefore, C' satisfies the Gate Number Conditions.
  • After Step 2, the addition gates and the multiplication gates within each segment are organized into gate groups of size ℓ. After Step 3 and Step 5, the input gates and the output gates belonging to each client are organized into gate groups of size ℓ. Also, after Step 4, for the output wires of each multiplication gate group and of each addition gate group, the number of times that those wires are used as input wires in other gates is a multiple of ℓ. After Step 4 and Step 5, for the output wires of each input gate group, the number of times that those wires are used as input wires in other gates is a multiple of ℓ. Therefore, C' satisfies the Gate Grouping Conditions.
– Step 1 will not affect the size of the circuit. In Step 2 and Step 4, the circuit size increases by at most a factor of 2. In Step 3 and Step 5, the circuit size increases by $O(\ell \cdot c)$. Therefore, the size of C' after the transformation is bounded by $O(|C| + \ell \cdot c)$.

E.2 Preparing Random Sharings for Sharing Transformations

Definition of Ideal Functionality. Let $r = \lfloor \frac{n'-2t'+1}{2} \rfloor$. Let $\Pi_1, \Pi_2, \ldots, \Pi_{\ell \cdot r}$ be $\ell \cdot r$ arbitrary $\mathbb{Z}/2^k\mathbb{Z}$-arithmetic secret sharing schemes, where each Π_i is defined as follows:

– The secret space $Z_i = (\mathbb{Z}/2^k\mathbb{Z})^\ell$.

– The share space $U_i = (\mathcal{R})^2$, which can be viewed as $(\mathbb{Z}/2^k\mathbb{Z})^{2m}$.
– The sharing space $C_i = (\mathcal{R})^{2n'}$
– For a secret $\boldsymbol{x} \in Z_i$, the sharing of $\boldsymbol{x}$ is $([\phi(\boldsymbol{x})]_t, [\phi(L_i(\boldsymbol{x}))]_t)$, where L_i is a linear mapping from the vector space $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ to the vector space $(\mathbb{Z}/2^k\mathbb{Z})^\ell$.
– For a sharing $([\phi(\boldsymbol{x})]_t, [\phi(L_i(\boldsymbol{x}))]_t)$, we reconstruct the secret $\boldsymbol{x}$ using $[\phi(\boldsymbol{x})]_t$.
– The $\mathtt{share}_{\Pi_i}$ can be viewed as $\mathtt{share}_{\Pi_i} : Z \times (\mathbb{Z}/2^k\mathbb{Z})^{2t\cdot m} \to C$.

The functionality $\mathcal{F}_{\mathsf{RandSharTrans}}$ presented in Functionality 16 generates a random sharing in an $\mathbb{Z}/2^k\mathbb{Z}$-arithmetic secret sharing scheme $\Pi_i = (n', Z_i, U_i, C_i, \mathtt{share}_{\Pi_i}, \mathtt{rec}_{\Pi_i})$ for all $i \in [\ell \cdot r]$.

Functionality 16: $\mathcal{F}_{\mathsf{RandSharTrans}}(N)$

1. $\mathcal{F}_{\mathsf{RandSharTrans}}$ receives from the adversary one of the following:
   – The shares of active corrupted parties $(\boldsymbol{u}_j^{(i)})_{j\in\mathcal{C}_{\mathtt{active}}}$ where $\Pi_i(\mathcal{C}_{\mathtt{active}}, (u_j^{(i)})_{j\in\mathcal{C}_{\mathtt{active}}}) \neq \emptyset$ for all $i \in [\ell \cdot r]$;
   – A semi-corrupted pair $\{P_{j_1}, P_{j_2}\}$ where $\{P_{j_1}, P_{j_2}\} \cap \mathcal{C}_{\mathtt{active}} \neq \emptyset$.
2. Based on what it has received, $\mathcal{F}_{\mathsf{RandSharTrans}}$ does one of the following:
   – On receiving $(\boldsymbol{u}_j^{(i)})_{j\in\mathcal{C}_{\mathtt{active}}}$ for all $i \in [\ell \cdot r]$, $\mathcal{F}_{\mathsf{RandSharTrans}}$ uniformly randomly samples a random Π_i-sharing $[\![\boldsymbol{x}]\!] \in \Pi(\mathcal{C}_{\mathtt{active}}, (\boldsymbol{u}_j^{(i)})_{j\in\mathcal{C}_{\mathtt{active}}})$ for all $i \in [\ell \cdot r]$. For each honest party $P_h \in \mathcal{P}_{\mathtt{active}}$, $\mathcal{F}_{\mathsf{RandSharTrans}}$ sends the ℓ-th share of $[\![\boldsymbol{x}]\!]$ to P_h for all $i \in [\ell \cdot r]$.
   – On receiving $\{P_{j_1}, P_{j_2}\}$, $\mathcal{F}_{\mathsf{RandSharTrans}}$ sends $\{P_{j_1}, P_{j_2}\}$ to all active honest parties.

Protocol Description. For $\boldsymbol{x} \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r)$ where each $\boldsymbol{x}_i$ is a vector in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$, let $\Phi(\boldsymbol{x})$ denote the vector $(\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_r))$. For $\boldsymbol{y} \in \mathcal{R}^r = (y_1, \ldots, y_r)$ where each y_i is an element in $\mathcal{R}$, let $\Psi(\boldsymbol{y})$ denote the vector $(\psi(y_1), \ldots, \psi(y_r))$.

We have each secret in the secret space needs ℓ elements of randomness in $\mathbb{Z}/2^k\mathbb{Z}$, and each random tape needs $2t \cdot m$ elements of randomness in $\mathbb{Z}/2^k\mathbb{Z}$. Following the idea in [GPS22], we can use random packed Shamir sharings over $\mathcal{R}$ to provide such randomness for the Π_i-sharings. Note that $2t \cdot m + \ell < 2n' \cdot m$, so for a packed Shamir secret sharing scheme whose secret space is in $\mathcal{R}^r$, we need to prepare $2n' \cdot m$ random sharings in this scheme to provide enough randomness for $\ell \cdot r$ random Π_i sharings.

In our protocol, we will use the following packed secret sharing scheme Σ over $\mathbb{Z}/2^k\mathbb{Z}$ [GPS22]:

– The secret space is $Z = (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$.
– The share space is $U = \mathcal{R}$, which can be viewed as $(\mathbb{Z}/2^k\mathbb{Z})^m$.
– For $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is computed as follows: First we divide $\boldsymbol{x}$ into r vectors of dimension ℓ. Then we map each vector to a ring element of $\mathcal{R}$ using $\phi(\cdot)$, i.e. we map $\boldsymbol{x}$ into $\Phi(\boldsymbol{x})$. Next, we use a degree-d' packed Shamir secret sharing scheme over $\mathcal{R}$ to store the r elements in $\mathcal{R}$.
– For a sharing $[\![\boldsymbol{x}]\!] \in \Sigma$, we first view it as a degree-d' packed Shamir secret sharing over $\mathcal{R}$ and reconstruct its secret $\boldsymbol{s} \in \mathcal{R}^r$. To recover the secret, we apply ψ, the inverse map of ϕ, to each element in $\boldsymbol{s}$.

Also, we define another secret sharing scheme Σ' over $\mathbb{Z}/2^k\mathbb{Z}$ as follows [GPS22]:

– The secret space is $Z' = (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$.
– The share space is $U' = \mathcal{R}$, which can be viewed as $(\mathbb{Z}/2^k\mathbb{Z})^m$.
– For $\boldsymbol{x} \in Z$, the sharing of $\boldsymbol{x}$ is computed as follows: First we divide $\boldsymbol{x}$ into r vectors of dimension ℓ. Then we map $\boldsymbol{x}$ into $\Phi(\boldsymbol{x})$ We sample a vector of elements $\boldsymbol{o} = (o_1, \ldots, o_r) \in \mathcal{R}^r$ where $\Psi(\boldsymbol{o}) = \boldsymbol{0} \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$. It has been shown in section 2.6 that an efficient sampling algorithm for $o_i \in \mathcal{R}$ such that $\psi(o_i) = \boldsymbol{0} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ exists. Next, we use a degree-d' packed Shamir secret sharing scheme over $\mathcal{R}$ to store the r elements of $\Phi(\boldsymbol{x}) + \boldsymbol{o}$ in $\mathcal{R}$.
– For a sharing $[\![\boldsymbol{x}]\!] \in \Sigma'$, we first view it as a degree-$(d' + r - 1)$ packed Shamir secret sharing over $\mathcal{R}$ and reconstruct its secret $\boldsymbol{s} \in \mathcal{R}^r$. To recover the secret, we apply $\Psi(\cdot)$ to $\boldsymbol{s}$ and get $\Psi(\boldsymbol{s})$.

We pick $d' = \lceil \frac{n'-1}{2} \rceil$ and $r = \lfloor \frac{n'-2t'+1}{2} \rfloor$, to guarantee that the degree-d' packed Shamir sharings can hold r secrets in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$ ($d' - r + 1 \geq t$) and that the degree-$(d' + r - 1)$ packed Shamir sharings can be reconstructed towards an honest party while detecting errors ($d' + r - 1 < n' - t'$ by Lemma 2). Therefore, Σ is a degree-$\lceil \frac{n'-1}{2} \rceil$ packed Shamir secret sharing scheme that holds

$r = \lfloor \frac{n'-2t'+1}{2} \rfloor$ secrets in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$, and Σ' is a degree-$(n'-t'-1)$ packed Shamir secret sharing scheme that holds $r = \lfloor \frac{n'-2t'+1}{2} \rfloor$ secrets in $(\mathbb{Z}/2^k\mathbb{Z})^\ell$.

In the following, we will use $[\Phi(\boldsymbol{x})]_{d'}$ to denote a Σ-sharing of the secret $\boldsymbol{x} \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$, and we will use $[\boldsymbol{y}]_{n'-t'-1}$ to denote a Σ'-sharing of $\boldsymbol{z} \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$ such that $\Psi(\boldsymbol{y}) = \boldsymbol{z}$. It has been shown in [GPS22] that Σ and Σ' satisfy that for all $\boldsymbol{c} \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}$

$$[\Phi(\boldsymbol{c}) \star \Phi(\boldsymbol{x})]_{n'-t'-1} = \Phi(\boldsymbol{c}) \star [\Phi(\boldsymbol{x})]_{d'}.$$

To reconstruct the secret $\boldsymbol{c} \star \boldsymbol{x}$ from $[\Phi(\boldsymbol{c}) \star \Phi(\boldsymbol{x})]_{n'-t'-1}$, we apply $\Psi(\cdot)$ to $\Phi(\boldsymbol{c}) \star \Phi(\boldsymbol{x})$ and get $\Psi(\Phi(\boldsymbol{c}) \star \Phi(\boldsymbol{x})) = \boldsymbol{c} \star \boldsymbol{x}$.

Our construction will use the ideal functionality $\mathcal{F}_{\mathsf{RandShPacked}}$ (Functionality 10) that prepare random Σ-sharings, and the ideal functionality $\mathcal{F}_{\mathsf{RandZeroSh}}$ (Functionality 12) that prepares random Σ'-sharings of $\mathbf{0} \in \mathcal{R}^r$.

