Honest Majority GOD MPC with $O(\mathsf{depth}(C))$ Rounds and Low Online Communication

Amit Agarwal[1], Alexander Bienstock[2], Ivan Damgård[3], and Daniel Escudero[2]

[1] University of Illinois Urbana-Champaign
[2] J.P. Morgan AI Research and J.P. Morgan AlgoCRYPT CoE
[3] Aarhus University

Abstract. In the context of secure multiparty computation (MPC) protocols with guaranteed output delivery (GOD) for the honest majority setting, the state-of-the-art in terms of communication is the work of (Goyal *et al.* CRYPTO'20), which communicates $O(n|C|)$ field elements, where $|C|$ is the size of the circuit being computed and n is the number of parties. Their round complexity, as usual in secret-sharing based MPC, is proportional to $O(\mathsf{depth}(C))$, but only in the optimistic case where there is no cheating. Under attack, the number of rounds can increase to $\Omega(n^2)$ before honest parties receive output, which is undesired for shallow circuits with $\mathsf{depth}(C) \ll n^2$. In contrast, other protocols that only require $O(\mathsf{depth}(C))$ rounds *even in the worst case* exist, but the state-of-the-art from (Choudhury and Patra, Transactions on Information Theory, 2017) still requires $\Omega(n^4|C|)$ communication in the offline phase, and $\Omega(n^3|C|)$ in the online (for both point-to-point and broadcast channels). We see there exists a tension between efficient communication and number of rounds. For reference, the recent work of (Abraham et al., EUROCRYPT'23) shows that for perfect security and $t < n/3$, protocols with both linear communication and $O(\mathsf{depth}(C))$ rounds exist.

We address this state of affairs by presenting a novel honest majority GOD protocol that maintains $O(\mathsf{depth}(C))$ rounds, *even under attack*, while improving over the communication of the most efficient protocol in this setting by Choudhury and Patra. More precisely, our protocol has point-to-point (P2P) online communication of $O(n|C|)$, accompanied by $O(n|C|)$ broadcasted (BC) elements, while the offline has $O(n^3|C|)$ P2P communication with $O(n^3|C|)$ BC. This improves over the previous best result, and reduces the tension between communication and round complexity. Our protocol is achieved via a careful use of packed secret-sharing in order to improve the communication of existing verifiable secret-sharing approaches, although at the expense of weakening their robust guarantees: reconstruction of shared values may fail, but only if the adversary gives away the identities of many corrupt parties. We show that this less powerful notion is still useful for MPC, and we use this as a core building block in our construction. Using this weaker VSS, we adapt the recent secure-with-abort Turbopack protocol (Escudero *et al.* CCS'22) to the GOD setting without significantly sacrificing in efficiency.

1 Introduction

Secure multiparty computation, or MPC for short, is a set of tools that enable a set of parties $P_1, \ldots, P_n$, each P_i holding an input x_i, to compute any function $y = f(x_1, \ldots, x_n)$ while revealing only the output y. This holds even if an adversary corrupts t out of the n parties, and if the protocol is what is known as *actively secure*, then security holds even if the corrupt parties deviate arbitrarily from the protocol execution. A particularly interesting setting, which is the focus of this work, is the *honest majority* case where it is assumed that $t < n/2$, that is, the adversary only corrupts a minority of the parties. In this case, the strong property of *guaranteed output delivery* (G.O.D.) can be achieved, which ensures the honest parties receive the output y in spite of any malicious behavior by the corrupt parties. Furthermore, if one is willing to assume the existence of a broadcast channel, the whole protocol can be made unconditionally secure, meaning it is resistant against unbounded adversaries and it only has a negligible amount of error. This is in contrast to *security with abort*, which ensures privacy but may not guarantee output provision.

G.O.D. is the strongest security notion for MPC and very important in practice as it removes all problems with deciding who failed if the protocol aborts. Hence, optimizing the efficiency of G.O.D. protocols has been a topic of study in recent literature. First, as usual in MPC, let us model the computation to be carried out as an arithmetic circuit C over a large-enough finite field $\mathbb{F}$. Let us begin by discussing the case in which $t < n/3$, which is *weaker* than honest majority $t < n/2$ as it tolerates less corruptions. Earlier works such as BGW [BGW88] and (concurrently) [CCD88] show that G.O.D. is possible for $t < n/3$ with perfect security, having a communication complexity (measured in the total number of field elements) of $O(n^4|C|)$. Several follow-up works were devoted to improving the communication. Towards such goal, [HMP00] introduces the *player elimination* framework, which reduces communication to $O(n^3|C|)$, and building on top of this framework, the works of [BTH08; GLS19] get linear communication $O(n|C|)$. Unfortunately, the number of rounds—which is also an important metric directly related to end-to-end performance—of protocols based on player elimination is proportional to $O(\mathsf{depth}(C))$, but only in the *optimistic case* where there is no cheating. When the corrupt parties misbehave, the protocol enters a "recovery state" and eventually resumes after a previous "checkpoint". This process can happen $\Omega(n)$ times, and overall it adds $\Omega(n)$ rounds to the round-count, which is particularly impactful when $\mathsf{depth}(|C|) = o(n)$. For high latency settings such as wide area networks this can drastically affect the performance of the protocol. Studying the communication of $t < n/3$ MPC with $O(\mathsf{depth}(C))$ rounds (independently of n, even in the worst case) has been a topic of study of recent works: [AAY21] achieved $O(n^3|C|)$ communication, and the recent work of [Abr+23] lowered it to $O(n|C|)$, matching (asymptotically) that of the previous state-of-the-art [BTH08; GLS19], but crucially, without paying any additive overhead in n on the number of rounds.

The above discussion is for the $t < n/3$ setting, which is considerably easier than the $n/3 < t < n/2$ regime. For $t < n/2$, it is known that G.O.D. is possible

assuming a broadcast channel, and the early work by Rabin and Ben-Or [RB89] achieved $O(\mathsf{depth}(C))$ rounds even in the worst case, but it is very expensive in terms of communication. For instance, its communication is higher than that of [Cra+99], which is $O(n^5|C|) + O(n^3) \times \mathsf{BC}$, where the term multiplying the BC denotes the number of elements sent over the broadcast channel. The work of [Cra+99] however has a number of rounds that, in the worst case, can become $O(n \cdot \mathsf{depth}(C))$. In the last two decades several subsequent works approached the task of improving communication [BFO12; BTH06; GSZ20], culminating in the current state-of-the-art by Goyal, Song, and Zhu [GSZ20], which has $O(n|C| + O(n^3) \times \mathsf{BC})$ communication. All these works require $O(\mathsf{depth}(|C|))$ rounds in the optimistic case, but in case of cheating, the "recovery state" adds not only $\Omega(n)$, but $\Omega(n^2)$ extra rounds,[4] which is extremely harmful for circuits with $\mathsf{depth}(C) \ll n^2$. Currently, and unlike the $t < n/3$, removing this round-count overhead can only be done at the expense of increasing communication. Indeed, the most recent work exploring the task of $O(\mathsf{depth}(C))$-round honest majority G.O.D. MPC is [CP17], which has a communication of $O(n^4|C|) + O(n^4|C|) \times \mathsf{BC}$ in total, with an online phase of $O(n^3|C|) + O(n^3|C|) \times \mathsf{BC}$.

Towards improving this state of affairs, recent work by Escudero and Fehr [EF21] presents an honest majority G.O.D. protocol that achieves $O(\mathsf{depth}(C))$ rounds, independent of n, and simultaneously has linear communication $O(n|C|) + 0 \times \mathsf{BC}$. However, this is done in the *preprocessing model* where the parties are assumed to have certain correlated randomness independent of their inputs available for free. When instantiating this preprocessing with a protocol that has number of rounds independent of n—like the one from [CP17]—the resulting preprocessing complexity would be $O(|C|n^6) + O(|C|n^6) \times \mathsf{BC}$, which is too large. This situation leads to the following interesting and challenging question:

How communication-efficient can honest majority G.O.D. protocols be, while using only $O(\mathsf{depth}(|C|))$ rounds even under attack (that is, independent of n)?

1.1 Our Contribution

In this work we make progress on this question by providing an MPC protocol in the honest majority case $t < n/2$ that has $O(\mathsf{depth}(C))$ rounds even in the worst case, while improving over the communication of the state-of-the-art [CP17] in this regime. We achieve a communication of $O(n|C| + \mathsf{depth}(C)n^3) + O(n|C| + \mathsf{depth}(C)n^3) \times \mathsf{BC}$ in the online phase (a factor of n^2 improvement), and for the offline phase our communication is $O(n^3|C|) + O(n^3|C|) \times \mathsf{BC}$ (a factor of n improvement). Table 1 presents our communication in relation to the work of [CP17], as well as the previous works that achieved G.O.D. with an amount of rounds proportional to $\mathsf{depth}(C)$, independent of n.

Interestingly, for the *online phase*, our protocol matches the *peer-to-peer* communication of the best-known protocol [GSZ20], which is linear $O(n|C|)$ (also

[4] In $t < n/2$ the recovery is done with a technique called *dispute control* [BTH06], which is repeated n^2 times in the worst case, in contrast to *player elimination*—only suitable for $t < n/3$—which is repeated n times.

Work	Offline Comm.	Online Comm.
[RB89]	N/A	$\omega(\|C\|n^5) + \omega(\|C\|n^5) \times \mathsf{BC}$
[CP17]	$O(\|C\|n^4) + O(\|C\|n^4) \times \mathsf{BC}$	$O(\|C\|n^3) + O(\|C\|n^3) \times \mathsf{BC}$
[EF21]	$O(\|C\|n^5) + O(\|C\|n^5) \times \mathsf{BC}$ *	$O(\|C\|n + \mathsf{depth}(C)n^3) + 0 \times \mathsf{BC}$
Ours	$O(\|C\|n^3) + O(\|C\|n^3) \times \mathsf{BC}$	$O(\|C\|n + \mathsf{depth}(C)n^3) +$ $O(\|C\|n + \mathsf{depth}(C)n^3) \times \mathsf{BC}$

Table 1. Works with G.O.D. for $t < n/2$, and $O(\mathsf{depth}(C))$ rounds (independent of n, even in the worst case). "N/A" in the offline phase means these works did not consider an offline/online separation. We ignore $\mathsf{poly}(n)$ terms that do not depend on C. * The work of [EF21] does not instantiate the offline phase. The complexity reported is obtained by calculating the cost of generating their preprocessing using our protocol. For this, we take their preprocessing size, $n^2|C|$, and multiply it by the total communication of our protocol, leading to $O(n^3 \cdot n^2|C|)$.

believed to be optimal [DLN19]).[5] This constitutes significant progress in the direction of matching the communication of G.O.D. protocols for $t < n/2$ with $O(\mathsf{depth}(C))$ rounds, like ours, with protocols that add $O(n^2)$ rounds in the worst case, like [GSZ20]. We recall that for $t < n/3$ and perfect security, the task of designing protocols with $O(\mathsf{depth}(C))$ rounds whose communication matches that of $O(\mathsf{depth}(C) + n)$-round protocols was only recently settled in [Abr+23].

Remark 1 (On the term $\mathsf{depth}(C) \cdot n^3$). Our online communication is $O(n|C| + \mathsf{depth}(C)n^3) + O(n|C| + \mathsf{depth}(C)n^3) \times \mathsf{BC}$. Note that the term $\mathsf{depth}(C)n^3$ is absorbed by $|C|n$, as long as $|C|/\mathsf{depth}(C) = \Omega(n^2)$, in which case the communication becomes $O(|C|n) + O(|C|n) \times \mathsf{BC}$. This can be satisfied for instance if the circuit has uniform width $\Theta(n^2)$, but this is not strictly necessary: for example, a few layers can have a very small amount of multiplication gates (say $o(n)$), while some others may have many more gates (say $\approx n^3$), and the property $|C|/\mathsf{depth}(C) = \Omega(n^2)$ may still be satisfied.

1.2 Other Related Work

We have already discussed the works of [BFO12; BTH06; CP17; Cra+99; EF21; GSZ20]—which are the works most related to us—as well as their comparison with respect to our work. We present in Section A in the Supplementary Material a more detailed description of how these protocols work. As other related literature, an important mention is [Ish+16], which presents a series of compilers which, among other things, enable "upgrading" secure-with-abort protocols into G.O.D.. Their approach also results in a protocol whose round complexity depends on n, since it consists of identifying corrupt parties and then re-running certain parts of the protocol.

[5] Note that our term $O(|C|n) \times \mathsf{BC}$ (which is not present in [GSZ20]) would require all parties to *receive* at least $n|C|$ messages, which in practice means a communication of at least $n^2|C|$, widening the gap between the protocol from [GSZ20] and ours. See also Remark 1.

Using information-theoretic randomized encodings [IK00; IK02], it is possible to achieve statistically secure MPC in the $n = 2t + 1$ setting with *constant* round complexity [AKP23] (i.e. the round complexity is independent of both n and $\mathsf{depth}(C)$) where the constant is 4. This is akin to how Garbled Circuits (which are an instance of *computationally* secure randomized encoding) enable us to achieve constant round complexity in the computational setting. In such protocols, the communication complexity is always proportional to the size of the randomized encoding. With current known techniques, the size of information-theoretic randomized encoding grows exponentially with the circuit depth and reducing this exponential dependency has been a big open problem for the past two decades. Hence, this approach of using randomized encodings to get low round complexity is currently practical only for NC^1 circuits.

1.3 Overview of our Techniques

We discuss at a very high level how our final protocol is achieved. First, to avoid the extra n^2 rounds, we must deviate from the dispute-control paradigm used in all communication-efficient works [BFO12; BTH06; GSZ20]. Instead, let us take as a starting point the protocol from [CP17, Section VI], which is more communication heavy but removes the round dependency on n. In [CP17] the authors make use of the verifiable secret-sharing (VSS) ideas from [Cra+99; RB89], which enable parties to obtain sharings $[s]$ of a secret in such a way that the honest parties can always reconstruct the given secret s, using a *constant* number of rounds. The authors make use of multiplication triples [Bea92], produced in an offline phase. With this preprocessing at hand, the online phase is comprised only of linear combinations and reconstruction of secret-shared data, which is possible thanks to the VSS guarantees. The main contribution of [CP17] lies in the generation of the multiplication triples. Traditionally, the most efficient approach for triple generation is the first generating so-called double-sharings [DN07], which can be later used to obtain triples. However, this approach requires more than what VSS can provide: there are certain proofs of correctness that the parties must perform, which ultimately add substantially to the final costs (intuitively, these complexities come from local multiplication increasing the degree from t to $2t$). Instead, the work of [CP17] introduces a novel approach which only relies on the basic properties of VSS, letting each party contribute with triples directly, which are later checked for correctness and "combined" in such a way that truly random triples are produced. Intuitively, this only relies on sharing, linear combinations, and reconstructions, and hence can be handled by the underlying VSS alone. Now, this approach is *less* communication-efficient than using double-sharings, but it is much more suitable for reducing the number of rounds. From the above, in order to improve the communication complexity of [CP17], which maintains the number of rounds we are looking for, it is imperative to improve that of the underlying VSS.

Optimizing—but weakening—verifiable secret-sharing. We start from the VSS used in [CP17], which is that from [RB89] in conjunction with the

so-called information-checking protocol (ICP) from [Cra+99]. Intuitively, these techniques involve distributing Shamir sharings, and additionally, distributing shares of each share to the parties. The shares-of-shares exist so that, when a party announces a share, it can prove to the others this is correct by also showing them the shares of the announced share, which the parties can contrast with the share-of-share they have internally. Now, a corrupt party may complain of a correctly announced share, and to prevent this the ICP machinery is used: that ensures that (1) honest parties can always prove the correctness of their shares, and (2) corrupt parties who modify their shares are identified. The degree of the polynomials is t, which requires $t+1$ shares to be reconstructed. By using the "signatures", the $\leq t+1$ correct shares coming from the honest shares can be identified, which allows for the reconstruction of the secret. Inspired by [Abr+23], our approach to improve the complexity of this VSS is to make use of *packed secret-sharing* [FY92], which allows for having $\ell \geq 1$ secrets instead of only one, without penalty in the communication costs. However, using packed secret-sharing comes at the expense of increasing the degree of the underlying polynomials from t to $t+(\ell-1)$, and in particular reconstruction now requires more shares than honest parties. This is not a problem in [Abr+23], which is set in $t < n/3$, but in our $t < n/2$ regime extra care is needed.

We use packed secret-sharing twice, resulting in packed vectors of dimension $\Theta(n^2)$. First, we use degree $(t+(\ell-1))$ to secret-share $\ell = \Theta(n)$ secrets at once (we will specify the exact value of ℓ), but crucially, we use degree-t for the "shares of shares", which ensures the $t+1$ honest parties alone still have enough joint information to reconstruct the secrets. Now, each share-of-share is signed using the ICP from [Cra+99], which at a high level works by secret-sharing the message to be signed towards the parties, so that later on when this message is revealed, the parties can jointly verify if this is consistent with the shares they hold. In [Cra+99] degree-t polynomials are used, but a useful observation from [PR10] is that this can be improved by using packed secret-sharing, signing multiple messages towards multiple verifiers simultaneously. We adapt their ideas to our setting. This requires a batch of $m = \Theta(n)$ shares to be signed, each of which corresponds to $\ell = \Theta(n)$ secrets, so overall our VSS works on vectors of dimension $m\ell = \Theta(n^2)$.

Finally, an important remark is that our construction does not directly instantiate the notion of VSS. Increasing the degree from t to $t+(\ell-1)$ or, as we will see, $t+2(\ell-1)$ in some cases, comes at the expense of corrupt parties being able to disrupt the reconstruction if they decide to misbehave or abort. However, we still guarantee the crucial property that such corrupt parties are identified. We call this weaker notion *detectable secret-sharing* (DSS), and one of our core contributions, on top of the formal definition of such primitive as a UC functionality and its efficient instantiation, is showing that this weaker form of VSS can still be useful for our goal of MPC with $O(\mathsf{depth}(C))$ rounds. We discuss this below.

1.4 Our MPC Protocol

Since our core secret-sharing primitive operates on vectors instead of individual values, we cannot make direct use of the MPC approach from [CP17], which follows the standard Beaver triple paradigm [Bea92]. Instead, we adapt the ideas from the recent Turbopack work by Escudero et al. [Esc+22], which shows how to efficiently make use of packed secret-sharing, supporting arbitrary circuits without noticeable overhead. The details are provided in Section 5, but at a high level, there are two main components required in Turbopack. In the offline phase, the main challenge is generating packed multiplication triples, while in the online phase, the main challenge is reconstructing degree-$(t + 2(\ell - 1))$ secrets. For the first part, we show how to adapt the triple extraction ideas from [CP17], which quite surprisingly turn out to also work for the packed secret-sharing regime. The most interesting and challenging part is addressing the degree-$(t + 2(\ell - 1))$ reconstructions.

Recall that in our DSS, the adversary may completely halt the reconstruction of degree-$(t + 2(\ell - 1))$ secrets. This is because, even though as in [Cra+99; RB89] our scheme guarantees that honest parties can convince the others that their announced shares are correct, and also that corrupt parties announcing incorrect shares are identified, given that there are only $t + 1$ honest parties, only $t + 1$ out of the $t + 2(\ell - 1) + 1$ shares needed for reconstruction are guaranteed to be announced. That is, there could be $2(\ell - 1)$ shares missing. Our core observation is the following. If the t corrupt parties collectively send less than these $2(\ell - 1)$ shares, hence halting reconstruction, it is because more than $t - 2(\ell - 1)$ cheaters misbehaved, and crucially, their identities become known due to the properties of our DSS. At this point these parties can be removed, restarting the computation with threshold $t' < t - (t - 2(\ell - 1))$ and total number of parties $n' < n - (t - 2(\ell - 1))$. This time, the corruption ratio is t'/n', and it turns out we can upper bounded it by $1/3$ as long as we take $\ell \leq \frac{n+6}{8}$. The rest of the protocol is now set in the $t' < n'/3$ regime, point in which we can apply any existing work for that threshold to finish the computation. In particular, we can use the recent work of [Abr+23], which achieves perfect security but most importantly, requires linear communication and has no overhead in terms of the number of rounds.[6] Note that the adversary can only cause an abort-and-restart *once*, hence keeping the overall number of rounds $O(\mathsf{depth}(C))$.

There is a subtle issue when instantiating this novel idea. Restarting the protocol from scratch may allow the parties to change their inputs, which is not secure if in the first run the adversary was able to learn some information about the output. We propose two ways to address this, which perform differently depending on the amount of inputs versus amount of outputs. The first approach is more suitable if there are not many inputs, and consists of having the parties provide sharings of their inputs not only in our DSS scheme—which is packed and does not guarantee reconstruction—but also in the original VSS of [Cra+99;

[6] Furthermore, this protocol can presumably be optimized by avoiding the instantiation of the broadcast channel—which comes "for free" in our setting—and relaxing perfect security to statistical, but we find it to be unnecessary for our feasibility results.

RB89], while proving these two are consistent. In this way, if there is a restart, the parties can provide shares of the inputs for the $t' < n'/3$ protocol, while proving they hold the same secrets as the initial VSS sharings.

The second approach is more adequate if there are not many outputs. First, we note that it is fine to restart the computation while allowing the parties to change their inputs as long as this happens before the output phase, since in this case no sensitive leakage occurs. Now, we modify the output phase as follows: instead of attempting straight reconstruction of the degree-$(t + 2(\ell - 1))$ output sharings, the parties first convert the packed sharing of the outputs in the main protocol into a non-packed VSS representation. We design this conversion mechanism so that it does not leak anything about the outputs in case it aborts. This way, there are two potential outcomes: either the conversion succeeds, point in which the outputs are VSS'ed and then can be reconstructed with no abort, or the conversion fails, which does not leak anything and hence it is fine to restart the computation with $t' < n'/3$, even if the parties change their inputs. Details on these protocols are given in Sections 5 and C.

2 Preliminaries

Let κ be a statistical security parameter. We consider a finite field $\mathbb{F}$ with $|\mathbb{F}| = \omega(\mathtt{poly}(\kappa))$, so that $\mathtt{poly}(\kappa)/|\mathbb{F}| = \mathtt{negl}(\kappa)$. Given two strings x and y, we denote by $x\|y$ the concatenation of the two. We denote length-ℓ vectors as $\boldsymbol{v} = (v^1, \ldots, v^\ell)^\intercal \in \mathbb{F}^\ell$. Given two vectors $\boldsymbol{u} = (u^1, \ldots, u^\ell)^\intercal, \boldsymbol{v} = (v^1, \ldots, v^\ell)^\intercal \in \mathbb{F}^\ell$, we define $\boldsymbol{u} * \boldsymbol{v} = (u^1 \cdot v^1, \ldots, u^\ell \cdot v^\ell)^\intercal$ as the component-wise multiplication of $\boldsymbol{u}$ and $\boldsymbol{v}$. We denote $m \times n$ matrices as $\boldsymbol{M} \in \mathbb{F}^\ell$. For any given $C \subseteq [n]$, we denote by $\boldsymbol{M}^C$ the sub-matrix of $\boldsymbol{M}$ corresponding to the columns with indices C. We study MPC amongst n parties in the setting where the number of corrupted parties is exactly t with $n = 2t + 1$. We denote the set of honest parties as $\mathsf{Hon} \subseteq [n]$ and the set of corrupted parties as $\mathsf{Corr} \subseteq [n]$. We say that a protocol has communication complexity $\mathsf{P2P}(M) + N \times \mathsf{BC}(L)$ if it sends M field elements in total over the peer-to-peer channels, and it calls the broadcast channel N times with messages containing L field elements. A degree-d univariate polynomial over $\mathbb{F}$ is of the form $f(x) = \sum_{i=0}^{d} c_i x^i$, where $c_i \in \mathbb{F}$. We say that a collection of field elements $z_{i_1}, \ldots, z_{i_m}$ for $m > d$ and unique $i_1, \ldots, i_m \in \mathbb{F}$ is consistent with a degree-d polynomial, if there exists some degree-d polynomial $f(x)$ such that $f(i_j) = z_{i_j}$ for $i_j \in [m]$. A degree-(d_x, d_y) bivariate polynomial over $\mathbb{F}$ is of the form $F(x, y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} c_{i,j} x^i y^j$ where $c_{i,j} \in \mathbb{F}$. For $i \in \mathbb{F}$, we can isolate univariate polynomials $f_i(x) = F(x, i)$ and $g_i(y) = F(i, y)$ of degree d_x and d_y respectively. In this paper, we will always assume that $d_x = \max\{d_x, d_y\}$.

2.1 Bivariate Polynomials

We now present some lemmas regarding bivariate polynomials. We borrow ideas from [AL17] for the proofs. We begin by showing that $d_y + 1$ sets of points that define degree-d_x univariate polynomials $f_k(x)$ (for $k \in K$ of size $|K| = d_y + 1$)

uniquely determine a degree-(d_x, d_y) bivariate polynomial $F(x, y)$ such that $F(x, k) = f_k(x)$ for all $k \in K$.

Lemma 1. *Let $n > d_x \geq d_y \in \mathbb{N}$ and $J, K \subseteq [n]$ be such that $|J| \geq d_x + 1$ and $|K| = d_y + 1$. Let $\{z^{jk} \in \mathbb{F} : j \in J, k \in K\}$ be a set of field elements such that for each $k \in K$, $\{z^{jk}\}_{j \in J}$ are consistent with a degree-d_x univariate polynomial $f_k(x)$ such that $f_k(j) = z^{jk}$ for each $j \in J$. Then there exists a unique degree-(d_x, d_y) bivariate polynomial $F(x, y)$ such that $F(j, k) = z^{jk}$, for $j \in J, k \in K$.*

Proof. Define the bivariate polyonimal $F(x, y)$ via Lagrange interpolation:

$$F(x, y) = \sum_{i \in K} f_i(x) \cdot \frac{\prod_{j \in K \setminus \{i\}} (y - j)}{\prod_{j \in K \setminus \{i\}} (i - j)}. \tag{1}$$

It is easy to see that $F(x, y)$ has degree (d_x, d_y). Moreover, for every $k \in K$ it holds that:

$$\begin{aligned} F(x, k) &= \sum_{i \in K} f_i(x) \cdot \frac{\prod_{j \in K \setminus \{i\}} (k - j)}{\prod_{j \in K \setminus \{i\}} (i - j)} \\ &= f_k(x) \cdot \frac{\prod_{j \in K \setminus \{k\}} (k - j)}{\prod_{j \in K \setminus \{k\}} (k - j)} + \sum_{i \in K \setminus \{k\}} f_i(x) \cdot \frac{\prod_{j \in K \setminus \{i\}} (k - j)}{\prod_{j \in K \setminus \{i\}} (i - j)} \\ &= f_k(x) + 0 \\ &= f_k(x), \end{aligned}$$

and therefore that $F(j, k) = f_k(j) = z^{jk}$, for each $j \in J$, as desired.

It remains to show that $F(x, y)$ is unique. Assume that there exists $F'(x, y)$ that also satisfies the lemma statement. Define the polynomial

$$R(x, y) \leftarrow F(x, y) - F'(x, y) = \sum_{i=0}^{d_x} \sum_{j=0}^{d_y} r_{i,j} x^i y^j.$$

We will now show that $R(x, y) = 0$. First, for every $k \in K$ it holds that:

$$R(x, k) = \sum_{i,j=0}^{t} r_{i,j} x^i k^j = F(x, k) - F'(x, k) = f_k(x) - f_k(x) = 0,$$

by assumption. We can rewrite the univariate polynomial $R(x, k)$ as

$$R(x, k) = \sum_{i=0}^{x_x} \left(\left(\sum_{j=0}^{d_y} r_{i,j} k^j \right) \cdot x^i \right).$$

As we have seen, $R(x, k) = 0$. Thus, its coefficients are all zeroes, implying that for every fixed $i \in [0, d_x]$ it holds that $\sum_{j=0}^{d_y} r_{i,j} k^j = 0$. This in turn implies that

for every fixed $i \in [0, d_x]$, the polynomial $h_i(y) = \sum_{j=0}^{d_y} r_{i,j} y^j$ is zero for $d_y + 1$ points ($k \in K$), and so $h_i(y)$ is also the zero polynomial. Thus, its coefficients $r_{i,j}$ are equal to 0 for every $j \in [0, d_y]$. This holds for every fixed i, and therefore for every $i \in [0, d_x], j \in [0, d_y]$, we have that $r_{i,j} = 0$. We conclude that $R(x, y) = 0$ and hence $F(x, y) = F'(x, y)$. □

Now we show a corollary, which extends Lemma 1, by expanding K to be of size at least $d_y + 1$ (instead of exactly $d_y + 1$), and requiring that the sets of points $\{z^{jk}\}_{k \in K}$ are consistent with a degree-d_y polynomial $g_j(y)$, for each $j \in J$.