The goal of the protocol is to prepare $\ell \cdot r$ random sharings $[\![\boldsymbol{x}_1]\!], \ldots, [\![\boldsymbol{x}_{\ell\cdot r}]\!]$ such that $[\![\boldsymbol{x}_i]\!]$ is a random Π_i-sharing, i.e. realizing $\mathcal{F}_{\mathsf{RandSharTrans}}(\Pi_1, \ldots, \Pi_{\ell\cdot r})$. The protocol that implements $\mathcal{F}_{\mathsf{RandSharTrans}}$ is shown in Protocol 12.

Protocol 12: $\Pi_{\mathsf{RandSharTrans}}(\Pi_1, \ldots, \Pi_{\ell\cdot r})$

1. All parties run the protocol Π_{Rand} to obtain $2n' \cdot m$ random Σ-sharings denoted by
$$[\Phi(\boldsymbol{u}_1)]_{d'}, \ldots, [\Phi(\boldsymbol{u}_{2n'\cdot m})]_{d'},$$
or get a semi-corrupted pair denoted by E_1. On receiving E_1, all parties take E_1 as output and halt. Otherwise, suppose $\boldsymbol{u}_i = (\boldsymbol{u}_i^{(1)}, \ldots, \boldsymbol{u}_i^{(r)})$ where $\boldsymbol{u}_i^{(j)} \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$ and denote the ℓ-th element of $\boldsymbol{u}_i^{(j)}$ by $u_i^{(j,\ell)}$. For all $i \in [r]$ and for all $q \in [\ell]$, let
$$\boldsymbol{\tau}_{(q-1)\cdot r+i} = (u_1^{(i,q)}, u_2^{(i,q)}, \ldots, u_\ell^{(i,q)}) \in (\mathbb{Z}/2^k\mathbb{Z})^{\ell\cdot\ell},$$
and
$$\boldsymbol{\rho}_{(q-1)\cdot r+i} = (u_{\ell+1}^{(i,q)}, u_{\ell+2}^{(i,q)}, \ldots, u_{\ell+\tilde{q}_i}^{(i,q)}) \in (\mathbb{Z}/2^k\mathbb{Z})^{2t\cdot m\cdot\ell}.$$
The goal of this protocol is to compute the Π_i-sharing $[\![\boldsymbol{x}_i]\!] = \mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$.
2. All parties run the protocol Π_{Rand} and obtain $2n' \cdot m$ random Σ'-sharings of $\mathbf{0} \in \mathcal{R}^r$, denoted by
$$\{[\boldsymbol{o}_j^{(1)}]_{n'-t'-1}, [\boldsymbol{o}_j^{(2)}]_{n'-t'-1}, \ldots, [\boldsymbol{o}_j^{(2m)}]_{n'-t'-1}\}_{j=1}^{n'},$$
or get a semi-corrupted pair denoted by E_2. On receiving E_2, all parties take E_2 as output and halt.
3. For all $i \in [r \cdot \ell]$, for all $j \in [n']$, and for all $p \in [2m]$, let $\mathcal{L}_j^{i,p} : Z_i \times (\mathbb{Z}/2^k\mathbb{Z})^{2t\cdot m} \to \mathbb{Z}/2^k\mathbb{Z}$ denote the $\mathbb{Z}/2^k\mathbb{Z}$ linear transformation such that for all $\boldsymbol{\tau} \in \tilde{Z}_i$ and $\boldsymbol{\rho} \in (\mathbb{Z}/2^k\mathbb{Z})^{2t\cdot m}$, $\mathcal{L}_j^{i,p}(\boldsymbol{\tau}, \boldsymbol{\rho})$ outputs the p-th element of the j-th share of the Π_i-sharing $\mathtt{share}_i(\boldsymbol{\tau}, \boldsymbol{\rho})$. Then there exist $c_{j,1}^{(i,p)}, \ldots, c_{j,\ell+2t\cdot m}^{(i,p)} \in \mathbb{Z}/2^k\mathbb{Z}$ such that
$$\mathcal{L}_j^{i,p}(\boldsymbol{\tau}, \boldsymbol{\rho}) = \sum_{w=1}^{\ell} c_{j,w}^{(i,p)} \cdot \tau_w + \sum_{w=1}^{2t\cdot m} c_{j,\ell\cdot\ell+w}^{(i,p)} \cdot \rho_w.$$
For all $j \in [n']$, for all $p \in [2m]$, and for all $w \in [2n' \cdot m]$, let
$$\boldsymbol{c}_{j,w}^{(\star,p)} = (c_{j,w}^{(1,p)}, c_{j,w}^{(2,p)}, \ldots, c_{j,w}^{(r\cdot\ell,p)}) \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell},$$
where $c_{j,w}^{(i,p)} = 0$ for all $w > \ell + 2t \cdot m$.
4. For all $i \in [r \cdot \ell]$, for all $j \in [n']$, and for all $p \in [2m]$, let $v_j^{(i,p)} = \mathcal{L}_j^{i,p}(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$. Let
$$\boldsymbol{v}_j^{(\star,p)} = (v_j^{(1,p)}, v_j^{(2,p)}, \ldots, v_j^{(r\cdot\ell,p)}) \in (\mathbb{Z}/2^k\mathbb{Z})^{r\cdot\ell}.$$
Then for all $j \in [n']$ and for all $p \in [2m]$, all parties locally compute the Σ'-sharing
$$[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1} = [\boldsymbol{o}_j^{(p)}]_{n'-t'-1} + \sum_{w=1}^{2n'\cdot m} \Phi(\boldsymbol{c}_{j,w}^{(\star,p)}) \star [\Phi(\boldsymbol{u}_w)]_{d'}.$$
Note that $\boldsymbol{s}_j^{(\star,p)} \in \mathcal{R}^r$, and $\Psi(\boldsymbol{s}_j^{(\star,p)}) = \boldsymbol{v}_j^{(\star,p)}$. Then all parties send their shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ to P_j.

5. For all $j \in [n']$ and for all $p \in [2m]$, P_j reconstructs the Σ' sharing $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ and gets $\boldsymbol{v}_j^{(\star,p)} = (\Psi(\boldsymbol{s}_j^{(1,p)}), \Psi(\boldsymbol{v}_j^{(2,p)}), \ldots, \Psi(\boldsymbol{v}_j^{(r,p)}))$, or P_j detects error in the sharing $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ and gets `unhappy`.
6. **Fault Detection Phase**:
   (a) All players broadcast their happy-bits using the byzantine agreement protocol.
   (b) For each party, if it receives at least one `unhappy` then sets his or her happy-bit to `unhappy`.
   (c) All players run a consensus protocol on their happy-bits. If the consensus is `happy`, all parties output the output of π and terminate the procedure. Otherwise, they proceed to the following steps.
7. **Fault Localization Phase**:
   (a) All players agree the party with the smallest index in $\mathcal{P}_A$ as the dealer D. All other players send all their generated values and communication to D.
   (b) On receiving all the information, D simulates π and the fault detection phase itself. D either prepares the message $(P_i, \texttt{corrupt})$ if P_i failed to follow the protocol, or the message $(P_{j_1}, P_{j_2}, \texttt{idx}, \texttt{b}, \texttt{b}', \texttt{disputed})$ if the `idx`-th bit `b` that P_{j_1} sends to P_{j_2} is inconsistent with P_{j_2}'s bit `b`$'$ that P_{j_2} claims to have received. Then D broadcasts the prepared message to all players.
   (c) If $(P_i, \texttt{corrupt})$ is received, all players set the eliminating set $E = \{D, P_i\}$.
   Otherwise, if $(P_{j_1}, P_{j_2}, \texttt{idx}, \texttt{b}, \texttt{b}', \texttt{disputed})$ is received, P_{j_1} and P_{j_2} will broadcast if they agree with this message. If P_{j_1} does not agree, all players set the eliminating set $E = \{D, P_{j_1}\}$; otherwise if P_{j_2} does not agree, all players set the eliminating set $E = \{D, P_{j_2}\}$; otherwise all players set the eliminating set $E = \{P_{j_1}, P_{j_2}\}$.
   (d) All players update $\mathcal{P}_{\texttt{active}} := \mathcal{P}_{\texttt{active}} - E$ and halt.
8. For all $i \in [r \cdot \ell]$, P_j sets its share of the Π_i sharing $[\![\boldsymbol{x}_i]\!]$ to be $\boldsymbol{v}_j^{(i)} = (v_j^{(i,1)}, v_j^{(i,2)}, \ldots, v_j^{(i,2m)})$. All parties take $[\![\boldsymbol{x}_1]\!], [\![\boldsymbol{x}_2]\!], \ldots, [\![\boldsymbol{x}_{r\cdot\ell}]\!]$ as output.

Cost of Protocol 12. The communication complexity of running Π_{Rand} in Step 2 and Step 3 are both $O(n^2 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Step 5 has communication complexity of $O(n^2 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the total communication complexity of Protocol 12 is $O(n^2 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 11. *In Step 5 of Protocol 12, either at least one active honest party has* `unhappy` *happy-bit or the shares of the output sharings* $\{[\![\boldsymbol{x}_i]\!]\}_{i=1}^r$ *held by active honest parties are valid and consistent.*

Proof. If no honest party get `unhappy`, by Lemma 2 all the Shamir sharings $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ are consistent for $j \in \mathcal{H}_{\texttt{active}}$. This indicates that no corrupted parties have deviated from the protocol in Step 4 by sending wrong sharings, so all the parties have acted faithfully. Therefore, the output sharings $\{[\![\boldsymbol{x}_i]\!]\}_{i=1}^r$ held by active honest parties are valid and consistent.

Lemma 12. *The protocol* $\Pi_{\mathsf{RandSharTrans}}$ *computes* $\mathcal{F}_{\mathsf{RandSharTrans}}$ *with perfect security when* $|\mathcal{C}_{\texttt{active}}| < |\mathcal{P}_{\texttt{active}}|/3$.

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator `Sim` to simulate the behaviors of honest parties for the adversary. Recall that $\mathcal{C}_{\texttt{active}}$ denotes the active corrupted parties and that $\mathcal{H}_{\texttt{active}}$ denotes the active honest parties.

Simulation for $\Pi_{\mathsf{RandSharTrans}}$. `Sim` simulates the protocol $\Pi_{\mathsf{RandSharTrans}}$ as follows:

1. In Step 1, `Sim` calls the simulator for Π_{Rand} to simulate the protocol Π_{Rand}. `Sim` receives either a semi-corrupted pair or the active corrupted parties' shares of the sharings $[\Phi(\boldsymbol{u}_1)]_{d'}, \ldots, [\Phi(\boldsymbol{u}_{2n'\cdot m})]_{d'}$. On receiving a semi-corrupted pair, `Sim` sends it to $\mathcal{F}_{\mathsf{RandSharTrans}}$ and halts.
2. In Step 2, `Sim` calls the simulator for Π_{Rand} to simulate the protocol Π_{Rand}. `Sim` receives either a semi-corrupted pair or the active corrupted parties' share of the sharings

$$\{[\boldsymbol{o}_j^{(1)}]_{n'-t'-1}, [\boldsymbol{o}_j^{(2)}]_{n'-t'-1}, \ldots, [\boldsymbol{o}_j^{(2m)}]_{n'-t'-1}\}_{j=1}^{n'}.$$

   On receiving a semi-corrupted pair, `Sim` sends it to $\mathcal{F}_{\mathsf{RandSharTrans}}$ and halts.