Corollary 1. *Let $n > d_x \geq d_y \in \mathbb{N}$ and $J, K \subseteq [n]$ be such that $|J| \geq d_x + 1$ and $|K| \geq d_y + 1$. Let $\{z^{jk} \in \mathbb{F} : j \in J, k \in K\}$ be a set of field elements such that:*

1. *For each $k \in K$, $\{z^{jk}\}_{j \in J}$ are consistent with a degree-d_x univariate polynomial $f_k(x)$ such that $f_k(j) = z^{jk}$ for each $j \in J$; and*
2. *For each $j \in J$, $\{z^{jk}\}_{k \in K}$ are consistent with a degree-d_y univariate polynomial $g_j(y)$ such that $g_j(k) = z^{jk}$ for each $k \in K$.*

Then there exists a unique degree-(d_x, d_y) bivariate polynomial $F(x, y)$ such that $F(j, k) = z^{jk}$, for $j \in J, k \in K$.

Proof. Let L be any subset of K of cardinality exactly $d_y + 1$. By Lemma 1, there exists a unique degree-(d_x, d_y) bivariate polynomial $F(x, y)$ such that $F(j, l) = z^{jl}$ for $j \in J, l \in L$. We now show that for all $j \in J$, $g_j(y) = F(j, y)$.

By definition of $g_j(y)$, we have that $g_j(l) = z^{jl}$. Furthermore, by the definition of F from above, we have that $z^{jl} = F(j, l)$. Thus, for all $j \in J$ and $l \in L$ it holds that $g_j(l) = F(j, l)$. Since both $g_j(y)$ and $F(j, y)$ are degree-d_y polynomials, and $g_j(l) = F(j, l)$ for $d_y + 1$ points l, it follows that $g_j(y) = F(j, y)$ for every $j \in J$.

Therefore, for every $j \in J, k \in K$, $F(j, k) = g_j(k) = z^{jk}$. This concludes the proof. □

Next, we show that for a random degree-(d_x, d_y) bivariate polynomial $F(x, y)$ such that $F(-l + 1, 0) = s^l$ for $l \in [\ell]$, no information about $\boldsymbol{s} = (s^1, \ldots, s^\ell)$ is revealed given the set of points $\{(F(j, k), F(k, j)\}_{j \in T, k \in [n]}$ for some $T \subseteq [n]$ such that $|T| \leq d_y \leq d_x - \ell + 1$.

Lemma 2. *Let $1 \leq \ell \leq d_y \leq d_x - \ell + 1 < n$ and $T \subseteq [n]$ be such that $|T| \leq d_y$. Let $\boldsymbol{s}_1, \boldsymbol{s}_2$ be any two vectors of ℓ elements. Then*

$$\mathcal{D}_1 \coloneqq \{(z^{jk}, z^{kj})\}_{j \in T, k \in [n]} \equiv \mathcal{D}_2 \coloneqq \{(\zeta^{jk}, \zeta^{kj})\}_{j \in T, k \in [n]},$$

where $F^1(x, y)$ and $F^2(x, y)$ are degree-(d_x, d_y) bivariate polynomials chosen at random under the constraints that $F^1(-l + 1, 0) = s_1^l$ and $F^2(-l + 1, 0) = s_2^l$, for $l \in [\ell]$, $z^{jk} \leftarrow F^1(j, k), z^{kj} \leftarrow F^1(k, j)$ and $\zeta^{jk} \leftarrow F^2(j, k), \zeta^{kj} \leftarrow F^2(k, j)$ for all $j \in T, k \in [n]$.

Proof. To show that this holds, we first show that for any set of pairs of elements $Z = \{(v^{jk}, v^{kj})\}_{j \in T, k \in [n]}$, the number of bivariate polynomials in the support

of $\mathcal{D}_1$ that are consistent with Z equals the number of bivariate polynomials in the support of $\mathcal{D}_2$ that are consistent with Z. By consistent, we mean that $v^{jk} = F(j,k)$ and $v^{kj} = F(k,j)$ for $j \in T, k \in [n]$.

First note that if there exists $j_1, j_2 \in T$ such that $(v^{j_1,j_2}, \cdot) \in Z$ and $(\cdot, u^{j_1,j_2}) \in Z$ for $v^{j_1,j_2} \neq u^{j_1,j_2}$, then there does not exist any bivariate polynomial in the support of $\mathcal{D}_1$ or $\mathcal{D}_2$ that is consistent with Z, since it must be that $v^{j_1,j_2} = F(j_1,j_2) = u^{j_1,j_2}$. Also if there exists $j \in T$ such that $(v^{j1}, \ldots, v^{jn})$ are not consistent with a degree-d_y polynomial or $(v^{1j}, \ldots, v^{nj})$ are not consistent with a degree-d_x polynomial, then there clearly does not exist any degree-(d_x, d_y) bivariate polynomial in the support of $\mathcal{D}_1$ or $\mathcal{D}_2$ that is consistent with Z.

Now, let us count how many polynomials that are consistent with Z exist in the support of $\mathcal{D}_1$. We have that Z contains points $(v^{1j}, \ldots, v^{nj})$ for $j \in T$ that are consistent with $|T|$ degree-d_x polynomials $F(x,j)$. By Lemma 1, if we have $d_y + 1 - |T|$ more sets of points, $(v^{1j^*}, \ldots, v^{nj^*})$ that are consistent with degree-d_x polynomials, then we can define a unique degree-(d_x, d_y) bivariate polynomial $F(x,y)$. In fact, since for $j^* = 0$, we need to satisfy the constraint that $F(-l+1, 0) = s_1^l$ for $l \in [\ell]$, in order to be in the support of $\mathcal{D}_1$, we will indeed use $j^* = 0$. So, we have to pick $(v^{10}, \ldots, v^{n0})$ consistent with a degree-d_x polynomial $F(x,0)$, such that for $j \in T$, $v^{j0} = F(j,0)$ for the polynomial $F(j,y)$ defined by $\{(v^{j,k}, \cdot)\}_{k \in [n]} \subseteq Z$, and $F(-l+1, 0) = s_1^l$ for $l \in [\ell]$. Since $d_x + 1$ points define such polynomials, and we already have $|T| + \ell$ points, we can pick any $d_x + 1 - |T| - \ell$ remaining points, and still be consistent with Z. Therefore, there exists $|\mathbb{F}|^{d_x+1-|T|-\ell}$ ways to choose $F(x,y)$ from $\mathcal{D}_1$ that will be consistent with Z. However, note that we can do the same exact calculation for finding the number of $F(x,y)$ from $\mathcal{D}_2$ that will be consistent with Z, which is exactly what we wanted to show.

Now, let $Z = \{(v^{jk}, v^{kj})\}_{j \in T, k \in [n]}$ be any set of pairs of elements from $\mathbb{F}$. We have already shown that the number of bivariate polynomials in the support of $\mathcal{D}_1$ that are consistent with any such Z is the same as the number of bivariate polynomials in the support of $\mathcal{D}_2$ that are consistent with Z. This also means that the number of *total* bivariate polynomials in the support of $\mathcal{D}_1$ is the same as that of $\mathcal{D}_2$. Since the polynomials from $\mathcal{D}_1$ and $\mathcal{D}_2$ are chosen randomly, it follows that that the probability that Z is obtained is exactly the same in both cases, as required. □

3 Linear Batched Information-Checking Signatures

In this section, we introduce a crucial building block that will be used in our packed detectable secret sharing scheme: linear batched information-checking signatures (IC signatures). This primitive and its construction are based on that of [PR10], which in turn are based on that of [Cra+99; RB89]. A batched IC signature protocol is executed amongst n parties and allows a dealer D to send a "signature" σ of a batch To ensure that a corrupt INT (or a corrupt D) does not cheat, the n parties, who we call verifiers, each get a "share" of the signature. Importantly, the corrupted parties' shares should together not reveal anything

about $\boldsymbol{s}$. Later, INT can reveal the signature σ of $\boldsymbol{s}$ to the n verifiers. Using the shares previously received, the verifiers then decide whether or not to accept the signature. In fact, we allow D to sign many such batches,based on which INT can add their corresponding signatures together, to get a signature of their sum. We also allow INT to compute the signature of a signed batch of secrets component-wise multiplied with some public vector $\boldsymbol{u} \in \mathbb{F}^\ell$, using the signature of the original batch.

3.1 IC Signature Ideal Functionality

We now formally introduce our ideal functionality for linear batched IC signatures. The properties that we want from an IC signature are intuitively as follows: (i) If the dealer D is honest, then with all-but-negligible probability, the honest verifiers will only accept a signature σ on the batch $\boldsymbol{s}$ input by D; (ii) If the intermediary INT is honest, then after the signing phase phase, INT knows a signature σ on some batch $\boldsymbol{s}$ that the honest verifiers will later accept with all-but-negligible probability; and (iii) If both the dealer D and intermediary INT are honest, then nothing about $\boldsymbol{s}$ is revealed to the corrupt verifiers before the reveal phase. Note that if both the dealer D and intermediary INT are corrupted, we guarantee nothing about the signatures, and in fact, INT can decide to reveal a signature σ for any $\boldsymbol{s}$ of the adversary's choosing.

Furthermore, we require the following linearity properties from our IC signatures: (i) Given a signature σ_1 on $\boldsymbol{s}_1$ and a signature σ_2 on $\boldsymbol{s}_2$, we can define a signature σ_3 on $\boldsymbol{s}_1 + \boldsymbol{s}_2$ with the above properties; and (ii) Given a signature σ on $\boldsymbol{s}$ and a public vector $\boldsymbol{u} \in \mathbb{F}^\ell$, we can define a signature σ' on $\boldsymbol{s} * \boldsymbol{u}$ with the above properties, only if σ is a linear combination of other signatures that were not themselves multiplied by a public vector.

This latter property of multiplication with a public vector is our main contribution to IC signatures. We now present the ideal functionality $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, which captures the above properties (Note: as far as we are aware, there has been no formal treatment in UC, or any other simulation-based framework, of IC signatures individually in prior works).

Functionality 1: $\mathcal{F}_{\mathsf{batch\text{-}IC}}$

This functionality is parameterized by $\ell \in \mathbb{N}$ and n parties, two of which are the dealer D and intermediary INT. It allows the dealer D to create signatures on several vectors $\boldsymbol{s} \in \mathbb{F}^\ell$, add them together, and multiply them with public vectors $\boldsymbol{u} \in \mathbb{F}^\ell$.

1. In the initialization phase, if either the dealer D or intermediary INT are corrupted, then $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ receives Corr from the adversary. If $\mathsf{Corr} = 1$ then $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ outputs this to all parties and for all future inputs $(\mathsf{sign}, \boldsymbol{s}, \mathsf{sid})$ from D, simply outputs $\boldsymbol{s}$ to all parties, and for all other inputs, ignores them.
2. On input $(\mathsf{sign}, \boldsymbol{s}, \mathsf{sid})$ from the dealer D, where $\boldsymbol{s} \in \mathbb{F}^\ell$, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ first stores $\mathsf{isMult} \leftarrow 0$ and $\boldsymbol{s}$ with sid, then outputs $(\boldsymbol{s}, \mathsf{sid})$ to INT, and outputs $(\mathsf{signed}, \mathsf{sid})$ to all other parties. Then, if the dealer D is corrupted, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$

receives $\boldsymbol{s}'$ from the adversary, and stores it with sid. Finally, if $\boldsymbol{s}' \neq \boldsymbol{s}$, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ outputs $(\mathsf{verified}, \boldsymbol{s}', \mathsf{sid})$ to all other parties; otherwise, it outputs $(\mathsf{verified}, \mathsf{sid})$ to all other parties.

3. On input $(\mathsf{reveal}, \mathsf{sid})$ from INT, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ first sends $\boldsymbol{s}$ stored at sid to the adversary. Then:
   (a) If INT is corrupted and D is honest, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ asks the adversary whether to reject. If so, it outputs $(\mathsf{reject}, \mathsf{sid})$ to all parties. Otherwise, it outputs $(\boldsymbol{s}, \mathsf{sid})$ to all parties.
   (b) If INT and D are corrupted, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ asks the adversary whether to reject. If so, it outputs $(\mathsf{reject}, \mathsf{sid})$ to all parties. Otherwise, it receives $\boldsymbol{s}'$ and outputs $(\boldsymbol{s}', \mathsf{sid})$ to all parties.
   (c) Otherwise, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ outputs $(\boldsymbol{s}, \mathsf{sid})$ to all parties.
4. On input $(\mathsf{add}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$ from all parties, let $\mathsf{isMult}_3 \leftarrow 1$, if $\mathsf{isMult}_1 = 1$ or $\mathsf{isMult}_2 = 1$; and $\mathsf{isMult}_3 \leftarrow 0$, otherwise; where $\boldsymbol{s}_1$ and isMult_1 are stored with sid_1 and $\boldsymbol{s}_2$ and isMult_2 are stored with sid_2. $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ stores $\boldsymbol{s}_1 + \boldsymbol{s}_2$ and isMult_3 with sid_3.
5. On input $(\mathsf{mult}, \boldsymbol{u}, \mathsf{sid}, \mathsf{sid}')$ from all parties, where $\boldsymbol{s}$ and $\mathsf{isMult} = 0$ are stored with sid, and $\boldsymbol{u} \in \mathbb{F}^\ell$, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ stores $\boldsymbol{s} * \boldsymbol{u}$ and $\mathsf{isMult} \leftarrow 1$ with sid'.
6. On input (corr, D) or (corr, INT) from the adversary, $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ sends to the adversary all $(\mathsf{sid}, \boldsymbol{s})$ pairs that have not been revealed yet.

3.2 IC Signature Protocol

Now, we present our protocol $\Pi_{\mathsf{batch\text{-}IC}}$ which instantiates $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. In the initialization phase, the dealer D first sends to each party P_i, random $\alpha_i \leftarrow_\$ \mathbb{F}$, for $i \in [n]$.