3. In step 5, $\mathtt{Sim}$ simulates the happy-bits of active honest parties as follows. For each honest party P_i, and for each corrupted party P_j, $\mathtt{Sim}$ compares the real shares held by P_j and the shares that P_j sent to P_i. If they are different, $\mathtt{Sim}$ sets P_i's happy-bit to $\mathtt{unhappy}$ on behalf of P_i.
4. In Fault Detection Phase, $\mathtt{Sim}$ simulates the byzantine agreement protocol faithfully using active honest parties' happy-bits.
5. If the simulation proceeds to Fault Localization Phase, $\mathtt{Sim}$ samples random Σ sharings that are compatible with active corrupted parties' shares on behalf of active honest parties as inputs of Π_{Rand} in Step 1. Also, $\mathtt{Sim}$ samples random Σ' sharings that are compatible with active corrupted parties' shares on behalf of active honest parties as inputs of Π_{Rand} in Step 2. Then $\mathtt{Sim}$ faithfully follows the protocol.
6. If the simulation does not proceeds to Fault Localization Phase, in Step 4, for each honest party P_j, $\mathtt{Sim}$ receives from $\mathcal{A}$ the active corrupted parties' authentic shares of the sharing $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ for all $p \in [2m]$. Then $\mathtt{Sim}$ gets from $\mathcal{A}$ the shares sent to P_j of the sharing $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ for all $p \in [2m]$. $\mathtt{Sim}$ simulates the active honest parties' shares of $([\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1})_{j\in\mathcal{C}_{\mathtt{active}}}$ for all $p \in [2m]$ as follows:
   (a) For all $i \in [r \cdot \ell]$, $\mathtt{Sim}$ samples a random Π_i-sharing $[\![\tilde{\boldsymbol{x}}_i]\!]$ that is compatible with active corrupted parties' shares sent to $\mathcal{F}_{\mathsf{RandSharTrans}}$ from $\mathcal{A}$. Because Π_i has threshold t, $\mathtt{Sim}$ can always sample such sharings. Suppose party P_j's share is denoted by $\boldsymbol{v}_j^{(i)} = (v_j^{(i,1)}, v_j^{(i,2)}, \ldots, v_j^{(i,2m)})$.
   (b) For all $j \in [n']$ and $q \in [2m]$, $\mathtt{Sim}$ computes $\boldsymbol{v}_j^{(\star,q)}$.
   (c) Because Σ' has threshold $n'-t'-r \geq t'$, the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active honest parties are independent of the secret $\boldsymbol{v}_j^{(\star,p)}$ and the shares held by active corrupted parties. For all $j \in \mathcal{C}_{\mathtt{active}}$ and for all $q \in [2m]$, $\mathtt{Sim}$ samples the random Σ'-sharings $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ based on the secrets $\boldsymbol{v}_j^{(\star,p)}$ and the shares of active corrupted parties simulated by $\mathtt{Sim}$.
   (d) $\mathtt{Sim}$ sends the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active honest parties to $\mathcal{A}$.

Hybrid Arguments. We show that $\mathtt{Sim}$ perfectly simulates the behaviors of honest parties.

Hybrid$_0$: The execution in the real world.

Hybrid$_1$: In this hybrid, $\mathtt{Sim}$ calls the simulator for Π_{Rand} to simulate the protocol Π_{Rand}. in Step 1 of $\Pi_{\mathsf{RandSharTrans}}$, and calls the simulator for Π_{Rand} to simulate the protocol Π_{Rand} in Step 3 of $\Pi_{\mathsf{RandSharTrans}}$. Then the distribution of this hybrid is identical to the distribution of **Hybrid$_0$** by the correctness of these two simulators.

Hybrid$_2$: In this hybrid, $\mathtt{Sim}$ simulates the happy-bits of active honest parties and uses them to simulate the byzantine agreement protocol in the Fault Detection Phase. Because the happy-bits of active honest parties only depends on the communication of $\mathcal{A}$, $\mathtt{Sim}$ can perfectly simulate this process.

Therefore, the distribution of this hybrid is identical to the distribution of **Hybrid$_1$**.

Hybrid$_3$: In this hybrid, $\mathtt{Sim}$ simulates the Fault Localization Phase by sampling random sharings that are compatible with active corrupted parties' shares for active honest parties in Step 1 and Step 2. Note that the Σ sharings and Σ' sharings sampled by $\mathtt{Sim}$ have the same distribution as the actual sharings sampled by active honest parties. Also note that it will not affect the output distribution, since the semi-corrupted pair is only related to the corrupted parties' messages. Therefore, the distribution of this hybrid is identical to the distribution of **Hybrid$_2$**.

Hybrid$_4$: In this hybrid, $\mathtt{Sim}$ simulates the shares of $([\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1})_{j\in\mathcal{C}_{\mathtt{active}}}$ held by active honest parties for all $p \in [2m]$ instead of getting them from active honest parties.

– We first show that the correct shares of the sharings $([\boldsymbol{s}_i^{(\star,p)}]_{n'-t'-1})_{p=1}^{2m}$ held by active corrupted parties (i.e. $\{\tilde{\boldsymbol{v}}_j^{(i)}\}_{j\in\mathcal{C}_{\mathtt{active}}}$) are the same as the distribution of that in **Hybrid$_3$**. To see this, in **Hybrid$_3$** the shares held by active corrupted parties are computed by following $\mathtt{share}_i(\boldsymbol{\tau}_i, \boldsymbol{\rho}_i)$ for all $i \in [r]$. Because Σ is a degree-d' packed Shamir secret sharing and $d'-r+1 \geq t'$, $\boldsymbol{\tau}_i$ and $\boldsymbol{\rho}_i$ are uniformly random. While in this hybrid, $\mathtt{Sim}$ randomly samples the shares of active corrupted parties by first sampling random sharings of $\Pi_1, \ldots, \Pi_r$ and then obtaining

the shares of active corrupted parties. Therefore, in this hybrid $\{\tilde{\boldsymbol{v}}_j^{(i)}\}_{j\in\mathcal{C}_{\mathtt{active}}}$ has the same distribution as that in $\mathbf{Hybrid_3}$.

– Next we argue that the active honest parties' shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ has the same distribution as that in $\mathbf{Hybrid_3}$. Recall that $[\boldsymbol{o}_j^{(p)}]_{n'-t'-1}$ are random Σ' sharings of $\mathbf{0}$. Because Σ' has a threshold no less than t', in $\mathbf{Hybrid_3}$ $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ is a random Σ' sharing of the secret $\boldsymbol{v}_j^{(\star,p)}$ given the shares of the corrupted parties and the secret. Since $\boldsymbol{v}_j^{(\star,p)}$ is determined by $\{\boldsymbol{\tau}_i,\boldsymbol{\rho}_i\}_{i=1}^r$, which are independent of the shares of $\{[\Phi(\boldsymbol{u}_w)]_{d'}\}_{w=1}^{2n'\cdot m}$ and $[\boldsymbol{o}_j^{(p)}]_{n'-t'-1}$ held by active corrupted parties, $\boldsymbol{v}_j^{(\star,p)}$ is independent of the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active corrupted parties.
In this hybrid, because $\mathtt{Sim}$ computes the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active corrupted parties given the shares of $[\boldsymbol{o}_j^{(p)}]_{n'-t'-1}$ held by active corrupted parties, and because the secret $\boldsymbol{v}_j^{(\star,p)}$, a part of $\{\boldsymbol{v}_j^{(i)}\}_{j\in\mathcal{C}_{\mathtt{active}}}$, has the same distribution as the one in $\mathbf{Hybrid_3}$, we have that the secret $\boldsymbol{v}_j^{(\star,p)}$ and the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active corrupted parties has the same distribution as in $\mathbf{Hybrid_3}$. Since the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active honest parties in this hybrid are sampled according to the secret $\boldsymbol{v}_j^{(\star,p)}$ and the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active corrupted parties computed by $\mathtt{Sim}$, they have the same distribution as that in $\mathbf{Hybrid_3}$.

– Finally we show that the distribution of active honest parties' output in $\Pi_{\mathsf{RandSharTrans}}$ given the view of corrupted parties is the same as that in $\mathbf{Hybrid_3}$. We have (1) for all $w\in[2n'\cdot m]$, the shares of $[\Phi(\boldsymbol{u}_w)]_{d'}$ held by active corrupted parties are independent of $\{\boldsymbol{\tau}_i,\boldsymbol{\rho}_i\}_{i=1}^r$; (2) for all $j\in[n']$ and for all $p\in[2m]$, the shares of $[\boldsymbol{o}_j^{(p)}]_{n'-t'-1}$ held by active corrupted parties are independent of $\{\boldsymbol{\tau}_i,\boldsymbol{\rho}_i\}_{i=1}^r$; (3) for all $j\in\mathcal{C}_{\mathtt{active}}$, the shares of $[\boldsymbol{s}_j^{(\star,p)}]_{n'-t'-1}$ held by active honest parties are only dependent on $\{\boldsymbol{v}_j^{(i)}\}_{j\in\mathcal{C}_{\mathtt{active}}}$. Recall that $[\![\boldsymbol{x}_i]\!]=\mathtt{share}_i(\boldsymbol{\tau}_i,\boldsymbol{\rho}_i)$. Therefore, in $\mathbf{Hybrid_3}$ the view of corrupted parties is independent of the joint of shares of $[\![\boldsymbol{x}_i]\!]=\mathtt{share}_i(\boldsymbol{\tau}_i,\boldsymbol{\rho}_i)$ held by active honest parties and the secret $\boldsymbol{x}_i$.
In this hybrid, $\mathtt{Sim}$ perfectly simulates the shares of $[\![\boldsymbol{x}_i]\!]=\mathtt{share}_i(\boldsymbol{\tau}_i,\boldsymbol{\rho}_i)$ held by active corrupted parties and send them to $\mathcal{F}_{\mathsf{RandSharTrans}}$, and then $\mathcal{F}_{\mathsf{RandSharTrans}}$ samples a random Π_i-sharing based on the shares of active corrupted parties. The output of active honest parties is their shares of $[\![\boldsymbol{x}_i]\!]$ for all $i\in[r\cdot\ell]$.
Therefore, the output of active honest parties given the joint view of corrupted parties in this hybrid has the same distribution as that in $\mathbf{Hybrid_3}$.

Therefore, the distribution of this hybrid is identical to the distribution of $\mathbf{Hybrid_3}$.

Note that this hybrid is the execution in the ideal world.

E.3 Performing Linear Transformations

Let $L_1,\ldots,L_N$ be N linear mappings in the linear mapping set $\mathcal{L}((\mathbb{Z}/2^k\mathbb{Z})^\ell,(\mathbb{Z}/2^k\mathbb{Z})^\ell)$, and let $[\phi(\boldsymbol{x}_1)]_t,\ldots,[\phi(\boldsymbol{x}_N)]_t$ be N Shamir sharings. We introduce the functionality that applies the linear transformation L_i on the Shamir sharing $[\phi(\boldsymbol{x}_i)]_t$ and gets $[\phi(L_i(\boldsymbol{x}_i))]_t$ for all $i\in[N]$, or outputs a semi-corrupted pair $\{P_i,P_j\}$ such that $\{P_i,P_j\}\cap\mathcal{C}_{\mathtt{active}}\neq\emptyset$. The functionality $\mathcal{F}_{\mathsf{LinearTrans}}$ is described in Functionality 17. We assume that for each degree-t Shamir secret sharing, the shares of all active honest parties lie on a degree-t polynomial.

Functionality 17: $\mathcal{F}_{\mathsf{LinearTrans}}(N)$

1. Let $[\phi(\boldsymbol{x}_1)]_t,\ldots,[\phi(\boldsymbol{x}_N)]_t$ denote the input sharings. Let $L_1,\ldots,L_N$ be N linear mappings in the linear mapping set $\mathcal{L}((\mathbb{Z}/2^k\mathbb{Z})^\ell,(\mathbb{Z}/2^k\mathbb{Z})^\ell)$.
2. $\mathcal{F}_{\mathsf{LinearTrans}}$ receives from active honest parties their shares of $[\phi(\boldsymbol{x}_1)]_t,\ldots,[\phi(\boldsymbol{x}_N)]_t$, and reconstructs $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_N$. $\mathcal{F}_{\mathsf{LinearTrans}}$ further computes the shares of $[\phi(\boldsymbol{x}_1)]_t,\ldots,[\phi(\boldsymbol{x}_N)]_t$ held by active corrupted parties and send them to the adversary.
3. $\mathcal{F}_{\mathsf{LinearTrans}}$ receives from the adversary a set of even number of parties $S\subset\mathcal{P}_{\mathtt{active}}$ such that $|S|/2\le|S\cap\mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{LinearTrans}}$ updates $\mathcal{P}_{\mathtt{active}}:=\mathcal{P}_{\mathtt{active}}-S$. $\mathcal{F}_{\mathsf{LinearTrans}}$ then sends S to all active honest parties.

4. $\mathcal{F}_{\mathsf{LinearTrans}}$ computes $L_1(\boldsymbol{x}_1), \ldots, L_N(\boldsymbol{x}_N)$, and then $\mathcal{F}_{\mathsf{LinearTrans}}$ receives from the adversary the shares of active corrupted parties of $[\phi(L_1(\boldsymbol{x}_1))]_t, \ldots, [\phi(L_N(\boldsymbol{x}_N))]_t$;
5. $\mathcal{F}_{\mathsf{LinearTrans}}$ samples the whole sharing $[\phi(L_1(\boldsymbol{x}_1))]_t, \ldots, [\phi(L_N(\boldsymbol{x}_N))]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{LinearTrans}}$ sends P_h's shares of $[\phi(L_1(\boldsymbol{x}_1))]_t, \ldots, [\phi(L_N(\boldsymbol{x}_N))]_t$ to P_h.

Protocol Description. Recall that in Section E.2, for the linear mapping $L_i : (\mathbb{Z}/2^k\mathbb{Z})^\ell \to (\mathbb{Z}/2^k\mathbb{Z})^\ell$ we construct a corresponding $\mathbb{Z}/2^k\mathbb{Z}$-arithmetic secret sharing scheme Π_i as follows:

- The secret space $Z_i = (\mathbb{Z}/2^k\mathbb{Z})^\ell$.
- The share space $U_i = (\mathcal{R})^2$.
- For a secret $\boldsymbol{x} \in Z_i$, the sharing of $\boldsymbol{x}$ is $([\phi(\boldsymbol{x})]_t, [\phi(L_i(\boldsymbol{x}))]_t)$.
- For a sharing $([\phi(\boldsymbol{x})]_t, [\phi(L_i(\boldsymbol{x}))]_t)$, we reconstruct the secret $\boldsymbol{x}$ using $[\phi(\boldsymbol{x})]_t$.

Therefore, we can use $\mathcal{F}_{\mathsf{RandSharTrans}}$ to prepare random Π_i sharings, and perform linear transformation to a sharing $[\phi(\boldsymbol{x})]_t$ following the idea in [GPS22]. Our protocol that implements $\mathcal{F}_{\mathsf{LinearTrans}}$ is described in Protocol 13.

Protocol 13: $\Pi_{\mathsf{LinearTrans}}(N, \{[\phi(\boldsymbol{x}_i)]_t\}_{i=1}^N, \{L_i\}_{i=1}^N)$

1. Recall that $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t$ are the input sharings, and that $L_1, \ldots, L_N$ are N linear mappings in the linear mapping set $\mathcal{L}((\mathbb{Z}/2^k\mathbb{Z})^\ell, (\mathbb{Z}/2^k\mathbb{Z})^\ell)$. Let Π_i denote the $\mathbb{Z}/2^k\mathbb{Z}$-arithmetic secret sharing scheme corresponding to L_i as defined previously. All parties set the eliminated party set $S = \emptyset$.
2. All parties invoke $\mathcal{F}_{\mathsf{RandSharTrans}}$ $\lceil \frac{N}{r \cdot \ell} \rceil$ times. All parties receive the random $\Pi_1, \ldots, \Pi_N$ sharings $([\phi(\boldsymbol{r}_1)]_t, [\phi(L_1(\boldsymbol{r}_1))]_t), \ldots, ([\phi(\boldsymbol{r}_N)]_t, [\phi(L_N(\boldsymbol{r}_N))]_t)$ If any call of $\mathcal{F}_{\mathsf{RandSharTrans}}$ outputs a semi-corrupted pair E, all parties eliminate the semi-corrupted pair, set $S := S \cup E$ and call $\mathcal{F}_{\mathsf{RandSharTrans}}$ again.
3. All parties invoke $\mathcal{F}_{\mathsf{OpenPub}}(N)$ on the sharings $[\phi(\boldsymbol{x}_1)]_t + [\phi(\boldsymbol{r}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t + [\phi(\boldsymbol{r}_N)]_t$ and get the output $\phi(\boldsymbol{x}_1) + \phi(\boldsymbol{r}_1), \ldots, \phi(\boldsymbol{x}_N) + \phi(\boldsymbol{r}_N)$. Then all parties locally compute $\boldsymbol{x}_1 + \boldsymbol{r}_1, \ldots, \boldsymbol{x}_N + \boldsymbol{r}_N$ by applying $\psi(\cdot)$ to $\phi(\boldsymbol{x}_1) + \phi(\boldsymbol{r}_1), \ldots, \phi(\boldsymbol{x}_N) + \phi(\boldsymbol{r}_N)$.
4. All parties output the eliminated party set S. For all $i \in [N]$, all parties locally compute $\phi(L_i(\boldsymbol{x}_i + \boldsymbol{r}_i))$, and then take $L_i(\boldsymbol{x}_i + \boldsymbol{r}_i) - [\phi(L_i(\boldsymbol{r}_i))]_t$ as output.

Cost of Protocol 13. In Step 2, all parties invoke $\mathcal{F}_{\mathsf{RandSharTrans}}$ $O(\frac{N}{r \cdot \ell} + t)$ times, so the cost of Step 2 is $O(\frac{N}{r \cdot \ell} \cdot n^2 \cdot m^2 + n^2 \cdot m^2 \cdot t)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. In Step 3, the communication complexity of calling $\mathcal{F}_{\mathsf{OpenPub}}(N)$ is $O(n \cdot m \cdot N + n^2 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Note that $m/\ell = O(1)$, $r = O(n)$ and $t = O(n)$, so the total communication complexity of Protocol 13 is $O(N \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 13. *The protocol $\Pi_{\mathsf{LinearTrans}}$ computes $\mathcal{F}_{\textit{linear-transformation}}$ with perfect security in the $(\mathcal{F}_{\mathsf{RandSharTrans}}, \mathcal{F}_{\mathsf{OpenPub}})$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 2, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{RandSharTrans}}$. In Step 3, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{OpenPub}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

E.4 Fan-Out, Permute, and Collect

Instantiating $\mathcal{F}_{\mathsf{FanOut}}$. The functionality $\mathcal{F}_{\mathsf{FanOut}}$ is described in Functionality 18.

Functionality 18: $\mathcal{F}_{\mathsf{FanOut}}(N)$

1. Let $[\phi(\boldsymbol{z}^{(1)})]_t, \ldots, [\phi(\boldsymbol{z}^{(N)})]_t$ denote the input sharings. Let $n_j^{(i)}$ denote the number of times the j-th wire of $\boldsymbol{z}^{(i)}$ needs to be copied and let $m^{(i)} = \frac{\sum_{j=1}^{\ell} n_j^{(i)}}{\ell}$. For all $i \in [N]$, let $f_j^{(i)}$ denote the desired fan-out linear transformations for $j = 1, \ldots, m^{(i)}$.
2. $\mathcal{F}_{\mathsf{FanOut}}$ receives from active honest parties their shares of $[\phi(\boldsymbol{z}^{(1)})]_t, \ldots, [\phi(\boldsymbol{z}^{(N)})]_t$. Then $\mathcal{F}_{\mathsf{FanOut}}$ reconstructs the secrets $\phi(\boldsymbol{z}^{(1)}), \ldots, \phi(\boldsymbol{z}^{(N)})$. $\mathcal{F}_{\mathsf{FanOut}}$ further computes the shares of $[\phi(\boldsymbol{z}^{(1)})]_t, \ldots, [\phi(\boldsymbol{z}^{(N)})]_t$ held by active corrupted parties and send the shares to the adversary.
3. $\mathcal{F}_{\mathsf{FanOut}}$ locally computes $\boldsymbol{z}^{(i)} = \psi(\phi(\boldsymbol{z}^{(i)}))$ for all $i \in [N]$. Then for all $i \in [N]$, $\mathcal{F}_{\mathsf{FanOut}}$ computes $\boldsymbol{x}_j^{(i)} := f_j^{(i)} \cdot \boldsymbol{z}^{(i)}$ for all $j = 1, \ldots, m^{(i)}$.
4. $\mathcal{F}_{\mathsf{FanOut}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{FanOut}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{FanOut}}$ then sends S to all active honest parties.
5. $\mathcal{F}_{\mathsf{FanOut}}$ receives from the adversary the shares of active corrupted parties of $[\phi(\boldsymbol{x}_j^{(i)})]_t$ for all $i \in [N]$ and for all $j \in [m^{(i)}]$.
6. $\mathcal{F}_{\mathsf{FanOut}}$ samples the whole sharings $[\phi(\boldsymbol{x}_j^{(i)})]_t$ so that they are compatible with the active corrupted parties' shares for all $i \in [N]$ and for all $j \in [m^{(i)}]$. For each active honest party P_h, $\mathcal{F}_{\mathsf{FanOut}}$ sends P_h's shares of $[\phi(\boldsymbol{x}_j^{(i)})]_t$ for all $i \in [N]$ and for all $j \in [m^{(i)}]$ to P_h.