Signing. When signing some $\boldsymbol{s} \in \mathbb{F}^\ell$, the dealer D samples a random degree-$(t+\ell-1)$ polynomial $f(x)$ such that $f(-j+1) = s^j$ for $j \in [\ell]$ and a random degree-$(t+\ell-1)$ polynomial $r(x)$. D then sends $f(x)$ and $r(x)$ to INT, and to each other party P_i, $v_i \leftarrow f(\alpha_i)$ and $r_i \leftarrow r(\alpha_i)$. Notice that since $f(x)$ is of degree-$(t+\ell-1)$, an adversary's t points $\{v_j\}_{j \in \mathsf{Corr}}$ reveal nothing about $\boldsymbol{s}$. Now, note that a corrupt D could send $v_i \neq f(\alpha_i)$ to some P_i. In order to catch this bad behavior, INT samples random $\beta \leftarrow_\$ \mathbb{F}$ and broadcasts $(\beta, b(x))$, where $b(x) \leftarrow \beta \cdot f(x) + r(x)$; since β is uniformly random, with all-but-negligible probability, P_i will see that $b(\alpha_i) \neq \beta \cdot v_i + r_i$, and thus D is corrupted. Also, observe that $r(x)$ masks $f(x)$ and thus $\boldsymbol{s}$. However, it could also be the case that D is honest and a corrupted INT broadcasts $(\beta, b(x))$ where $b(x) \neq \beta \cdot f(x) + r(x)$; in this case, the honest D will know that INT is corrupted, and thus the adversary knows $\boldsymbol{s}$, so it can simply broadcast $\boldsymbol{s}$. If D (honest or corrupt) broadcasts $\boldsymbol{s}$, then INT sets the signature $\sigma \leftarrow g(x)$, where $g(x)$ is the degree-ℓ polynomial such that $g(-j+1) = s^j$ for $j \in [\ell]$, and each P_i resets $v_i \leftarrow g(\alpha_i)$. If D does not broadcast $\boldsymbol{s}$, but $b(\alpha_i) \neq \beta \cdot v_i + r_i$ then P_i knows that D is corrupt, and so will accept any signature from INT for this batch of secrets. Also (whether or not D is honest or corrupt), if D does not broadcast $\boldsymbol{s}$, INT sets $\sigma \leftarrow f(x)$.

Adding and multiplication by public vectors. To add two signatures together, INT simply sets $\sigma_3(x) \leftarrow \sigma_1(x) + \sigma_2(x)$, and each P_i sets $v_{3,i} \leftarrow v_{1,i} + v_{2,i}$, where it stored $v_{1,i}$ and $v_{2,i}$ for σ_1 and σ_2, respectively. To multiply σ by some public vector $\boldsymbol{u} \in \mathbb{F}^\ell$, let $u(x)$ be the degree-$(\ell-1)$ polynomial such that $u(-j+1) = u^j$ for $j \in [\ell]$. INT simply sets $\sigma'(x) \leftarrow \sigma(x) \cdot u(x)$ (so that it is of degree $t + 2\ell - 2$) and each P_i sets $v'_i \leftarrow v_i \cdot u(\alpha_i)$.

Revealing. Finally, to reveal a signature, INT simply broadcasts $\sigma(x)$. Then each P_i broadcasts accept if σ is of degree at most $t + 2\ell - 2$ and $\sigma(\alpha_i) = v_i$ or they already marked D as corrupt for this signature (or any of which this signature consists). If at least $t + 1$ parties broadcast accept, then the honest parties set $s^j \leftarrow \sigma(= j + 1)$ for $j \in [\ell]$ and output $(s^1, \ldots, s^\ell)$; otherwise they output reject. Note that if D is honest and INT is corrupted, for any given honest P_i, if σ is incorrect, then the probability that $\sigma(\alpha_i) = v_i$ is negligible, by the Schwartz-Zippel Lemma, since α_i is random and unknown to the adversary. Thus, with all-but-negligible probability, there will be no P_i such that $\sigma(\alpha_i) = v_i$ for incorrect σ, and thus, the honest parties will only accept a correct σ (since there are at most $t < t + 1$ corrupted parties). Observe also that in the case of an honest INT and corrupted D, from above, we will already have that $\sigma(\alpha_i) = v_i$, or P_i marked D as corrupt for this signature, for all $\geq t + 1$ honest P_i, and thus all honest P_i will accept.

Rerandomizaing signatures of degree-$(2t + 2\ell - 2)$ before revealing. One subtlety in the security proof occurs if both D and INT are honest, and a multiplication with some $\boldsymbol{u}$ occurs, boosting the degree of σ to $2t + 2\ell - 2$. In this case, when σ is revealed in the ideal world, the simulator $\mathcal{S}$ only has at most t corrupted parties' shares and the ℓ points corresponding to the underlying signed $\boldsymbol{s}$, and thus cannot correctly simulate the polynomial $\sigma(x)$. For this reason, before broadcasting σ, INT re-randomizes it with a degree-$(2t + 2\ell - 2)$ polynomial $o(x)$ given to INT by D in the initialization phase, such that $o(-j + 1) = 0$ for $j \in [\ell]$. Each party P_i also adds to v_i, $o_i \leftarrow o(\alpha_i)$, given to them by D in the initialization phase. Similarly to before, for honest INT to ensure that corrupted D gave it $o(x)$ corresponding to the honest parties' o_i values, in the initialization phase it actually receives another such polynomial $o'(x)$ from D, and broadcasts $(\beta, \beta \cdot o(x) - o'(x))$, which the honest parties verifies is consistent with their o_i, o'_i. If an honest INT or D catches a corrupted D or INT, respectively, misbehaving during the initialization phase, then it broadcasts $\mathsf{Corr} \leftarrow 1$. If so, then for all future signatures, D simply broadcasts the secret batch $\boldsymbol{s}$ (since the adversary would learn it anyway).

Now we present the formal protocol $\Pi_{\mathsf{batch\text{-}IC}}$, below.

Protocol 1: $\Pi_{\mathsf{batch\text{-}IC}}$

1. In the initialization phase:
   (a) First, the dealer D samples random α_i for every other party P_i, and sends it to them.

(b) Then, for $\tau \in [N]$ (in parallel), for some number N:
   i. D samples two random degree-$(t+2\ell-2)$ polynomials $o_1(x)$ and $o_2(x)$ such that $o_1(-j+1) = o_2(-j+1) = 0$ for $j \in [\ell]$.
   ii. D then sends $(o_1(x), o_2(x))$ to INT and to each other party P_i, $o_{1,i} \leftarrow o_1(\alpha_i)$ and $o_{2,i} \leftarrow o_2(\alpha_i)$.
   iii. Next, if $o_1(x), o_2(x)$ are not degree-$(t+2\ell-2)$ polynomials such that $o_1(-j+1) = 0$ and $o_2(-j+1) = 0$ for all $j \in [\ell]$, then INT sets $\mathsf{Corr} \leftarrow 1$ and broadcasts Corr, then all parties output Corr. Otherwise, INT chooses random β and broadcasts $(\beta, o(x))$, where $o(x) \leftarrow \beta \cdot o_1(x) - o_2(x)$.
   iv. D then checks that $o(x) = \beta \cdot o_1(x) - o_2(x)$ and if not, sets $\mathsf{Corr} \leftarrow 1$, and broadcasts Corr, then all parties output Corr.
   v. If D did not broadcast $\mathsf{Corr} = 1$ but $o(\alpha_i) \neq \beta \cdot o_1(\alpha_i) - o_2(\alpha_i)$, then P_i sets $\mathsf{dealerbad}_\tau \leftarrow 1$.
   vi. Then INT stores $o_\tau(x) \leftarrow o_1(x)$ and each party P_i stores $o_{\tau,i} \leftarrow o_{1,i}$

(c) If D or INT broadcasted $\mathsf{Corr} = 1$ at any point above, then for all future inputs $(\mathsf{sign}, \boldsymbol{s}_{\mathsf{sid}}, \mathsf{sid})$, the protocol consists of D simply broadcasting $\boldsymbol{s}_{\mathsf{sid}}$, and for all other inputs, the parties ignore them.

2. On input $(\mathsf{sign}, \boldsymbol{s}_{\mathsf{sid}}, \mathsf{sid})$:
   (a) The dealer D samples a random degree-$(t+\ell-1)$ polynomial $f(x)$ such that $f(-j+1) = s^j_{\mathsf{sid}}$ for all $j \in [\ell]$, and a random degree-$(t+\ell-1)$ polynomial $r(x)$.
   (b) D then sends $f(x)$ and $r(x)$ to INT and to each other party P_i, $v_{\mathsf{sid},i} \leftarrow f(\alpha_i)$ and $r_{\mathsf{sid},i} \leftarrow r(\alpha_i)$.
   (c) Next, INT chooses random β and broadcasts $(\beta, b(x))$, where $b(x) \leftarrow \beta \cdot f(x) + r(x)$.
   (d) D then checks that $b(x) = \beta \cdot f(x) + r(x)$ and if not, broadcasts $\boldsymbol{s}$.
   (e) If D indeed broadcasts $\boldsymbol{s}$, let $g(x)$ be the degree-ℓ polynomial such that $g(-j+1) = s_j$ for $j \in [\ell]$. In this case, INT sets $\sigma_{\mathsf{sid}} \leftarrow g(x)$ and each verifier P_i resets their $v_{\mathsf{sid},i} \leftarrow g(\alpha_i)$; otherwise, INT sets $\sigma_{\mathsf{sid}} \leftarrow f(x)$ and each P_i locally sets $\mathsf{dealerbad}_{\mathsf{sid}} \leftarrow 1$ if $\beta \cdot v_{\mathsf{sid},i} + r_{\mathsf{sid},i} \neq b(\alpha_i)$. In both cases, the parties set $\mathsf{isMult}_{\mathsf{sid}} \leftarrow 0$.
3. On input $(\mathsf{reveal}, \mathsf{sid})$:
   (a) (For next available $\tau \in [N]$) INT sets $h(x) \leftarrow \sigma_{\mathsf{sid}}(x) + o_\tau(x)$, and each P_i sets $v_{\mathsf{sid},i} \leftarrow v_{\mathsf{sid},i} + o_{\tau,i}$ and $\mathsf{dealerbad}_{\mathsf{sid}} \leftarrow 1$ if $\mathsf{dealerbad}_{\mathsf{sid}}$ was already 1 or $\mathsf{dealerbad}_\tau = 1$.
   (b) INT then broadcasts $h(x)$.
   (c) Then, each P_i broadcasts accept if $h(x)$ is degree at most $t+2\ell-2$ and $h(\alpha_i) = v_{\mathsf{sid},i}$, OR $\mathsf{dealerbad}_{\mathsf{sid}} = 1$; otherwise, they broadcast reject.
   (d) If at least $t+1$ parties broadcast accept, then P_i sets $s^j \leftarrow h(-j+1)$ for $j \in [\ell]$ and then outputs $(s^1, \ldots, s^\ell)$; otherwise they output reject.
4. On input $(\mathsf{add}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$, INT computes $\sigma_{\mathsf{sid}_3} \leftarrow \sigma_{\mathsf{sid}_1} + \sigma_{\mathsf{sid}_2}$. Each other party P_i computes $v_{\mathsf{sid}_3,i} \leftarrow v_{\mathsf{sid}_1,i} + v_{\mathsf{sid}_2,i}$; $\mathsf{dealerbad}_{\mathsf{sid}_3} \leftarrow 1$ if $\mathsf{dealerbad}_{\mathsf{sid}_1} = 1$ or $\mathsf{dealerbad}_{\mathsf{sid}_2} = 1$, otherwise $\mathsf{dealerbad}_{\mathsf{sid}_3} \leftarrow 0$; and $\mathsf{isMult}_{\mathsf{sid}_3} \leftarrow 1$ if $\mathsf{isMult}_{\mathsf{sid}_1} = 1$ or $\mathsf{isMult}_{\mathsf{sid}_2} = 1$, otherwise $\mathsf{isMult}_{\mathsf{sid}_3} \leftarrow 0$.
5. On input $(\mathsf{mult}, \boldsymbol{u}, \mathsf{sid}, \mathsf{sid}')$, the parties first check that $\mathsf{isMult}_{\mathsf{sid}}$ stored with sid satisfies $\mathsf{isMult}_{\mathsf{sid}} = 0$, and abort if not. If so:

(a) Each party interpolates the degree-$(\ell-1)$ polynomial $u(x)$ such that $u(-j+1) = u^j$ for $j \in [\ell]$.
(b) INT then computes $\sigma_{\mathsf{sid}'} \leftarrow \sigma_{\mathsf{sid}} \cdot u(x)$ and each other party P_i computes $v_{\mathsf{sid}',i} \leftarrow v_{\mathsf{sid},i} \cdot u(\alpha_i)$, $\mathsf{dealerbad}_{\mathsf{sid}'} \leftarrow \mathsf{dealerbad}_{\mathsf{sid}}$, and $\mathsf{isMult}_{\mathsf{sid}} \leftarrow 1$.

Efficiency of $\Pi_{\mathsf{batch\text{-}IC}}$. First, it is clear the initialization phase takes $O(1)$ rounds and costs $\mathsf{P2P}(O(n))$ for sending the α_i. We will count the cost of generating the $o_\tau(x)$ in the corresponding reveal phase below.

In the signing phase, D first sends $f(x), r(x)$ to INT, which costs $\mathsf{P2P}(2(t+\ell))$. D then sends the v_i, r_i to the parties P_i, which costs $\mathsf{P2P}(2n)$ values. INT then broadcasts $(\beta, b(x))$ which costs $1 \times \mathsf{BC}(1+t+\ell)$. Then, in the worst case, D broadcasts $\boldsymbol{s}$, which costs another $1 \times \mathsf{BC}(\ell)$. Thus, the signing phase costs $\mathsf{P2P}(O(n+\ell))$ and $1 \times \mathsf{BC}(O(n+\ell))$. If $\ell = \Theta(n)$, then this is $\mathsf{P2P}(O(n))$ and $1 \times \mathsf{BC}(O(n))$. It is clear that the signing phase takes $O(1)$ rounds. Note that both adding and multiplication are local operations.

In the reveal phase, INT broadcasts $h(x)$ which costs at most $1 \times \mathsf{BC}(t+2\ell-2)$. Then, each P_i broadcasts accept or reject which costs $n \times \mathsf{BC}(1)$. Additionally, in the initialization phase, D sends $o_1(x), o_2(x)$ to INT, the former of which INT adds to $\sigma(x)$ to get $h(x)$, which costs $\mathsf{P2P}(2(t+2\ell-2))$, and $o_{1,i}, o_{2,i}$ to each P_i, which costs $\mathsf{P2P}(2n)$. Then INT broadcasts $(\beta, o(x))$, which costs $1 \times \mathsf{BC}(t+2\ell-1)$. Counting the cost to generate $o_1(x)$ as part of the reveal phase cost, we get the reveal phase costs $\mathsf{P2P}(O(n+\ell))$, $O(n) \times \mathsf{BC}(O(1))$, and $O(1) \times \mathsf{BC}(O(n+\ell))$. If $\ell = \Theta(n)$, then this is $\mathsf{P2P}(O(n))$, $O(n) \times \mathsf{BC}(O(1))$, and $O(1) \times \mathsf{BC}(O(n))$. It is clear that the reveal phase takes $O(1)$ rounds.

Theorem 1. *$\Pi_{\mathsf{batch\text{-}IC}}$ UC-realizes $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for any $\ell = \mathtt{poly}(\kappa)$ with probability $1 - \mathtt{negl}(\kappa)$.*

The proof is in Section B.1 in the Supplementary Material.