Note that each fan-out operation can be viewed as a linear transformation of the original secret sharing, so we will implement $\mathcal{F}_{\mathsf{FanOut}}$ using the functionality $\mathcal{F}_{\mathsf{LinearTrans}}$. Let $\{[\phi(\boldsymbol{z}^{(i)})]_t\}_{i=1}^N$ denote the input Shamir sharings of the protocol, and let $n_j^{(i)}$ denote the number of times the j-th wire needs to be copied in the fan-out sharings. We require that $\sum_{j=1}^{\ell} n_j^{(i)}$ is a multiple of ℓ greater than 0 for all $i \in [N]$. Recall that $m^{(i)} := \frac{\sum_{j=1}^{\ell} n_j^{(i)}}{\ell}$ denote the number of fan-out operations needed for the output sharing $[\phi(\boldsymbol{z}^{(i)})]_t$. All parties first need to agree on the sequence of all entries in the fan-out results. For all $i \in [N]$, let $f_j^{(i)}$ denote the agreed fan-out operations for $j = 1, \ldots, m^{(i)}$. Note that $f_j^{(i)}$ is a linear transformation. The description of Π_{FanOut} that implements $\mathcal{F}_{\mathsf{FanOut}}$ appears in Protocol 14.

Protocol 14: $\Pi_{\mathsf{FanOut}}(N, \{([\phi(\boldsymbol{z}^{(i)})]_t, (n_j^{(i)})_{j=1}^{m^{(i)}})\}_{i=1}^N)$

1. All parties locally compute the desired fan-out linear transformations $f_j^{(i)}$ on $\boldsymbol{z}^{(i)}$ for $j = 1, \ldots, m^{(i)}$ for all $i \in [N]$.
2. All parties invoke $\mathcal{F}_{\mathsf{LinearTrans}}$ with $[\phi(\boldsymbol{z}^{(1)})]_t, \ldots, [\phi(\boldsymbol{z}^{(N)})]_t$ and the fan-out operations $\{f_j^{(i)} : j \in [m^{(i)}], i \in [N]\}$. Then all parties get the set of eliminated parties S from $\mathcal{F}_{\mathsf{LinearTrans}}$ and the output sharings
$$\{[\phi(f_j^{(i)} \cdot \boldsymbol{z}^{(i)})]_t : j \in [m^{(i)}], i \in [N]\}.$$
All parties output S and all sharings in $\{[\phi(f_j^{(i)} \cdot \boldsymbol{z}^{(i)})]_t : j \in [m^{(i)}], i \in [N]\}$.

Let $M := \sum_{i=1}^N m^{(i)}$ denote the total number of fan-out sharings which are the output of the protocol. In Step 2, the communication complexity of calling $\mathcal{F}_{\mathsf{LinearTrans}}(M)$ is $O(M \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$. Therefore, the total communication complexity of Π_{FanOut} is $O(M \cdot n \cdot m + n^3 \cdot m^2)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Lemma 14. *The protocol Π_{FanOut} computes $\mathcal{F}_{\mathsf{FanOut}}$ with perfect security in the $\mathcal{F}_{\mathsf{LinearTrans}}$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 2, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{LinearTrans}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

Instantiating $\mathcal{F}_{\mathsf{Permute}}$. Let $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t$ denote the input sharings, and let $p_1, \ldots, p_N$ denote the desired permutations. The description of $\mathcal{F}_{\mathsf{Permute}}$ appears in Functionality 19.

Functionality 19: $\mathcal{F}_{\mathsf{Permute}}(N)$

1. Let $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t$ denote the input sharings. Let $p_1, \ldots, p_N$ denote the desired permutations. $\mathcal{F}_{\mathsf{Permute}}$ receives from active honest parties their shares of $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t$. Then $\mathcal{F}_{\mathsf{Permute}}$ reconstructs the secrets $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_N)$. $\mathcal{F}_{\mathsf{Permute}}$ further computes the shares of $([\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t)$ held by active corrupted parties and send the shares to the adversary.
2. $\mathcal{F}_{\mathsf{Permute}}$ locally computes $\boldsymbol{x}_i = \psi(\phi(\boldsymbol{x}_i))$ for all $i \in [N]$. Then $\mathcal{F}_{\mathsf{Permute}}$ computes $\boldsymbol{y}_i := p_i \cdot \boldsymbol{x}_i$ for all $i \in [N]$.
3. $\mathcal{F}_{\mathsf{Permute}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{Permute}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{Permute}}$ then sends S to all active honest parties.
4. $\mathcal{F}_{\mathsf{Permute}}$ receives from the adversary the shares of active corrupted parties of $[\phi(\boldsymbol{y}_1)]_t, \ldots, [\phi(\boldsymbol{y}_N)]_t$.
5. $\mathcal{F}_{\mathsf{Permute}}$ samples the whole sharings $[\phi(\boldsymbol{y}_1)]_t, \ldots, [\phi(\boldsymbol{y}_N)]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{Permute}}$ sends P_h's shares of $[\phi(\boldsymbol{y}_1)]_t, \ldots, [\phi(\boldsymbol{y}_N)]_t$ to P_h.

The description of Π_{Permute} that implements $\mathcal{F}_{\mathsf{Permute}}$ appears in Protocol 15. The communication complexity of Π_{Permute} with N input sharings is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Protocol 15: $\Pi_{\mathsf{Permute}}(N, \{[\phi(\boldsymbol{x}_i)]_t\}_{i=1}^N, \{p_i\}_{i=1}^N)$

1. All parties invoke $\mathcal{F}_{\mathsf{LinearTrans}}(N)$ with the input $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_N)]_t$ and the desired permutations $p_1, \ldots, p_N$, and get a set of eliminated parties S and the sharings $[\phi(p_1 \cdot \boldsymbol{x}_1)]_t, \ldots, [\phi(p_N \cdot \boldsymbol{x}_N)]_t$ as output.
2. All parties output S and the sharings $[\phi(p_1 \cdot \boldsymbol{x}_1)]_t, \ldots, [\phi(p_N \cdot \boldsymbol{x}_N)]_t$.

Lemma 15. *The protocol Π_{Permute} computes $\mathcal{F}_{\mathsf{Permute}}$ with perfect security in the $\mathcal{F}_{\mathsf{LinearTrans}}$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 1, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{LinearTrans}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

Instantiating $\mathcal{F}_{\mathsf{Collect}}$. The description of $\mathcal{F}_{\mathsf{Collect}}$ appears in Functionality 20.

Functionality 20: $\mathcal{F}_{\mathsf{Collect}}(N)$

1. Let $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_M)]_t$ denote the input Shamir sharings, and let $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$ denote the target output Shamir sharings. Suppose that the j-th wire of $\boldsymbol{z}_i$ is collected from the $u_{i,j}$-th wire of $\boldsymbol{x}_{w_{i,j}}$.
2. $\mathcal{F}_{\mathsf{Collect}}$ receives from active honest parties their shares of $[\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_M)]_t$. $\mathcal{F}_{\mathsf{Collect}}$ reconstructs the secrets $\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_M)$. $\mathcal{F}_{\mathsf{Collect}}$ further computes the shares of $([\phi(\boldsymbol{x}_1)]_t, \ldots, [\phi(\boldsymbol{x}_M)]_t)$ held by active corrupted parties and send the shares to the adversary.
3. $\mathcal{F}_{\mathsf{Collect}}$ locally computes $\boldsymbol{x}_i = \psi(\phi(\boldsymbol{x}_i))$ for all $i \in [M]$. Then $\mathcal{F}_{\mathsf{Collect}}$ computes
$$\boldsymbol{z}_i := \sum_{j=1}^{\ell} \boldsymbol{e}_{u_{i,j}} \star \boldsymbol{x}_{w_{i,j}}$$
for all $i \in [N]$.
4. $\mathcal{F}_{\mathsf{Collect}}$ receives from the adversary a set of even number of parties $S \subset \mathcal{P}_{\mathtt{active}}$ such that $|S|/2 \leq |S \cap \mathcal{C}_{\mathtt{active}}|$. $\mathcal{F}_{\mathsf{Collect}}$ updates $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$. $\mathcal{F}_{\mathsf{Collect}}$ then sends S to all active honest parties.
5. $\mathcal{F}_{\mathsf{Collect}}$ receives from the adversary the shares of active corrupted parties of $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$.
6. $\mathcal{F}_{\mathsf{Collect}}$ samples the sharings $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$ so that they are compatible with the active corrupted parties' shares. For each active honest party P_h, $\mathcal{F}_{\mathsf{Collect}}$ sends P_h's shares of $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$ to P_h.

The protocol Π_{Collect} implements $\mathcal{F}_{\mathsf{Collect}}$. The description of Π_{Collect} appears in Protocol 16. The communication complexity of Π_{Collect} to generate N output sharings is $O(N \cdot n \cdot m + n^3 \cdot m)$ elements in $\mathbb{Z}/2^k\mathbb{Z}$.

Protocol 16: $\Pi_{\mathsf{Collect}}(N, \{[\phi(\boldsymbol{x}_i)]_t\}_{i=1}^M, \{(u_{i,j}, w_{i,j})\}_{i\in[N], j\in[\ell]})$

1. For all $i \in [N]$, all parties locally compute
$$[s_i]_t := \sum_{j=1}^{\ell} \phi(\boldsymbol{e}_{u_{i,j}}) \cdot [\phi(\boldsymbol{x}_{w_{i,j}})]_t.$$
We note that $\psi(s_i) = \boldsymbol{x}_i$ for all $i \in [N]$.
2. All parties call the functionality $\mathcal{F}_{\mathsf{ReEncode}}$ with the input $[s_1]_t, \ldots, [s_N]_t$, and get a set of eliminated parties S and the sharings $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$.
3. All parties locally update $\mathcal{P}_{\mathtt{active}} := \mathcal{P}_{\mathtt{active}} - S$, and then output the eliminated party set S and the sharings $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$.

Lemma 16. *The protocol Π_{Collect} computes $\mathcal{F}_{\mathsf{Collect}}$ with perfect security in the $\mathcal{F}_{\mathsf{ReEncode}}$-hybrid model when $|\mathcal{C}_{\mathtt{active}}| < |\mathcal{P}_{\mathtt{active}}|/3$.*

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator $\mathtt{Sim}$ to simulate the behaviors of honest parties for the adversary. In Step 2, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{ReEncode}}$. Because there is no other communication, the distribution of the real world is the same as the distribution of the ideal world.

F Security Proofs

F.1 Proof of Lemma 3

Proof. Suppose by contradiction that in some scenario the semi-corrupted pair E selected by the protocol consists of two honest parties. We discuss two cases based on whether the dealer is corrupted or not.

When D is honest, we have that the sharing $\langle \hat{o} \rangle$ must be an invalid additive sharing of 0. To see this, if $\langle o'' \rangle$ is invalid D will broadcast $\langle o'' \rangle$ is invalid, and if $\langle o'' \rangle$ is valid then $\langle o \rangle - \langle o' \rangle + \langle o'' \rangle$ must be an invalid additive sharing of 0 since $\langle o \rangle$ is invalid and $\langle o' \rangle$ is derived robustly due to the robust reconstruction of the Shamir sharings. Also, D can broadcast at least one of the messages in Step 6. Otherwise, all parties have sent correct sharings and all parties' communication is consistent, so $\sum_{i=1}^{n'} \hat{o}_j^{(i)} = 0$ for all $j \in [n']$, which is contradictory to $\sum_{j=1}^{n'} \sum_{i=1}^{n'} \hat{o}_j^{(i)} \neq 0$. By the assumption, D should not have broadcast the message $(P_i, \mathtt{incorrect})$ in Step 6, or otherwise P_i must be corrupted. It follows that D broadcast the message $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \mathtt{inconsistent})$ in Step 6. Because P_{j_1} and P_{j_2} cannot both be honest for D to broadcast this message, and by the assumption at least one of them is honest, so exactly one of P_{j_1} and P_{j_2} is corrupted.

- If P_{j_1} is corrupted, then $E = \{D, P_{j_2}\}$ according to the assumption. So P_{j_1} agrees with the broadcast message, and P_{j_2} does not agree with the broadcast message. However, P_{j_2} will disagree only if the message $(\hat{o}_{j_1}^{(j_2)})'$ broadcast by D is not the message that P_{j_2} has sent to D, which is not possible since D is honest.
- If P_{j_2} is corrupted, then $E = \{D, P_{j_1}\}$ according to the assumption. So P_{j_1} does not agree with the broadcast message. However, P_{j_1} will disagree only if the message $\hat{o}_{j_1}^{(j_2)}$ broadcast by D is not the message that P_{j_1} has sent to D, which is not possible since D is honest.

When D is corrupted, by the assumption D must broadcast the message $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \mathtt{inconsistent})$ in Step 6. Also, we must have P_{j_1} and P_{j_2} are honest, and they all agree with the broadcast message according to the assumption. However, this is not possible because honest parties $\hat{o}_{j_1}^{(j_2)}$ that P_{j_1} claims to have sent to P_{j_2} will always equal to $(\hat{o}_{j_1}^{(j_2)})'$ that P_{j_2} claims to have received from P_{j_1}.

Above all, in all cases, the protocol is always able to find a semi-corrupted pair.

F.2 Proof of Lemma 10

Proof. In Step 2, if the functionality $\mathcal{F}_{\mathsf{VerifyBC}}$ outputs `consistent`, all parties will hold consistent values of $\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)}$ and $\boldsymbol{y}^{(i)}+\boldsymbol{b}^{(i)}$ for all $i \in [N]$. If the output of $\mathcal{F}_{\mathsf{VerifyBC}}$ is a semi-corrupted pair, it is guaranteed by the functionality that at least one party of the semi-corrupted pair is corrupted.

If all parties do not halt on Step 2, and if all the parties hold the correct $\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)}$ and $\boldsymbol{y}^{(i)}+\boldsymbol{b}^{(i)}$ for all $i \in [N]$, then it is clear the protocol will generate random output Shamir sharings for all gate groups correctly, and all parties will pass the verification.

Next we need only show that if all parties does not halt on Step 2, and if there exists some $\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)}$ or $\boldsymbol{y}^{(i)}+\boldsymbol{b}^{(i)}$ is not correct, then all parties will output a semi- where at least one is corrupted.

Consider the incorrect entry of $\boldsymbol{x}^{(i)}+\boldsymbol{a}^{(i)}$ or $\boldsymbol{y}^{(i)}+\boldsymbol{b}^{(i)}$ that has the smallest topological order, W.L.O.G. we suppose the entry is $x_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}$. Since from Step 3 to Step 6, all the computation that derives $\tilde{\boldsymbol{x}}^{(i_0)}+\boldsymbol{a}^{(i_0)}$ is robust and only involves the correct Shamir secret sharings, we have that $\tilde{x}_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}$ should be the correct value of $x_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}$. Therefore, in Step 7 all parties can detect this incorrect entry and they will proceed to Step 7.b.

If there already exist some eliminated parties, *i.e.* $S \neq \emptyset$ in Step 10.(b), it is guaranteed by the previous functionalities that at least half of S are corrupted. Otherwise, all active parties will run $\Pi_{\mathsf{FaultDetection}}$.

Note that the extended additive sharing of $x_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}$ is calculated correctly. Also note that all parties that holds the shares of $(\!|x_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}|\!)$ from the beginning of Π_{Eval} remain active in this step. Since the dealer D has all parties' shares of $\langle x_{j_0}^{(i_0)}\rangle+\langle a_{j_0}^{(i_0)}\rangle+\langle o_{j_0}^{(i_0)}\rangle$, and $\langle x_{j_0}^{(i_0)}\rangle+\langle a_{j_0}^{(i_0)}\rangle$ is derived from the extended additive sharing $(\!|x_{j_0}^{(i_0)}+a_{j_0}^{(i_0)}|\!)$, by Lemma 3 all parties can call $\Pi_{\mathsf{FaultDetection}}$ to output a set of eliminated parties where at least half of the set are corrupted.

F.3 Proof of Lemma 4

Proof. Let $\mathcal{A}$ denote the adversary. We will construct a simulator `Sim` to simulate the behaviors of honest parties for the adversary. Recall that $\mathcal{C}_{\mathtt{active}}$ denotes the active corrupted parties and that $\mathcal{H}_{\mathtt{active}}$ denotes the active honest parties.

The correctness of Π_{Main} follows the correctness of the protocol Π_{Verify} (Lemma 10) and the correctness of network routing (Theorem 2).

Simulation for Π_{Main}. `Sim` simulates Π_{Main} as follows:

1. In Step 1, `Sim` does nothing.
2. In Step 2, `Sim` simulates Π_{Input} as follows:
   (a) In Step 1.(b) of Π_{Input}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{InputShamir}}$.
   (b) In Step 1.(c) of Π_{Input}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{RandShamir}}$.
   (c) In Step 1.(d) of Π_{Input}, if Client is corrupted, `Sim` learns $y_1, \ldots, y_N$ from $\mathcal{A}$. Then `Sim` samples $\boldsymbol{r}_i \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$, and then send $y_i + \phi(\boldsymbol{r}_i)$ to the active corrupted parties on behalf of $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$. If Client is honest, `Sim` samples random $\tilde{\boldsymbol{r}}_i \in (\mathbb{Z}/2^k\mathbb{Z})^\ell$, and sends $\phi(\tilde{\boldsymbol{r}}_i)$ to $\mathcal{A}$ on behalf of $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$.
   (d) In Step 1.(e) of Π_{Input}, `Sim` does nothing.
   (e) In Step 2 of Π_{Input}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{FanOut}}$.
   (f) In Step 3 of Π_{Input}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{Permute}}$.
3. In Step 3.(a), `Sim` simulates Π_{Eval} for `seg` as follows:
   (a) In Step 2 of Π_{Eval}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{RandZeroAdd}}$.
   (b) In Step 3 of Π_{Eval}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{TripleAdd}}$ and $\mathcal{F}_{\mathsf{TripleMult}}$.
   (c) In Step 5.(b) of Π_{Eval}, if D is corrupted, `Sim` samples uniformly random values in $\mathbb{Z}/2^k\mathbb{Z}$ as the shares of $\langle x\rangle+\langle a_j\rangle+\langle o_1\rangle$ and $\langle y\rangle+\langle b_j\rangle+\langle o_2\rangle$ held by active honest parties whose indices are in $\{1, 2, \ldots, t+1\}$. If D is honest, `Sim` follows the protocol.
   (d) In step 5.(c) of Π_{Eval}, if D is corrupted, `Sim` follows the protocol. If D is honest, `Sim` samples two uniformly random elements in $\mathbb{Z}/2^k\mathbb{Z}$ as u and v sent to active corrupted parties.
4. In Step 3.(b), `Sim` simulates Π_{Verify} for `seg` as follows:
   (a) In Step 2 of Π_{Verify}, `Sim` faithfully emulates $\mathcal{F}_{\mathsf{VerifyBC}}$.

(b) In Step 4 of Π_{Verify}, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{Mult}}$ and then faithfully emulates $\mathcal{F}_{\mathsf{ReEncode}}$.

(c) In Step 5 of Π_{Verify}, $\mathtt{Sim}$ faithfully emulates the functionalities $\mathcal{F}_{\mathsf{FanOut}}$, $\mathcal{F}_{\mathsf{Permute}}$, $\mathcal{F}_{\mathsf{Collect}}$ and $\mathcal{F}_{\mathsf{Permute}}$ in the protocol $\Pi_{\mathsf{NetworkRouting}}$.

(d) In Step 6 of Π_{Verify}, recall that $\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}$ and $\boldsymbol{y}^{(i)} + \boldsymbol{b}^{(i)}$ are computed from the values sent by D in Π_{Eval}. Let $\tilde{u}_j^{(i)} := \tilde{x}_j^{(i)} + a_j^{(i)}$ denote the j-th wire of $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$, and let $u_j^{(i)} := \tilde{x}_j^{(i)} + a_j^{(i)}$ denote the j-th wire of $\boldsymbol{x}^{(i)} + \boldsymbol{a}^{(i)}$. Similarly, let $\tilde{y}_j^{(i)} + b_j^{(i)}$ denote the j-th wire of $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$. Note that $u_j^{(i)}$ is reconstructed by D at Step 5.(c) of Π_{Eval}. Also note that $u_j^{(i)}$ should equal $\tilde{u}_j^{(i)}$ if the reconstruction is correct, because of the correctness of network routing.
If D is corrupted, $\mathtt{Sim}$ simulates $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ as follows. Since D is corrupted, we must have that $\mathcal{A}$ locally possesses this correct value $\tilde{u}_j^{(i)}$ at Step 5.(b) of Π_{Eval}. Therefore, $\mathtt{Sim}$ gets $\tilde{u}_j^{(i)} = \tilde{x}_j^{(i)} + a_j^{(i)}$ from $\mathcal{A}$ for all $i \in [N]$ and for all $j \in [\ell]$. Similarly, $\mathtt{Sim}$ can get $\tilde{y}_j^{(i)} + b_j^{(i)}$ from $\mathcal{A}$ for all $i \in [N]$ and for all $j \in [\ell]$. Then $\mathtt{Sim}$ sends $\phi(\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)})$ to the active corrupted parties on behalf of $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$.
If D is honest, $\mathtt{Sim}$ simulates $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ as follows. At Step 5.(b) of Π_{Eval}, $\mathcal{A}$ possesses the additive error introduced to $\tilde{u}_j^{(i)}$, which we suppose is $d_j^{(i)} \in \mathbb{Z}/2^k\mathbb{Z}$. Therefore, $\mathtt{Sim}$ receives $d_j^{(i)}$ from $\mathcal{A}$, and sets $\tilde{u}_j^{(i)} = u_j^{(i)} - d_j^{(i)}$ for all $i \in [N]$ and for all $j \in [\ell]$. Similarly, $\mathtt{Sim}$ can get $\tilde{y}_j^{(i)} + b_j^{(i)}$ from $\mathcal{A}$ for all $i \in [N]$ and for all $j \in [\ell]$. Then $\mathtt{Sim}$ sends $\phi(\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)})$ to the active corrupted parties on behalf of $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$.