4 Packed, Batched, (Mass) Detectable Secret Sharing

In this section, we introduce our packed, batched, (mass) detectable secret sharing (DSS) ideal functionality and protocol. We base the protocol off of that of [Cra+99] and take from [Abr+23] the idea of "packing" many secrets into a single bivariate polynomial, as well as batching many bivariate polynomials to amortize costs. A DSS protocol is executed amongst n parties and allows any given P_i to act as a dealer and secret share a batch of secret vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m \in \mathbb{F}^\ell$. As usual, we want the corrupted parties' shares to reveal nothing about $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m \in \mathbb{F}^\ell$. Later, the parties can choose to publicly reconstruct $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$; the reconstruction must either succeed, or $\Omega(n)$ corrupted parties are publicly identified (the exact number depends on ℓ). In fact, we allow the parties to add their shares of many such sharings together, which results in a sharing of the vector-wise sum of the underlying batches of secret vectors. We also allow the parties to compute a

sharing of a shared batch of secrets, element-wise and component-wise multiplied by some public vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in \mathbb{F}^\ell$, using the original sharing.

4.1 Detectable Secret Sharing Ideal Functionality

We now present our packed, batched, DSS ideal functionality. The properties that we want from a DSS are as follows: (i) If the given dealer P_i is honest, then all honest parties will complete the sharing phase; (ii) If the given dealer P_i is honest, then nothing about $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ are revealed before the reconstruction phase; and (iii) If all honest parties finish a sharing phase, then there exists fixed $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ such that (a) if the given dealer P_i is honest, then each $\boldsymbol{x}_i = \boldsymbol{s}_i$, and (b) if all honest parties start the reconstruction phase, either it succeeds with them outputting $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$, or it fails, but $\Omega(n)$ corrupted parties are identified.

Furthermore, we require the following linearity properties from our DSS: (i) Given a sharing of $\boldsymbol{s}_{1,1}, \ldots, \boldsymbol{s}_{1,m}$ and a sharing of $\boldsymbol{s}_{2,1}, \ldots, \boldsymbol{s}_{2,m}$, the parties can compute a sharing of $\boldsymbol{s}_{1,1} + \boldsymbol{s}_{2,1}, \ldots, \boldsymbol{s}_{1,m} + \boldsymbol{s}_{2,m}$ with the above properties; and (ii) Given a sharing of $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ and public vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m \in \mathbb{F}^\ell$, the parties can compute a sharing of $\boldsymbol{s}_1 * \boldsymbol{u}_1, \ldots, \boldsymbol{s}_m * \boldsymbol{u}_m$ with the above properties, only if $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ is a linear combination of other sharings that were not themselves multiplied by a batch of public vectors.

Our main contributions to DSS are using packing and batching in the statistical setting, $t < n/2$ setting to amortize costs, as well as the mass detectability property in case of failure of reconstruction. Now we present the ideal functionality $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, which captures the above properties.

Functionality 2: $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$

This functionality is parameterized by $\ell, m \in \mathbb{N}$ and n parties. It allows parties to create several size-m batches of packed Verifiable Secret Sharings of size-ℓ vectors, add them together, and multiply them with size-m batches of size-ℓ public vectors.

1. On input $(\mathsf{share}, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m), \mathsf{sid})$ from party P_i, where each $\boldsymbol{s}_j \in \mathbb{F}^\ell$, and $(\mathsf{share}, P_i, \mathsf{sid})$ from all honest parties, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ first sets $\mathsf{isMult} \leftarrow 0$. Then if P_i is corrupted, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ first asks the adversary whether to continue. If so, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ stores $(P_i, \mathsf{isMult}, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m))$ with sid; else, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ outputs abort to all parties. If P_i is honest, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ stores $(P_i, \mathsf{isMult}, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m))$ with sid.
2. On input $(\mathsf{reconstruct}, \mathsf{sid})$ from all parties, where $(\cdot, \mathsf{isMult}, (\boldsymbol{s}_1 \ldots, \boldsymbol{s}_m))$ is stored at sid, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ first sends $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$ to the adversary and asks whether to continue. If so, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ outputs $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$ to all parties. Otherwise, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ receives from the adversary a set of indices $T \subseteq [n]$ corresponding to corrupted parties such that $|T| > t - \ell + 1$ if $\mathsf{isMult} = 0$ and $|T| > t - 2\ell + 2$ if $\mathsf{isMult} = 1$, and outputs T to all parties.[a]
3. On input $(\mathsf{add}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$ from all parties, where $(\cdot, \mathsf{isMult}_1, (\boldsymbol{s}_{1,1} \ldots, \boldsymbol{s}_{1,m}))$ is stored with sid_1 and $(\cdot, \mathsf{isMult}_2, (\boldsymbol{s}_{2,1} \ldots, \boldsymbol{s}_{2,m}))$ is stored with sid_2, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ stores $(\perp, \mathsf{isMult}_3, (\boldsymbol{s}_{1,1} + \boldsymbol{s}_{2,1}, \ldots, \boldsymbol{s}_{1,m} + \boldsymbol{s}_{2,m}))$

with sid_3, where $\mathsf{isMult}_3 \leftarrow 0$ if $\mathsf{isMult}_1 = \mathsf{isMult}_2 = 0$, and $\mathsf{isMult}_3 \leftarrow 1$ otherwise.

4. On input $(\mathsf{mult}, (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m), \mathsf{sid}, \mathsf{sid}')$ from all parties, where each $\boldsymbol{u}_i \in \mathbb{F}^\ell$ and $(\cdot, 0, (\boldsymbol{s}_1 \ldots, \boldsymbol{s}_m))$ is stored at sid, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ stores $(\perp, 1, (\boldsymbol{s}_1 * \boldsymbol{u}_1, \ldots, \boldsymbol{s}_m * \boldsymbol{u}_m))$ with sid'.
5. On input (corr, P_i) from the adversary, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ sends to the adversary all pairs $(\mathsf{sid}, (P_i, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)))$.

[a] If $\ell = 1$, $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$ will always be output to the parties since there are only t corrupted parties and thus the adversary cannot send T such that $|T| > t$.

4.2 Detectable Secret Sharing Subroutines

Before presenting our DSS protocol $\Pi_{\mathsf{Packed\text{-}DSS}}$, we will first present various procedures which $\Pi_{\mathsf{Packed\text{-}DSS}}$ uses. Note, however, that $\Pi_{\mathsf{Packed\text{-}DSS}}$ starts with each party initializing separate instances of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ as a dealer with each other party acting as intermediary. The procedures will use these instances of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.

Sharing Procedure $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ We begin by presenting the sharing procedure, $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}(d_x, \mathsf{sid}, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m))$, below. When a party P_i wants to secret share a batch of vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m \in F^\ell$,[7] with degree $n-1 \geq d_x \geq t+\ell-1$, it begins by sampling m random degree-(d_x, t) bivariate polynomials such that $F_\eta(-l+1, 0) = s_\eta^l$ for $l \in [\ell], \eta \in [m]$. Then, letting $z_\eta^{jk} \leftarrow F_\eta(j, k)$ and $\boldsymbol{z}_\eta^{jk} \leftarrow (z_1^{jk}, \ldots, z_m^{jk})$, P_i invokes the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as intermediary on input $\boldsymbol{z}_\eta^{jk}$ and $\boldsymbol{z}_\eta^{kj}$ (thus implicitly sending to P_j these vectors), for $j, k \in [n]$. Each P_j then ensures that the points it receives define valid degree-d_x and degree-$(t+\ell-1)$ polynomials, respectively. If not, it reveals their points to all of the parties, using $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Then, if a party sees such a set of bad points from some other party P_k (checking that indeed, the points are bad), it aborts. Then, P_j invokes the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_k as intermediary on input $\boldsymbol{z}_\eta^{kj}$ (thus implicitly sending to P_k this vector), for $k \in [n]$. Next, P_j compares those points it received from P_k to those received from the dealer P_i, and if there is any inconsistency, reveals those points that P_i gave it to all of the parties, using $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Then, P_j checks if some P_k revealed points that are not consistent with those it received from P_i, and if so, reveals those points that P_i gave it to all of the parties, using $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. If any pair of parties $P_j \neq P_k$ revealed two different vectors of points from the dealer P_i, then all parties abort. Otherwise, each party P_j outputs its share, $(\boldsymbol{z}_{\mathsf{sid}}^{j1}, \ldots, \boldsymbol{z}_{\mathsf{sid}}^{jn})$.

We will only ever explicitly use $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ in the following to generate sharings with degree $d_x = t + \ell - 1$ or degree $d_x = t + (2\ell - 1)$. If the parties do not abort in $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, we denote a sharing of $\boldsymbol{s} = (\boldsymbol{s_1}, \ldots, \boldsymbol{s_m})$ with degree $d_x = t + \ell - 1$ as $[\![\boldsymbol{s}]\!]$ and a sharing with degree $d_x = t + \ell - 1$ as $[\![\boldsymbol{s}]\!]_*$.

[7] For a given instance of $\Pi_{\mathsf{Packed\text{-}DSS}}$, we use the same packing parameter ℓ and batching parameter m for each call to $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$.

Procedure 2: $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}(P_i, d_x, \mathsf{sid}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$

This procedure takes in the party P_i acting as dealer, $n-1 \geq d_x \geq t+\ell-1$ and produces a degree-(d_x, t) sharing of $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$.

1. Party P_i samples m random bivariate polynomials $F_1(x,y), \ldots, F_m(x,y)$ of degree at most d_x in x and t in y, such that $F_\eta(-l+1, 0) = s_\eta^l$ for $l \in [\ell], \eta \in [m]$.
2. Let $z_\eta^{jk} \leftarrow F_\eta(j,k)$ and $\boldsymbol{z}_{\mathsf{sid}}^{jk} \leftarrow (z_1^{jk}, \ldots, z_m^{jk})$. P_i invokes the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as intermediary on inputs $(\mathsf{sign}, \boldsymbol{z}_{\mathsf{sid}}^{jk}, \mathsf{sid}\|k\|0)$ and $(\mathsf{sign}, \boldsymbol{z}_{\mathsf{sid}}^{kj}, \mathsf{sid}\|k\|1)$, for $j,k \in [n]$.
3. Each party P_j checks that for each $\eta \in [m]$, $z_\eta^{j1}, \ldots, z_\eta^{jn}$ received from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ define a degree-t polynomial and $z_\eta^{1j}, \ldots, z_\eta^{nj}$ received from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ define a degree-d_x polynomial. If not, P_j reveals $\boldsymbol{z}_{\mathsf{sid}}^{jk}, \boldsymbol{z}_{\mathsf{sid}}^{kj}$ to all of the parties by invoking $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ on $(\mathsf{reveal}, \mathsf{sid}\|k\|0)$ and $(\mathsf{reveal}, \mathsf{sid}\|k\|1)$, for all $k \in [n]$.
4. If a party sees polynomial evaluations from some other party P_k that do not define degree-t or degree-d_x polynomials, respectively, it aborts.
5. Each P_j invokes the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_k as the intermediary on input $(\mathsf{sign}, \boldsymbol{z}_{\mathsf{sid}}^{kj}, \mathsf{sid})$, for $k \in [n]$.
6. P_j compares the values $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ which he received from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for each $k \in [n]$ in the previous round to the values received from P_i. If there is any inconsistency, P_j reveals $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ received from P_i to all of the parties by invoking $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ on $(\mathsf{reveal}, \mathsf{sid}\|k\|0)$.
7. P_j checks if some P_k revealed a value $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ which is different from that which P_i gave it. If so, then P_j reveals $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ to all parties by invoking $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ on $(\mathsf{reveal}, \mathsf{sid}\|k\|1)$.
8. If for any index pair $(j,k) \in [n] \times [n]$, a party sees two different vectors of points from P_i, then it aborts; otherwise, each party P_j outputs its share $(\boldsymbol{z}_{\mathsf{sid}}^{j1}, \ldots, \boldsymbol{z}_{\mathsf{sid}}^{jn})$.

Now, we prove the following simple lemma about $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$:

Lemma 3. *If the dealer P_i is honest in $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, then all honest parties finish $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ without aborting.*

The proof is in Section B.2 in the Supplementary Material.

Next, we prove the following lemma, which shows that if the values $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ which the honest parties P_j input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ in step 5 of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, and which thus become part of P_k's share, define degree-t polynomials $g_k(y)$, then they uniquely define the underlying degree-(d_x, t) bivariate polynomials $F_\eta(x,y)$ and thus the shared secrets $\boldsymbol{s}_\eta$.

Lemma 4. *For any $K \subseteq [n]$ such that $|K| \geq d_x + 1$, let $\boldsymbol{z}_{\mathsf{sid}}^{kj}$, for $k \in K$, be the vectors that the honest parties P_j input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ in step 5 of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$. Assume that $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ does not abort and that for each $\eta \in [m], k \in K$, $\{z_\eta^{kj}\}_{j \in [\mathsf{Hon}]}$ define degree-t polynomials. Then for all $\eta \in [m]$, the $\{z_\eta^{kj}\}_{j \in \mathsf{Hon}}$ for $k \in K$ together define unique degree-(d_x, t) bivariate polynomials $F_\eta(x,y)$.*

Proof. First, consider just the $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ for $k \in K$ that the honest parties P_j input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ and fix any $\eta \in [m]$. We have that the number of honest parties P_j is at

least $t+1$. Observe that if $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ does not abort, then each honest party P_j's shares $\{z_\eta^{kj}\}_{k\in K}$ are consistent with degree-(d_x) polynomials. In addition, by assumption, the shares $\{z_\eta^{kj}\}_{j\in\mathsf{Hon}}$ define degree-t polynomials. So, we have $|K| \geq d_x+1, |\mathsf{Hon}| \geq t+1$ and $\{z_\eta^{jk}\}_{j\in\mathsf{Hon},k\in K}$ satisfying the requirements of Corollary 1. Therefore, there exists a unique degree-(d_x,t) bivariate polynomial $F_\eta(x,y)$ that are defined by these points. □

Efficiency of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$. For analyzing the communication complexity of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, we will utilize the efficiency of our $\Pi_{\mathsf{batch\text{-}IC}}$ protocol for $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. P_i, for each P_j, signs length-m vectors $\boldsymbol{z}_{\mathsf{sid}}^{jk}, \boldsymbol{z}_{\mathsf{sid}}^{kj}$ for $k\in[n]$, with $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ which costs $\mathsf{P2P}(O(n^2\cdot(n+m)))$ and $n^2\times\mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. Then, in the worst case, each P_j could reveal those signatures, which costs $\mathsf{P2P}(O(n^2\cdot(n+m)))$, $n^3\times\mathsf{BC}(O(1))$, and $n^2\times\mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. Next, each P_i sends signs for each P_k length-m vectors $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ with $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, which costs $\mathsf{P2P}(n^2\cdot(n+m))$ and $O(n^2)\times\mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. Next, in the worst case, each P_j could reveal the signatures from P_i for all $k\in[n]$, which costs $\mathsf{P2P}(O(n^2\cdot(n+m)))$, $n^3\times\mathsf{BC}(1)$, and $n^2\times\mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. Altogether, this is $\mathsf{P2P}(O(n^3+n^2m))$, $O(n^3)\times\mathsf{BC}(O(1))$, and $O(n^2)\times\mathsf{BC}(O(n+m))$. If $m=\Theta(n)$, this is $\mathsf{P2P}(O(n^3))$, $O(n^3)\times\mathsf{BC}(O(1))$, and $O(n^2)\times\mathsf{BC}(O(n))$. It is clear that $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ takes $O(1)$ rounds.