(e) In Step 7.(a) of Π_{Verify}, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{FanOut}}$, and then faithfully emulates $\mathcal{F}_{\mathsf{Permute}}$.

(f) In Step 7.(b) of Π_{Verify}, it is clear that $\mathtt{Sim}$ can locate the first wire that is incorrect, denoted by $\langle\!|s|\!\rangle$ and $\langle s \rangle + \langle o \rangle$. Then $\mathtt{Sim}$ simulates $\Pi_{\mathsf{FaultDetection}}$ as follows:

i. In Step 1 of $\Pi_{\mathsf{FaultDetection}}$, if P_i is corrupted, $\mathtt{Sim}$ receives from $\mathcal{A}$ the shares of $\langle o_i'' \rangle$ for active honest parties, and sends them to active honest parties. If P_i is honest, $\mathtt{Sim}$ samples uniformly random elements in $\mathbb{Z}/2^k\mathbb{Z}$ as the shares of $\langle o_i'' \rangle$ held by the active corrupted parties whose indices are in $\{1, 2, \ldots, t+1\}$.

ii. In Step 2 of $\Pi_{\mathsf{FaultDetection}}$, if D is corrupted, $\mathtt{Sim}$ samples a random additive zero sharing as the sharing $\sum_{i \in \mathcal{H}_{\mathtt{active}}} \langle o_i'' \rangle$. Then $\mathtt{Sim}$ sets $\langle o'' \rangle = \sum_{i \in \mathcal{H}_{\mathtt{active}}} \langle o_i'' \rangle + \sum_{i \in \mathcal{C}_{\mathtt{active}}} \langle o_i'' \rangle$ as the communication from active honest parties to D. Afterwards, $\mathtt{Sim}$ simulates all the communication in the byzantine agreement protocol.
If D is honest, $\mathtt{Sim}$ receives from $\mathcal{A}$ the shares of $\langle o'' \rangle$ sent by active corrupted parties. $\mathtt{Sim}$ follows the protocol to send active corrupted parties' shares of $\langle o'' \rangle$ to D. Then $\mathtt{Sim}$ checks if $\sum_{i \in \mathcal{C}_{\mathtt{active}}} \langle o'' \rangle$ sent by active corrupted parties equals $\sum_{i \in \mathcal{C}_{\mathtt{active}}} \langle o'' \rangle$ actually held by active corrupted parties. If so, $\mathtt{Sim}$ learns that D broadcasts that $\langle o'' \rangle$ is valid. Otherwise, $\mathtt{Sim}$ learns that D broadcasts that $\langle o'' \rangle$ is invalid. Next $\mathtt{Sim}$ simulates the byzantine agreement protocol according to what is broadcast by D.

iii. In Step 4 of $\Pi_{\mathsf{FaultDetection}}$, $\mathtt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{RandParity}}$.
If D is corrupted, note that $\mathtt{Sim}$ has already sample s in Step 3.(c). Then $\mathtt{Sim}$ samples the shares of $\langle\!|s|\!\rangle + [p]_t$ held by active honest parties that are compatible with the shares of $\langle\!|s|\!\rangle + [p]_t$ held by active corrupted parties and with s. $\mathtt{Sim}$ sends the shares of $\langle\!|s|\!\rangle + [p]_t$ held by active honest parties to $\mathcal{A}$.
If D is honest, $\mathtt{Sim}$ follows the protocol.

iv. In Step 5 of $\Pi_{\mathsf{FaultDetection}}$, if D is corrupted, $\mathtt{Sim}$ samples random additive zero sharings for $\{\langle \hat{o}_i \rangle\}_{i \in \mathcal{H}_{\mathtt{active}}}$, such that $\sum_{i \in \mathcal{H}_{\mathtt{active}}} \langle \hat{o}_i \rangle = \langle \hat{o} \rangle - \sum_{i \in \mathcal{C}_{\mathtt{active}}} \langle \hat{o}_i \rangle$, and $\langle \hat{o}_i \rangle$ is compatible with the simulated active corrupted parties' shares. For all $P_i \in \mathcal{H}_{\mathtt{active}}$, $\mathtt{Sim}$ gets $\hat{o}_i^{(j)}$ from the sampled $\langle \hat{o}_i \rangle$, and $\mathtt{Sim}$ gets $(\hat{o}_j^{(i)})'$ from the sampled sharing $\langle \hat{o}_j \rangle$ if $j \in \mathcal{H}_{\mathtt{active}}$, or from $\mathcal{A}$ if $j \in \mathcal{C}_{\mathtt{active}}$.
If D is honest, $\mathtt{Sim}$ follows the protocol.

v. In Step 6 of $\Pi_{\mathsf{FaultDetection}}$, if D is corrupted, since $\mathtt{Sim}$ has sampled all sharings, $\mathtt{Sim}$ can simply follow the protocol.
If D is honest, $\mathtt{Sim}$ receives from $\mathcal{A}$ all the incorrect or inconsistent messages. From these messages, $\mathtt{Sim}$ can determine D's broadcast message. Then $\mathtt{Sim}$ follows the byzantine agreement protocol to broadcast the message.

vi. In Step 7 of $\Pi_{\mathsf{FaultDetection}}$, if $(P_i, \texttt{incorrect})$ is broadcast by D, $\texttt{Sim}$ follows the protocol. If $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \texttt{inconsistent})$ is broadcast by D, $\texttt{Sim}$ can perfectly simulate if P_{j_1} or P_{j_2} agree as follows. If D is corrupted, since $\texttt{Sim}$ has sampled all sharings, $\texttt{Sim}$ follows the protocol to simulate if P_{j_1} or P_{j_2} agrees. If D is honest, $\texttt{Sim}$ knows that the honest party between P_{j_1} and P_{j_2} will always agree, and receives from $\mathcal{A}$ whether the corrupted party between P_{j_1} and P_{j_2} agrees. Then $\texttt{Sim}$ follows the byzantine agreement protocol.

5. In Step 4, $\texttt{Sim}$ simulates Π_{Output} as follows:
   (a) In Step 1 of Π_{Output}, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{Collect}}$.
   (b) In Step 2 of Π_{Output}, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{Permute}}$.
   (c) In Step 3 of Π_{Output}, if Client is corrupted, $\texttt{Sim}$ receives from $\mathcal{F}_{\mathsf{Main}}$ the output towards Client. Then $\texttt{Sim}$ samples the shares of the output Shamir sharings held by active honest parties that are compatible with the secrets and the active corrupted parties' shares, and then $\texttt{Sim}$ sends them to Client.
   If Client is honest, $\texttt{Sim}$ follows the protocol.

Hybrid Arguments. We show that $\texttt{Sim}$ perfectly simulates the behaviors of honest parties.

Hybrid$_0$: The execution in the real world.

Hybrid$_1$: In this hybrid $\texttt{Sim}$ faithfully emulates the functionalities $\mathcal{F}_{\mathsf{InputShamir}}$, $\mathcal{F}_{\mathsf{RandShamir}}$ in Step 1.(b) and Step 1.(c) of Π_{Input}. The distribution is the same as **Hybrid$_0$**.

Hybrid$_2$: In this hybrid $\texttt{Sim}$ simulates Step 1.(d) of Π_{Input}. If Client is corrupted, the distribution is the same since $\texttt{Sim}$ receives the true value of $\{y_i\}_{i=1}^N$ from $\mathcal{A}$, and samples from the same distribution for the secrets $\{\phi(\boldsymbol{r}_i)\}_{i=1}^N$. If Client is honest, since Client's input will always be valid and has the form of $y_i = \phi(\boldsymbol{x}_i)$ for all $i \in [N]$, we have the distribution of $y_i + \phi(\boldsymbol{r}_i)$ is the same as the distribution that $\texttt{Sim}$ samples from for all $i \in [N]$. Therefore, the distribution is the same as **Hybrid$_1$**.

Hybrid$_3$: In this hybrid $\texttt{Sim}$ faithfully emulates the functionalities $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ in Step 2 and Step 3 of Π_{Input}. The distribution is the same as **Hybrid$_2$**.

Hybrid$_4$: $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{RandZeroAdd}}$ in Step 2 of Π_{Eval}, and $\mathcal{F}_{\mathsf{TripleAdd}}$ and $\mathcal{F}_{\mathsf{TripleMult}}$ in Step 3 of Π_{Eval}. The distribution is the same as **Hybrid$_3$**.

Hybrid$_5$: In this hybrid, $\texttt{Sim}$ simulates Step 5.(b) and 5.(c) of Π_{Eval} when D is corrupted. Note that the shares of $\langle x \rangle + \langle a_j \rangle + \langle o_1 \rangle$ and $\langle y \rangle + \langle b_j \rangle + \langle o_2 \rangle$ held by active corrupted parties are independent of the shares of $\langle x \rangle + \langle a_j \rangle + \langle o_1 \rangle$ and $\langle y \rangle + \langle b_j \rangle + \langle o_2 \rangle$ held by active honest parties and the secrets $x + a_j$, $y + b_j$. This is because $x + a_j$ and $y + b_j$ are uniformly random given the shares of $[\phi(\boldsymbol{a})]_t$ and $[\phi(\boldsymbol{b})]_t$ held by active corrupted parties, and because additive sharings have threshold t. Note that in Step 5.(c) $\texttt{Sim}$ follows the protocol, which will not change the distribution. Therefore, the distribution is the same as **Hybrid$_4$**.

Hybrid$_6$: In this hybrid, $\texttt{Sim}$ faithfully emulates the functionalities among Step 2 to Step 5 of Π_{Verify} when D is corrupted. Specifically, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{VerifyBC}}$ in Step 2 of Π_{Verify}, $\mathcal{F}_{\mathsf{Mult}}$ and $\mathcal{F}_{\mathsf{ReEncode}}$ in Step 4 of Π_{Verify}, and $\mathcal{F}_{\mathsf{FanOut}}$, $\mathcal{F}_{\mathsf{Permute}}$, $\mathcal{F}_{\mathsf{Collect}}$, and $\mathcal{F}_{\mathsf{Permute}}$ in Step 5 of Π_{Verify}. The distribution is the same as **Hybrid$_5$**.

Hybrid$_7$: In this hybrid, $\texttt{Sim}$ simulates Step 9 of Π_{Verify} when D is corrupted. Because $\texttt{Sim}$ has sampled the entire sharing of $\langle x \rangle + \langle a_j \rangle + \langle o_1 \rangle$ and $\langle y \rangle + \langle b_j \rangle + \langle o_2 \rangle$ when simulating Step 3.(c) of Π_{Eval}, $\texttt{Sim}$ can locally generate the correct values $\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)}$ and $\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)}$ for all $i \in [N]$. Therefore, the distribution of $\phi(\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)})$ is the same as receiving the reconstructed value from $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$. It follows that the distribution of this hybrid is the same as **Hybrid$_6$**.