Adding Packed, Batched DSS's and Multiplying them by Public Vectors Let us assume that the parties have a packed, batched DSS $[\![\boldsymbol{s}_1]\!]$ and a packed, batched DSS $[\![\boldsymbol{s}_2]\!]$. Adding two such sharings together is a simple, local procedure, $\pi_{\mathsf{Packed\text{-}DSS\text{-}Add}}$, which consists of parties simply adding (the signatures on) their shares together:

Procedure 3: $\pi_{\mathsf{Packed\text{-}DSS\text{-}Add}}([\![\boldsymbol{s}_1]\!], [\![\boldsymbol{s}_2]\!])$

Let $(\boldsymbol{z}_{\mathsf{sid}_1}^{j1},\ldots,\boldsymbol{z}_{\mathsf{sid}_1}^{jn})$ and $(\boldsymbol{z}_{\mathsf{sid}_2}^{j1},\ldots,\boldsymbol{z}_{\mathsf{sid}_2}^{jn})$ be party P_j's respective shares of sharings sid_1 and sid_2.

1. For each $j\in[n]$, P_j sets $\boldsymbol{z}_{\mathsf{sid}_3}^{jk} \leftarrow \boldsymbol{z}_{\mathsf{sid}_1}^{jk} + \boldsymbol{z}_{\mathsf{sid}_2}^{jk}$ and all parties invoke the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as intermediary and P_k as dealer on input $(\mathsf{add},\mathsf{sid}_1,\mathsf{sid}_2,\mathsf{sid}_3)$, for $k\in[n]$.[a]
2. Each P_j outputs new shares $(\boldsymbol{z}_{\mathsf{sid}_3}^{j1},\ldots,\boldsymbol{z}_{\mathsf{sid}_3}^{jn})$.

[a] Note that for our $\Pi_{\mathsf{batch\text{-}IC}}$, the add operation is indeed local.

We denote this as $[\![\boldsymbol{s}_3]\!] \leftarrow [\![\boldsymbol{s}_1]\!] + [\![\boldsymbol{s}_2]\!]$. Note that this also works for sharings $[\![\boldsymbol{s}_1]\!]_*$ and/or $[\![\boldsymbol{s}_2]\!]_*$ of higher degree ($d_x = t+2(\ell-1)$), in which case we denote $[\![\boldsymbol{s}_3]\!]_*$ as the resulting sharing.

Now, let us assume that the parties have a single packed, batched DSS of secrets $[\![\boldsymbol{s}]\!]$ (note that for such a sharing, $d_x = t+\ell-1 \leq n-\ell$), and some public vectors $\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m\in\mathbb{F}^\ell$. Multiplying the sharing by this batch of public vectors is a simple, local procedure, $\pi_{\mathsf{Packed\text{-}DSS\text{-}Mult}}$, which consists of parties simply

multiplying their shares (and the signatures on those shares) by the degree-$(\ell-1)$ polynomials $u_\eta(x)$ such that $u_\eta(-l+1) = u_\eta^l$ for $\eta \in [n], l \in [\ell]$:

Procedure 4: $\pi_{\mathsf{Packed\text{-}DSS\text{-}Mult}}(\llbracket \boldsymbol{s} \rrbracket_{d_x,t}, (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m))$

Let $(\boldsymbol{z}_{\mathsf{sid}}^{j1}, \ldots, \boldsymbol{z}_{\mathsf{sid}}^{jn})$ be party P_j's shares of sharing sid.

1. Each party first interpolates the degree-$(\ell-1)$ polynomials $u_\eta(x)$ such that $u_\eta(-l+1) = u_\eta^l$ for $\eta \in [m], l \in [\ell]$.
2. Then, for each $j \in [n]$, P_j locally computes $\boldsymbol{z}_{\mathsf{sid}'}^{jk} \leftarrow \boldsymbol{z}_{\mathsf{sid}}^{jk} * (u_1(j), \ldots, u_m(j))$ and all parties invoke the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as intermediary and P_k as dealer on input $(\mathsf{mult}, (u_1(j), \ldots, u_m(j)), \mathsf{sid}, \mathsf{sid}')$, for $k \in [n]$.[a]
3. Finally, each P_j outputs new shares $(\boldsymbol{z}_{\mathsf{sid}'}^{j1}, \ldots, \boldsymbol{z}_{\mathsf{sid}'}^{jn})$.

[a] Note that for our $\Pi_{\mathsf{batch\text{-}IC}}$, the mult operation is indeed local.

We denote this as $\llbracket \boldsymbol{s} \rrbracket_* \leftarrow \llbracket \boldsymbol{s} \rrbracket * \boldsymbol{u}$, since the new sharing has degree $d_x = t + 2(\ell-1)$.

Let $F_{\mathsf{sid}',\eta}(x,y)$ be the unique polynomials defined by the $\{z_{\mathsf{sid}',\eta}^{kj}\}_{j \in \mathsf{Hon}}$ for $k \in K$, of some sharings $\llbracket \boldsymbol{s}' \rrbracket$ output by $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, according to Lemma 4. We can again prove the following lemma similar to Lemma 4, which essentially says that for any sharing $\llbracket \boldsymbol{s} \rrbracket$ (or $\llbracket \boldsymbol{s} \rrbracket_*$) formed by running the addition and multiplication procedures above on sharings $\llbracket \boldsymbol{s}' \rrbracket$ originally output by $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, the $\{\boldsymbol{z}_{\mathsf{sid}}^{jk}\}_{j \in \mathsf{Hon}}$ part of each P_k's share (defined by the corresponding signatures), together uniquely define the underlying degree-(d_x, t) bivariate polynomials $F_\eta(x,y)$, which are equal to the polynomials that result from applying the same addition and multiplication procedures on the $F_{\mathsf{sid}',\eta}(x,y)$ above from the original sharings $\llbracket \boldsymbol{s}' \rrbracket$.[8]

Lemma 5. *Let the sharing $\llbracket \boldsymbol{s} \rrbracket$ (resp. $\llbracket \boldsymbol{s} \rrbracket_*$), be the result of addition and multiplication procedures on sharings $\llbracket \boldsymbol{s}' \rrbracket$ originally output by $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$. For any $K \subseteq [n]$ such that $|K| \geq d_x + 1$, let $\{\boldsymbol{z}_{\mathsf{sid}}^{kj}\}_{j \in \mathsf{Hon}}$ be the part of each P_k's shares of $\llbracket \boldsymbol{s} \rrbracket$ (resp. $\llbracket \boldsymbol{s} \rrbracket_*$) defined by the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as dealer and P_k as intermediary. Assume that for each $\eta \in [m], k \in K$, $\{z_{\mathsf{sid},\eta}^{kj}\}_{j \in [\mathsf{Hon}]}$ define degree-t polynomials. Then for all $\eta \in [m]$, the $\{z_{\mathsf{sid},\eta}^{kj}\}_{j \in \mathsf{Hon}}$ for $k \in K$ together define unique degree-(d_x, t) bivariate polynomials $F_{\mathsf{sid},\eta}(x,y)$ which are equal to the polynomials which result from applying the same addition and multiplication procedures on the unique polynomials $F_{\mathsf{sid}',\eta}(x,y)$ defined by the $\{z_{\mathsf{sid}',\eta}^{kj}\}_{j \in \mathsf{Hon}}$ for $k \in K$, by Lemma 4.*

Proof. If for each $\eta \in [m], k \in K$, the parts of shares $\{z_{\mathsf{sid},\eta}^{kj}\}_{j \in [\mathsf{Hon}]}$ of sharing $\llbracket \boldsymbol{s} \rrbracket$ (resp. $\llbracket \boldsymbol{s} \rrbracket_*$) define degree-$t$ polynomials, it must be that for each $\eta \in [m], k \in K$, the parts of shares $\{z_{\mathsf{sid}',\eta}^{kj}\}_{j \in [\mathsf{Hon}]}$ of sharings $\llbracket \boldsymbol{s}' \rrbracket$ define degree-t polynomials in y, since the shares are only added together and multiplied by univariate polynomials

[8] A 'multiplication procedure' multiplying a sharing by $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$ corresponds to multiplying the polynomials defined by the sharing by the degree-$(\ell-1)$ polynomials $u_1(x), \ldots, u_m(x)$ defined by the above vectors.

in x, using operations on $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instances (which are thus performed correctly). Then by Lemma 4, we have that for all $\eta \in [m]$, the $\{z^{kj}_{\mathsf{sid}',\eta}\}_{j\in\mathsf{Hon}}$ for $k \in K$ together define unique degree-$(d_{\mathsf{sid}',x}, t)$ bivariate polynomials $F_{\mathsf{sid}',\eta}(x,y)$. Let us first consider the case where some sharing $[\![\boldsymbol{s}']\!]$ is multiplied by public vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m$. Since the shares are defined by the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instances, it must be that the new shares are $\{z^{kj}_{\mathsf{sid}',\eta} * u_\eta(j)\}_{j\in\mathsf{Hon}}$. Therefore, the new shares together define unique degree-$(d_{\mathsf{sid}',x} + \ell - 1, t)$ bivariate polynomials $F_{\mathsf{sid}',\eta}(x,y) \cdot u_\eta(x)$. Now, let us consider the case where two sharings $[\![\boldsymbol{s}_1]\!], [\![\boldsymbol{s}_2]\!]$ are added together (where possibly one or both of them were multiplied by public vectors). Since the shares are defined by the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instances, it must be that the new shares are $\{z^{kj}_{\mathsf{sid}'_1,\eta} + z^{kj}_{\mathsf{sid}'_2,\eta}\}_{j\in\mathsf{Hon}}$. Therefore, the new shares together define unique degree-$(\max\{d_{\mathsf{sid}'_1,x}, d_{\mathsf{sid}'_1,x}\}, t)$ bivariate polynomials $F_{\mathsf{sid}_1,\eta}(x,y) + F_{\mathsf{sid}_2,\eta}(x,y)$. We can continue the process above inductively, as long as d_x remains at most $n-1$, until we arrive at unique degree-(d_x, t) bivariate polynomials $F_{\mathsf{sid},\eta}(x,y)$. □

Essentially, the above lemma shows that our add and multiplication procedures have the desired outcome of performing the corresponding operations. The following corollary will help us show that the correct secrets can then be reconstructed from such sharings.

Corollary 2. *If for each $\eta \in [n], k \in K$, $\{z^{ki}_{\mathsf{sid},\eta}\}_{i\in[n]}$ in P_k's share of $[\![\boldsymbol{s}]\!]$ (resp. $[\![\boldsymbol{s}]\!]_*$) define a degree-t polynomial, then given any $I \subseteq [n]$ such that $|I| = t+1$ (such as $[t+1]$) $\{z^{ki}_{\mathsf{sid},\eta}\}_{i\in I}$ can be used to interpolate as in Equation 1 the same unique degree-(d_x, t) bivariate polynomials $F_{\mathsf{sid},\eta}(x,y)$ as in Lemma 5.*

Proof. For each $k \in K$, we have by assumption that $\{z^{ki}_{\mathsf{sid},\eta}\}_{i\in[n]}$ define a degree-t polynomial $g_k(y)$. In particular, since $F_{\mathsf{sid},\eta}(k,y)$ is also a degree-t polynomial consistent with at least $t+1$ points $\{z^{kj}_{\mathsf{sid},\eta}\}_{j\in\mathsf{Hon}} \subseteq \{z^{ki}_{\mathsf{sid},\eta}\}_{i\in[n]}$, it must be that $F_{\mathsf{sid},\eta}(k,y) = g_k(y)$ and therefore $F_{\mathsf{sid},\eta}(k,i) = z^{ki}_{\mathsf{sid},\eta}$ for every $i \in [n]$. Moreover, for every $i \in [n]$, we have that $F_{\mathsf{sid},\eta}(x,i)$ is a degree-d_x polynomial consistent with the at least $d_x + 1$ points $\{z^{ki}_{\mathsf{sid},\eta}\}_{k\in K}$. Therefore, we have that any $I \subseteq [n]$ of size $|I| = t+1$ (such as $[t+1]$), the given K of size $|K| \geq d_x + 1$, and $\{z^{ki}_{\mathsf{sid},\eta}\}_{i\in I, k\in K}$ satisfy the requirements of Lemma 1, and so define a unique degree-(d_x, t) bivariate polynomial $F'_{\mathsf{sid},\eta}(x,y)$ (that can be interpolated as in Lemma 1). Since $F'_{\mathsf{sid},\eta}(x,y)$ is uniquely defined by $\{z^{ki}_{\mathsf{sid},\eta}\}_{i\in I,k\in K}$ and from above, $F_{\mathsf{sid},\eta}(k,i) = z^{ki}_{\mathsf{sid},\eta}$ for each $i \in I, k \in K$, it must be that $F'_{\mathsf{sid},\eta}(x,y) = F_{\mathsf{sid},\eta}(x,y)$, for any such I. □

Reconstruction Procedure $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ Next, we present the reconstruction procedure, $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}([\![\boldsymbol{s}]\!])$, which the honest parties use to reconstruct the batch of secret vectors defined by their shares of the sharing $[\![\boldsymbol{s}]\!]$. All parties P_k first reveal their share $\boldsymbol{z}^{k1}_{\mathsf{sid}}, \ldots, \boldsymbol{z}^{kn}_{\mathsf{sid}}$ to all parties using $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Then, each P_k checks if the points that each P_j revealed define degree-t polynomials in y, and if not, marks them as corrupt. Then, if the number of parties marked corrupt is greater than $2t - d_x$, the honest parties output those parties' identities that

are marked corrupt (note that sharings must satisfy $d_x \leq n-1$, so $2t - d_x > 0$). Otherwise, the parties use the shares of those parties that are not marked corrupt to interpolate the unique, correct $F_\eta(x, y)$ (that exist by Corollary 2) and output the corresponding secrets $\boldsymbol{s}_\eta$. Note that this procedure works in exactly the same way for sharings $[\![\boldsymbol{s}]\!]_*$ of higher degree $d_x = t + 2(\ell - 1)$.

Procedure 5: $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}([\![\boldsymbol{s}]\!])$

This procedure takes as input the party's shares of $[\![\boldsymbol{s}]\!]$ of degree $d_x = t + \ell - 1$ or $[\![\boldsymbol{s}]\!]_*$ of degree $d_x = t + 2(\ell - 1)$.

1. Every party P_k reveals $\boldsymbol{z}_{\mathsf{sid}}^{k1}, \ldots, \boldsymbol{z}_{\mathsf{sid}}^{kn}$ to all other parties by invoking for $j \in [n]$, the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance with P_j as intermediary on input $(\mathsf{reveal}, \mathsf{sid})$.
2. P_k first sets $T \leftarrow \emptyset$ then checks whether P_j's shares revealed in the previous step define degree-t polynomials $F_\eta(j, y)$, $\eta \in [m]$. If not, then P_j is added to T and thus marked as corrupt.
3. If the number of parties marked corrupt in T is greater than $2t - d_x$, then output T and abort.
4. For every P_k not marked as corrupt in $K = [n] \setminus T$, P_j uses the values $F_\eta(k, 1), \ldots, F_\eta(k, t+1)$, together, to interpolate the unique degree-(d_x, t) bivariate polyonmial $F_\eta(x, y)$, as in Equation 1.
5. P_j finally outputs $s_\eta^l \leftarrow F_\eta(-l+1, 0)$ for $\eta \in [m], l \in [\ell]$.

We now have the following lemma, which shows that if $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ are output by the honest parties, then they are the correct secrets corresponding to $[\![\boldsymbol{s}]\!]$; otherwise, each party $P_k \in T$ is actually corrupt. The latter is because for sharings output by $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, honest parties' shares are always consistent with degree-t polynomials, for otherwise they would have aborted. Furthermore, addition and multiplication operations do not affect the degree in the y variable, so the shares always stay consistent with degree-t polynomials.

Lemma 6. *Let $\{\boldsymbol{z}_{\mathsf{sid}}^{kj}\}_{k \in K, j \in [n]}$ be the points that are revealed via $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ in $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$. If $|T| \leq 2t - d_x$, then the honest parties output the correct secrets $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ defined by the unique degree-$(t + \ell - 1, t)$ bivariate polynomials $(F_1(x, y), \ldots, F_m(x, y))$ from Lemma 5. If $|T| > 2t - d_x$, then the honest parties output (abort, T) such that for each $P_j \in T$, $P_j \in \mathsf{Corr}$.*

Proof. If $|T| \leq 2t - d_x$, then we have that $|K| \geq n - 2t + d_x$. Since $n - 2t > 0$, we thus have $|K| \geq d_x + 1$. Moreover, we have that the $\{z_{\mathsf{sid},\eta}^{ki}\}_{i \in [n]}$ for $k \in K$ define degree-t polynomials. So, by Corollary 2, the honest parties interpolate the correct polynomials defined by Lemma 5 and thus output the correct $\boldsymbol{s}_\eta$.

If $|T| > 2t - d_x$, let us analyze why honest parties mark some P_j as corrupt. This happens if P_j is not able to reveal via $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, $(z_{\mathsf{sid},\eta}^{j1}, \ldots, z_{\mathsf{sid},\eta}^{jn})$ consistent with degree-t polynomials, for every $\eta \in [m]$. However, for sharings output by $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, honest parties' shares are always consistent with degree-t polynomials (for otherwise, they would have aborted). Since the sharing sid is only formed from valid addition and multiplication operations, neither of which affect the degree in y, the honest parties' shares $(z_{\mathsf{sid},\eta}^{j1}, \ldots, z_{\mathsf{sid},\eta}^{jn})$ must be

consistent with degree-t polynomials. Thus, they will never be marked as corrupt. Therefore, if $|T| > 2t - d_x$, then the honest parties output (abort, T) such that for each $P_j \in T$, $P_j \in \mathsf{Corr}$. □

Efficiency of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$. For analyzing the communication complexity of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$, we will utilize the efficiency of our $\Pi_{\mathsf{batch\text{-}IC}}$ protocol for $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Each P_k simply reveals $\boldsymbol{z}_{\mathsf{sid}}^{kj}$, for $j \in [n]$ with $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, which costs $\mathsf{P2P}(O(n^2 \cdot (n+m)))$, $n^3 \times \mathsf{BC}(1)$, and $n^2 \times \mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. If $m = \Theta(n)$, this is $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n))$. It is clear that $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ takes $O(1)$ rounds.

Creating Random Sharings $[\![\mathbf{0}]\!]_*$. After multiplying sharings by batches of public vectors, the degree of the sharing increases by $\ell - 1$ in x. Thus, in order to securely open such sharings, we need to mask them by random sharings $[\![\mathbf{0}]\!]_*$, of degree $x_x = t + 2(\ell - 1)$, since all sharings created as part of the $\Pi_{\mathsf{Packed\text{-}DSS}}$ protocol will start as degree $d_x = t + \ell - 1$. We use the typical random extraction technique from [DN07] to do this efficiently, which consists of each party creating their own such random sharings, and then using some super-invertible $(n-t) \times n$ matrix $\boldsymbol{M}$ to take linear combinations of these sharings and then output the resulting sharings that are random to the adversary.

However, it may be that the underlying secrets that corrupted parties share are not equal to $\mathbf{0}, \ldots, \mathbf{0}$. For this, we adapt a standard technique, which takes as input two sharings from the same party which supposedly share $\mathbf{0}, \ldots, \mathbf{0}$, take a random linear combination of the two, then open them to check if they are indeed sharings of $\mathbf{0}, \ldots, \mathbf{0}$. We will adapt standard techniques to sample the random coefficients of the linear combination.

Below we present the procedures corresponding to the above, $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$, $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, and $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$.

Sampling a Random β. The first tool we need is allowing the parties to jointly sample some $\beta \leftarrow_{\$} \mathbb{F}$ that is random to all. We use the following simple procedure $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$ to do so, in which parties sample their own random sharings, add them together, reconstruct the resulting sharing, and output the corresponding secret.[9]

Procedure 6: $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$

1. Each party P_j samples random $\boldsymbol{\beta}_{j,1}, \ldots, \boldsymbol{\beta}_{j,m}$ and runs $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}(P_j, t + \ell - 1, (\boldsymbol{\beta}_{j,1}, \ldots, \boldsymbol{\beta}_{j,m}))$ so that the parties obtain $[\![\boldsymbol{\beta}_j]\!]$.
2. Then, the parties compute $[\![\boldsymbol{\beta}]\!] \leftarrow \sum_{j=1}^{n} [\![\boldsymbol{\beta}_j]\!]$.
3. Finally, the parties reconstruct $[\![\boldsymbol{\beta}]\!]$ using $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ and output the resulting $(\sum_{j=1}^{n} \boldsymbol{\beta}_{j,1}, \ldots, \sum_{j=1}^{n} \boldsymbol{\beta}_{j,m})$ or corrupted parties set T such that $|T| > 2t - (t + \ell - 1) = t - \ell + 1$.

[9] Note that there are more efficient ways to sample such random values, but since it will not affect efficiency of our eventual MPC protocol, we present it this way for simplicity.

We now prove that $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$ works.

Lemma 7. *If $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$ does not abort with some set T of corrupted parties, then it outputs $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m$ that are random and unknown to the adversary.*

The proof appears in Section B.2 in the Supplementary Material.

Efficiency of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$. Each party P_j runs $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ once which costs $\mathsf{P2P}(O(n \cdot n^2 \cdot (n+m)))$, $n \cdot n^3 \times \mathsf{BC}(1)$, and $n \cdot n^2 \times \mathsf{BC}(O(n+m))$, using $\Pi_{\mathsf{batch\text{-}IC}}$. Then all parties run $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ together, which costs $\mathsf{P2P}(O(n^2 \cdot (n+m)))$, $n^3 \times \mathsf{BC}(1)$, and $n^2 \times \mathsf{BC}(O(n+m))$. Altogether, this is $\mathsf{P2P}(O(n^4 + n^2 m))$, $O(n^4) \times \mathsf{BC}(O(1))$, and $O(n^3) \times \mathsf{BC}(O(n+m))$ for $m \cdot \ell$ random β's. If $m = \Theta(n)$, this is $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(O(1))$, and $O(n^3) \times \mathsf{BC}(O(n))$. If additionally $\ell = \Theta(n)$, this is $\mathsf{P2P}(O(n^2))$, $O(n^2) \times \mathsf{BC}(O(1))$, and $O(n) \times \mathsf{BC}(O(n))$ per output β. Clearly, it take $O(1)$ rounds.

Checking Parties' Own $[\![\mathbf{0}]\!]_*$ Sharings. Now we present how the honest parties catch corrupted parties who do not share $\mathbf{0}$. For this, we use the below procedure $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, which takes as input two sharings from the same party which supposedly share $\mathbf{0}, \ldots, \mathbf{0}$, take a random linear combination of the two, then open them to check if they are indeed sharings of $\mathbf{0}, \ldots, \mathbf{0}$.

Procedure 7: $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}([\![\mathbf{0}_1]\!]_*, [\![\mathbf{0}_2]\!]_*)$

This procedure takes as input two batches of $\mathbf{0}$ sharings from a single dealer P_i, checks that they indeed are sharings of $\mathbf{0}$, and if so, outputs the first batch of sharings.

1. First, the parties run $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$ to sample random value β.[a]
2. Next, the parties compute $[\![\mathbf{0}]\!]_* \leftarrow \beta \cdot [\![\mathbf{0}_1]\!]_* - [\![\mathbf{0}_2]\!]_*$ for $k \in [n]$.
3. Then, the parties reconstruct $[\![\mathbf{0}]\!]_*$ using $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ and receive back either (abort, T) or $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m)$.
4. If $(\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m) = (\mathbf{0}, \ldots, \mathbf{0})$, each party P_j outputs $[\![\mathbf{0}_1]\!]_*$; otherwise, they output abort.

[a] This procedure will eventually (only once) be used $\Omega(|C|)$ times in parallel in $\Pi_{\mathsf{Packed\text{-}DSS}}$. We use the same β value each time.

Lemma 8. *If P_i inputs sharings $([\![\boldsymbol{s}_1]\!]_*, [\![\boldsymbol{s}_2]\!]_*) \neq ([\![\mathbf{0}_1]\!]_*, [\![\mathbf{0}_2]\!]_*)$ to $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, then with probability at least $1 - \mathtt{negl}(\kappa)$, all honest parties output either (abort, T), for $|T| > t - 2\ell + 2$ where each $j \in T$ corresponds to a corrupt P_j, or abort. Otherwise, all honest parties output either (abort, T), for $|T| > t - 2\ell + 2$ where each $j \in T$ corresponds to a corrupt P_j, or $[\![\mathbf{0}_1]\!]_*$.*

The proof appears in Section B.2 in the Supplementary Material.

Lemma 9. *$\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ run on an honest party P_i's sharings does not give the adversary any more information on $[\![\mathbf{0}_1]\!]_*$.*

The proof appears in Section B.2 in the Supplementary Material.

Efficiency of $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$. The parties run $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$, which costs $\mathsf{P2P}(O(n^2 \cdot (n+m)))$, $n^3 \times \mathsf{BC}(1)$, and $n^2 \times \mathsf{BC}(O(n+m))$. If $m = \Theta(n)$, this is $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n))$. The parties use a single random β for all parallel runs of the procedure, which costs $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(O(1))$, and $O(n^3) \times \mathsf{BC}(O(n))$ (if $m, \ell = \Theta(n)$). This procedure will be used (only once) $\Omega(|C|)$ times in parallel in $\Pi_{\mathsf{Packed\text{-}DSS}}$, and thus the cost amortizes out if $|C| = \Omega(n)$. Clearly, it takes $O(1)$ rounds.

$\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$. Now, we present the actual procedure to generate random sharings $[\![\mathbf{0}]\!]_*$ using super-invertible matrix $\boldsymbol{M}$.

Procedure 8: $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$

This procedure has each party share batches of $\mathbf{0}$ and outputs batches of $n-t$ random sharings of $\mathbf{0}$ each. Initialize $T = \emptyset$.

1. Every party P_i uses $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ to share $[\![\mathbf{0}_{i,1}]\!]_*$ and $[\![\mathbf{0}_{i,2}]\!]_*$.
2. The parties then run $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ on $[\![\mathbf{0}_{i,1}]\!]_*$ and $[\![\mathbf{0}_{i,2}]\!]_*$ from each party P_i above and output the first batch for each P_i; or if P_i does not share $\mathbf{0}_{i,1} = \mathbf{0} = \mathbf{0}_{i,2}$, the parties add P_i to the set T.
3. For $P_i \in T$, store the canonical sharing $[\![\mathbf{0}_{i,1}]\!]_* \leftarrow [\![\mathbf{0}]\!]_*$ for this party.
4. Every party P_j then computes and outputs

$$([\![\mathbf{0}_1]\!]_*, \ldots, [\![\mathbf{0}_{n-t}]\!]_*)^\intercal \leftarrow \boldsymbol{M} \cdot ([\![\mathbf{0}_{1,1}]\!]_*, \ldots, [\![\mathbf{0}_{n,1}]\!]_*)^\intercal.$$

Lemma 10. *For each $i \in [n-t]$, the output $[\![\mathbf{0}_1]\!]_*, \ldots, [\![\mathbf{0}_{n-t}]\!]_*$ are distributed randomly given the corrupted parties' shares.*

The proof appears in Section B.2 in the Supplementary Material.

Efficiency of $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$. Each party P_i runs $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ twice, which costs $\mathsf{P2P}(O(n \cdot n^3 + n^2 m))$, $O(n \cdot n^3) \times \mathsf{BC}(O(1))$, and $O(n \cdot n^2) \times \mathsf{BC}(O(n+m))$. Then the parties run $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ for P_i's sharings, which costs $\mathsf{P2P}(O(n \cdot n^2 \cdot (n+m)))$, $n \cdot n^3 \times \mathsf{BC}(1)$, and $n \cdot n^2 \times \mathsf{BC}(O(n+m))$. Altogether, this is $\mathsf{P2P}(O(n^4 + n^3 m))$, $O(n^4) \times \mathsf{BC}(O(1))$, and $O(n^3) \times \mathsf{BC}(O(n+m))$ for $\Omega(n)$ such random zero sharings, or $\mathsf{P2P}(O(n^3 + n^2 m))$, $O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n+m))$ per sharing. If $m = \Theta(n)$, then this is $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n))$ per sharing. Clearly, it takes $O(1)$ rounds.