Hybrid$_8$: In this hybrid, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ in Step 10.(a) of Π_{Verify} when D is corrupted. The distribution is the same as **Hybrid$_7$**.

Hybrid$_9$: In this hybrid, $\texttt{Sim}$ simulates Step 1 of $\Pi_{\mathsf{FaultDetection}}$ when D is corrupted. Because additive zero sharings have threshold t, the shares of $\langle o_i'' \rangle$ held by active corrupted parties are uniformly random when P_i is honest, Since $\texttt{Sim}$ samples active corrupted parties' shares uniformly randomly, the distribution of this hybrid is the same as **Hybrid$_8$**.

Hybrid$_{10}$: In this hybrid, $\texttt{Sim}$ simulates Step 2 of $\Pi_{\mathsf{FaultDetection}}$ when D is corrupted. Consider the zero additive sharing $\langle o_i'' \rangle$ for an honest party P_i. Its shares held by active honest parties are independent of its shares held by active corrupted parties. It follows that the shares of $\sum_{i \in \mathcal{H}_{\mathsf{active}}} \langle o_i'' \rangle$ held by active honest parties are independent of the shares of $\langle o_i'' \rangle$ held by active corrupted parties

for all $i \in \mathcal{H}_{\texttt{active}}$. So the distribution of $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle o_i''\rangle$ sampled by $\texttt{Sim}$ is identical to the distribution of $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle o_i''\rangle$ sampled by active honest parties. Therefore, the distribution of this hybrid is the same as $\mathbf{Hybrid_9}$.

$\mathbf{Hybrid_{11}}$: In this hybrid, $\texttt{Sim}$ simulates Step 4 of $\Pi_{\mathsf{FaultDetection}}$ when D is corrupted. $\texttt{Sim}$ first faithfully emulates $\mathcal{F}_{\mathsf{RandParity}}$ in Step 4 of $\Pi_{\mathsf{FaultDetection}}$. After this step, the distribution is the same as $\mathbf{Hybrid_{10}}$.

Suppose the sharing $(\!|s|\!) + [p]_t$ sampled by $\texttt{Sim}$ is denoted by $[w]_t$. We argue that $[w]_t$ has the same distribution as the real sharing $(\!|s|\!) + [p]_t$ as follows.

- Given the secret s, the sharing $\langle s\rangle + \langle o\rangle$ is independent of the sharing $(\!|s|\!)$ and is therefore independent of the sharing of $(\!|s|\!) + [p]_t$. It follows that the sharing $\langle s\rangle + \langle o\rangle$ sampled by $\texttt{Sim}$ is independent of the distribution of $(\!|s|\!) + [p]_t$.
- Note that $[p]_t$ is a uniformly random parity sharing given active corrupted parties' shares, and the sampled sharing $[w]_t$ is valid if and only if $[w]_t - (\!|s|\!)$ is a parity sharing. So $(\!|s|\!) + [p]_t$ has the same distribution as $[w]_t$.

Therefore, the distribution of this hybrid is the same as $\mathbf{Hybrid_{10}}$.

$\mathbf{Hybrid_{12}}$: In this hybrid, $\texttt{Sim}$ simulates Step 5, Step 6, and Step 7 of $\Pi_{\mathsf{FaultDetection}}$ when D is corrupted. If $\langle\hat{o}\rangle = \langle o''\rangle$, $\texttt{Sim}$ has sampled $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle o_i''\rangle$. If $\langle\hat{o}\rangle = \langle o\rangle - \langle o'\rangle + \langle o''\rangle$, note that $\texttt{Sim}$ has sampled $(\!|s|\!) + [p]_t$, $\texttt{Sim}$ can derive $\sum_{i\in\mathcal{H}_{\texttt{active}}}(\langle o_i\rangle - \langle o_i'\rangle + \langle o_i''\rangle)$ from the sampled sharings. In either case, $\texttt{Sim}$ has $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle\hat{o}_i\rangle$, and it is uniformly random given the shares of $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle\hat{o}_i\rangle$ held by active corrupted parties.

For $i \in \mathcal{H}_{\texttt{active}}$, $\langle\hat{o}_i\rangle$ is a uniformly random additive zero sharing given the shares held by active corrupted parties. Therefore, when $\texttt{Sim}$ uniformly samples $\langle\hat{o}_i\rangle$ for all $i \in \mathcal{H}_{\texttt{active}}$ given $\sum_{i\in\mathcal{H}_{\texttt{active}}}\langle\hat{o}_i\rangle$, the distribution is the same as $\mathbf{Hybrid_{11}}$.

Since $\texttt{Sim}$ follows the rest of Steps of $\Pi_{\mathsf{FaultDetection}}$, and the sharing distribution is the same, we have that the distribution of this hybrid is the same as $\mathbf{Hybrid_{11}}$.

$\mathbf{Hybrid_{13}}$: In this hybrid, $\texttt{Sim}$ simulates Step 5.(b) and Step 5.(c) of Π_{Eval} when D is honest. We have the secret $x + a_j$ and $y + b_j$ are uniformly random, and are independent of the shares of $\langle x\rangle + \langle a_j\rangle + \langle o_1\rangle$ and $\langle y\rangle + \langle b_j\rangle + \langle o_2\rangle$ held by active corrupted parties. Since $\texttt{Sim}$ samples $x + a_j$ and $y + b_j$ uniformly randomly, we have the distribution of this hybrid is the same as $\mathbf{Hybrid_{12}}$.

$\mathbf{Hybrid_{14}}$: In this hybrid, $\texttt{Sim}$ faithfully emulates the functionalities among Step 2 to Step 8 of Π_{Verify} when D is honest. Specifically, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{VerifyBC}}$ in Step 2 of Π_{Verify}, $\mathcal{F}_{\mathsf{Mult}}$ and $\mathcal{F}_{\mathsf{ReEncode}}$ in Step 4 of Π_{Verify}, $\mathcal{F}_{\mathsf{FanOut}}$ in Step 5 of Π_{Verify}, $\mathcal{F}_{\mathsf{Permute}}$ in Step 6 and Step 8 of Π_{Verify}, and $\mathcal{F}_{\mathsf{Collect}}$ in Step 7 of Π_{Verify}. The distribution is the same as $\mathbf{Hybrid_{13}}$.

$\mathbf{Hybrid_{15}}$: In this hybrid, $\texttt{Sim}$ simulates Step 9 of Π_{Verify} when D is honest. Because the network routing is correct, and $\texttt{Sim}$ can receive from $\mathcal{A}$ the additive errors introduced to $\tilde{u}_j^{(i)}$, i.e. $u_j^{(i)} - \tilde{u}_j^{(i)}$, for all $i \in [N]$ and for all $j \in [\ell]$, $\texttt{Sim}$ correctly generates $\tilde{u}_j^{(i)}$ from the $u_j^{(i)}$ that it sampled when simulating Step 5.(d) of Π_{Eval}. Therefore, the distribution of $\phi(\tilde{\boldsymbol{x}}^{(i)} + \boldsymbol{a}^{(i)})$ and $\phi(\tilde{\boldsymbol{y}}^{(i)} + \boldsymbol{b}^{(i)})$ is the same as receiving the reconstructed value from $\mathcal{F}_{\mathsf{OpenPub}}$ for all $i \in [N]$. It follows that the distribution of this hybrid is the same as $\mathbf{Hybrid_{14}}$.

$\mathbf{Hybrid_{16}}$: In this hybrid, $\texttt{Sim}$ faithfully emulates $\mathcal{F}_{\mathsf{FanOut}}$ and $\mathcal{F}_{\mathsf{Permute}}$ in Step 10.(a) of Π_{Verify} when D is honest. The distribution is the same as $\mathbf{Hybrid_{15}}$.

$\mathbf{Hybrid_{17}}$: In this hybrid, $\texttt{Sim}$ simulates Step 1 of $\Pi_{\mathsf{FaultDetection}}$ when D is honest. Because additive zero sharings have threshold t, the shares of $\langle o_i''\rangle$ held by active corrupted parties are uniformly random when P_i is honest, Since $\texttt{Sim}$ samples active corrupted parties' shares uniformly randomly, the distribution of this hybrid is the same as $\mathbf{Hybrid_{16}}$.

$\mathbf{Hybrid_{18}}$: In this hybrid, $\texttt{Sim}$ simulates Step 2, Step 4, Step 5, Step 6, and Step 7 of $\Pi_{\mathsf{FaultDetection}}$ when D is honest. In Step 2 of $\Pi_{\mathsf{FaultDetection}}$, because $\texttt{Sim}$ can perfectly simulate the value broadcast by D, the distribution is the same. In Step 4 and Step 5 of $\Pi_{\mathsf{FaultDetection}}$, because $\texttt{Sim}$ follows the protocol, the distribution remains the same. In Step 6 of $\Pi_{\mathsf{FaultDetection}}$, because $\texttt{Sim}$ receives from $\mathcal{A}$ the incorrect or inconsistent message, $\texttt{Sim}$ can perfectly simulate D's broadcast message. In Step 7 of $\Pi_{\mathsf{FaultDetection}}$, if $(P_{j_1}, P_{j_2}, \hat{o}_{j_1}^{(j_2)}, (\hat{o}_{j_1}^{(j_2)})', \texttt{inconsistent})$ is broadcast, the honest party between P_{j_1} and P_{j_2} will always agree, and the other party is the corrupted party. So $\texttt{Sim}$ can perfectly simulate the message broadcast by P_{j_1} and P_{j_2}. Above all, the distribution of this hybrid is the same as $\mathbf{Hybrid_{17}}$.

Hybrid$_{19}$: In this hybrid, Sim faithfully emulates the functionalities $\mathcal{F}_{\mathsf{Collect}}$ and $\mathcal{F}_{\mathsf{Permute}}$ in Step 1 and Step 2 of Π_{Output}. The distribution is the same as **Hybrid$_{18}$**.

Hybrid$_{20}$: In this hybrid, Sim simulates Step 3 of Π_{Output}. Suppose the output sharings for Client are $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$. If Client is corrupted, it is guaranteed by $\mathcal{F}_{\mathsf{Permute}}$ that the shares of $[\phi(\boldsymbol{z}_1)]_t, \ldots, [\phi(\boldsymbol{z}_N)]_t$ held by active honest parties are uniformly sampled given the secrets $\phi(\boldsymbol{z}_1), \ldots, \phi(\boldsymbol{z}_N)$ and the shares held by active corrupted parties. Therefore, the shares held by active honest parties sampled by Sim have the same distribution as the real shares held by active honest parties in the view of $\mathcal{A}$. If Client is honest, since Sim follows the protocol, the distribution remains the same. Therefore, the distribution of this hybrid is the same as **Hybrid$_{19}$**.

Note that this hybrid is the execution in the ideal world.

G List of Functionalities and Protocols

Functionalities

Protocols