4.3 Detectable Secret Sharing Protocol

Now, we are finally ready to present our $\Pi_{\mathsf{Packed\text{-}DSS}}$ protocol. For generating a sharing, a given dealer simply uses $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ with degree $d_x \leftarrow t + \ell - 1$. For adding sharings and multiplying them by public vectors, the parties use $\pi_{\mathsf{Packed\text{-}DSS\text{-}Add}}$ and $\pi_{\mathsf{Packed\text{-}DSS\text{-}Mult}}$, respectively. The parties also keep track of when a given sharing is multiplied by a public vector. Then, for reconstruction, if the sharing is a linear combination of sharings that have not been multiplied by

a public vector, the parties simply us $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ to reconstruct it. Otherwise, the parties first re-randomize it by adding a random sharing $[\![\mathbf{0}]\!]_*$ to it, and then use $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ to reconstruct it.

Protocol 9: $\Pi_{\mathsf{Packed\text{-}DSS}}$

1. In the initialization phase, each party initializes instances of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ with each other party as intermediary. The parties also run $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$ to compute a number N of random packed zero sharings $[\![\mathbf{0}_\tau]\!]_*$ for $\tau \in N$ (in parallel).
2. On input $(\mathsf{share}, (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m), \mathsf{sid})$, P_i runs $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}(P_i, t + \ell - 1, \mathsf{sid}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$ to share $[\![\boldsymbol{s}_{\mathsf{sid}}]\!]$, then every party P_j sets $\mathsf{isMult} \leftarrow 0$ and stores $(\mathsf{sid}, (\mathsf{isMult}, [\![\boldsymbol{s}_{\mathsf{sid}}]\!]))$.
3. On input $(\mathsf{reconstruct}, \mathsf{sid})$, the parties first check isMult stored with sid. If $\mathsf{isMult} = 0$, then the parties run $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ on $[\![\boldsymbol{s}_{\mathsf{sid}}]\!]$ and output $\boldsymbol{s}_{\mathsf{sid}}$. Otherwise, the parties (for next available $\tau \in [N]$), compute $[\![\boldsymbol{s}_{\mathsf{sid}}]\!]_* + [\![\mathbf{0}_\tau]\!]_*$ and then run $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$ on it and output $\boldsymbol{s}_{\mathsf{sid}}$.
4. On input $(\mathsf{add}, \mathsf{sid}_1, \mathsf{sid}_2, \mathsf{sid}_3)$, the parties compute $[\![\boldsymbol{s}_{\mathsf{sid}_3}]\!] \leftarrow [\![\boldsymbol{s}_{\mathsf{sid}_1}]\!] + [\![\boldsymbol{s}_{\mathsf{sid}_2}]\!]$ (using $\pi_{\mathsf{Packed\text{-}DSS\text{-}Add}}$). Then if $\mathsf{isMult}_1 = \mathsf{isMult}_2 = 0$, they set $\mathsf{isMult}_3 \leftarrow 0$; otherwise, they set $\mathsf{isMult}_3 \leftarrow 1$ Finally, they store $(\mathsf{sid}_3, (\mathsf{isMult}, [\![\boldsymbol{s}_{\mathsf{sid}_3}]\!]))$.[a]
5. On input $(\mathsf{mult}, (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m), \mathsf{sid}, \mathsf{sid}')$: The parties first check that isMult stored with sid satisfies $\mathsf{isMult} = 0$, and abort if not. If so, they compute $[\![\boldsymbol{s}_{\mathsf{sid}'}]\!]_* \leftarrow [\![\boldsymbol{s}_{\mathsf{sid}}]\!] * \boldsymbol{u}$ (using $\pi_{\mathsf{Packed\text{-}DSS\text{-}Mult}}$). Finally, the parties store $(\mathsf{sid}', (1, [\![\boldsymbol{s}_{\mathsf{sid}'}]\!]_*))$.

[a] This also works if the sharings corresponding to sid_1 and/or sid_2 have higher degree $d_x = t + 2(\ell - 1)$; i.e., for sharings $[\![\boldsymbol{s}_{\mathsf{sid}_1}]\!]_*$ and/or $[\![\boldsymbol{s}_{\mathsf{sid}_2}]\!]_*$. In this case we store $[\![\boldsymbol{s}_{\mathsf{sid}_3}]\!]_*$ of degree $d_x = t + 2(\ell - 1)$.

Efficiency of $\Pi_{\mathsf{Packed\text{-}DSS}}$. Initialization uses $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$ in parallel, which takes $O(1)$ rounds. We will count the communication cost of generating each such zero sharing towards each such sharing that is reconstructed using it below.

Sharing uses $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, which costs $\mathsf{P2P}(O(n^3+n^2m)), O(n^3)\times\mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n+m))$. If $m, \ell = \Theta(n)$, then this is $\mathsf{P2P}(O(n))$, $O(n) \times \mathsf{BC}(O(1))$, and $O(1) \times \mathsf{BC}(O(n))$ per underlying secret. It also takes $O(1)$ rounds.

Reconstruction possibly uses a zero sharing, generated from $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$, which costs $\mathsf{P2P}(O(n^3+n^2m)), O(n^3)\times\mathsf{BC}(O(1))$, and $O(n^2)\times\mathsf{BC}(O(n+m))$ for this sharing. It then uses $\pi_{\mathsf{Packed\text{-}DSS\text{-}Rec}}$, which costs $\mathsf{P2P}(O(n^3+n^2m)), O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n+m))$. Altogether, this is $\mathsf{P2P}(O(n^3+n^2m))$, $O(n^3) \times \mathsf{BC}(O(1))$, and $O(n^2) \times \mathsf{BC}(O(n+m))$. If $m, \ell = \Theta(n)$, then this is $\mathsf{P2P}(O(n))$, $O(n) \times \mathsf{BC}(O(1))$, and $O(1) \times \mathsf{BC}(O(n))$ per underlying secret. It also takes $O(1)$ rounds.

Theorem 2. *$\Pi_{\mathsf{Packed\text{-}DSS}}$ UC-realizes $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ in the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$-hybrid model for any $\ell \le t/2$ and any $m = \mathtt{poly}(\kappa)$, with probability $1 - \mathtt{negl}(\kappa)$.*

The proof appears in Section B.2 in the Supplementary Material.

4.4 Extensions and Notation

The rest of the paper is devoted to using $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, Functionality 4.1, to obtain honest majority MPC with G.O.D., with our claimed communication and round complexities. In the real world, this functionality corresponds to our packed DSS, but from now on we will work with the $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ abstraction, which allows us to ignoring details regarding shares, degrees, and other aspects only needed to instantiate this functionality. $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ is implicitly parameterized by ℓ and m, and it models a simple but quite powerful arithmetic black box: parties can store vectors of dimension $m\ell$, and these vectors can be computed on by adding them together, as well as multiplying them element-wise by public constant vectors. Furthermore, any stored vector can be reconstructed, and the only way for the adversary to stop it is to reveal the identities of at least $t - 2(\ell - 1)$ corrupt parties. Recall that, for $\boldsymbol{s} \in \mathbb{F}^{m\ell}$, we denote $[\![\boldsymbol{s}]\!]$ a value stored in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ with $\mathsf{isMult} = 0$, and $[\![\boldsymbol{s}]\!]_*$ if $\mathsf{isMult} = 1$. Recall that given a public value $\boldsymbol{u} \in \mathbb{F}^{m\ell}$, it is possible to compute $[\![\boldsymbol{s}]\!]_* \leftarrow [\![\boldsymbol{s}]\!] * \boldsymbol{u}$. Addition of stored values and addition by public values is also possible. We use $[\![\boldsymbol{a}]\!] \leftarrow \mathsf{share}(\boldsymbol{a})$ to denote sharing, and $\boldsymbol{a} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{a}]\!])$ to denote reconstruction (this also applies to $[\![\cdot]\!]_*$).

To be able to work with this functionality effectively, we will add to it a few helpful instructions that can be easily instantiated based on what we have seen so far. These include multiplication by scalars and addition by constants, which are particularly useful in the MPC context. The $[\![\cdot]\!]$ notation suggestively represents these operations. Finally, we add an instruction, whose call we abbreviate by $r \leftarrow \mathsf{rand}()$, which allows the honest parties to obtain a fresh random value. This can be instantiated with a communication of $O(n^4)$, *cf.* Section 4.2.

5 Our MPC Protocol

We are now ready to put together the building blocks developed in previous sections to construct our final MPC protocol for honest majority with G.O.D.. As mentioned in technical overview (Section 1.3), the overall structure of our protocol is inspired on that of Turbopack [Esc+22], which is particularly suitable for the use of packed secret-sharing, a crucial tool we make use of in our work. While Turbopack uses plain packed secret-sharing, we make use of our optimized detectable secret-sharing, together with its reconstruction properties.

First, we define the MPC functionality with G.O.D. we aim at instantiating in this work. Let C be an arithmetic circuit over a finite field $\mathbb{F}$ comprised of inputs, addition and multiplication gates, and outputs. Each party P_i is responsible of providing a subset of the inputs. All parties are intended to receive the outputs. We use Greek letters α, β, γ, etc. to label wires in the circuit. We aim at instantiating $\mathcal{F}_{\mathsf{MPC}}$, Functionality 3, described below.

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

The functionality proceeds as follows:

- **Receive inputs:** Upon receiving $(\mathsf{input}, P_i, x, \alpha)$ from an honest party P_i, or from the adversary if P_i is corrupt, where $x \in \mathbb{F}$ and α is an input wire assigned to P_i, store (α, x)
- **Compute the circuit:** Once all inputs have been provided, compute the circuit C on these inputs. For every output wire α, if its associated result is y, send (α, y) to all parties.

MPC for $t < n/3$. As part of our protocol, we will need an MPC protocol with G.O.D. for $t < n/3$. We model this with a functionality that behaves *almost* exactly the same as $\mathcal{F}_{\mathsf{MPC}}$, with the only difference being that (1) it interacts only with a subset of the parties, aborting if the subset has at least a $1/3$ fraction of corruptions, and (2) it allows for *reactive* computation, meaning that different functions can be computed on the fly.[10] We denote this functionality by $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ The recent work of [Abr+23] instantiates $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ with linear communication $O(n|C'|)$ while maintaining the number of rounds $O(\mathsf{depth}(C'))$, where C' is the function being computed (we will use $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ with a function C' that is slightly different to C, but has roughly the same size and depth). For the purpose of this section we use $[x]$ when a value $x \in \mathbb{F}$ has been provided as input to $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, and we say "$P_i$ inputs x, obtaining $[x]$".

5.1 Offline Phase

We make use of two instances of $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$. To clearly differentiate between the two, we make the dependency of $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ with ℓ and m explicit by writing $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$. The first instance $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ allows parties to "share" or store vectors $\boldsymbol{s} \in \mathbb{F}^{m \cdot \ell}$, with $\ell = \lfloor \frac{n+6}{8} \rfloor$ and $m = n$. In what follows we use indistinctly "shared" and "stored" values/vectors, since even though we will be working in the $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$-hybrid model, in the real world these corresponds to sharings. Recall from Section 4.4 that we use $[\![\boldsymbol{s}]\!]$ and $[\![\boldsymbol{s}]\!]_*$ to denote secret-shared vectors with $\mathsf{isMult} = 0$ and $\mathsf{isMult} = 1$. This, in the real world, corresponds to sharings of degree $t+(\ell-1)$ and $t+2(\ell-1)$ respectively, and the crucial difference is that first type of sharings allows for multiplications by public values whereas the latter does not. Reconstructions of $[\![\boldsymbol{s}]\!]$ and $[\![\boldsymbol{s}]\!]_*$ shared values may abort, at the expense of identifying more than $t-(\ell-1)$ or $t-2(\ell-1)$ corrupt parties respectively. The second instance $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1)$ shares individual values $s \in \mathbb{F}$, and these stored values are denoted by $\langle s \rangle$. Here, the adversary cannot cause abort when reconstructing shared values. Throughout this section, we denote $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}^{m\ell}$, and for $i \in [m\ell]$ we write $\mathbf{1}_i \in \mathbb{F}^{m\ell}$ for the vector of all zeros, except for the i-th entry, which equals 1.

[10] $\mathcal{F}_{\mathsf{MPC}}$, as defined, is not reactive. However, this is only for presentation and it is not hard to extend our protocol to support reactive computation.

Our preprocessing is as in Turbopack [Esc+22]. First, we group multiplication gates in each layer in groups of $m \cdot \ell$ gates each, and we do the same with the input wires associated to each party, as well as the output wires. Each circuit wire α that is not the output of an addition gate has associated to it a random mask $\lambda_\alpha \in \mathbb{F}$. If two wires α, β are added to obtain wire γ, then $\lambda_\gamma := \lambda_\alpha + \lambda_\beta$. The preprocessing consists of sharings $[\![\boldsymbol{\lambda_\alpha}]\!]_*$ for every output group $\boldsymbol{\alpha}$, and sharings $([\![\boldsymbol{\lambda_\alpha}]\!], [\![\boldsymbol{\lambda_\beta}]\!], [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*)$ for every group of multiplication gates with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$. In addition, every party P_i having an input wire α must learn λ_α. For the case of a restart, we also require every such λ_α for input wires to be VSS'ed as $\langle \lambda_\alpha \rangle$.[11] This is captured by $\mathcal{F}_{\mathsf{Prep}}$, Functionality 4.

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

Extension of the Packed DSS functionality. $\mathcal{F}_{\mathsf{Prep}}$ has all of the instructions of $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1, 1)$ (extended as in Section 4.4).[a]

Sample random masks. Sample the following values

1. For each circuit wire α that is *not* the result of an addition gate, sample a random $\lambda_\alpha \in \mathbb{F}$ and store (α, λ_α).
2. For every addition gate with inputs α, β and output γ, compute $\lambda_\gamma = \lambda_\alpha + \lambda_\beta$ and store (γ, λ_γ)

Input and output sharings. For every group of $m\ell$ input gates with labels $\boldsymbol{\alpha}$ belonging to party P_i:

1. Send $\boldsymbol{\lambda_\alpha}$ to P_i,
2. Store $\langle \lambda_{\alpha_j} \rangle$ for $j \in [m\ell]$.

For every group of $m\ell$ output gates with labels $\boldsymbol{\alpha}$, store $[\![\boldsymbol{\lambda_\alpha}]\!]$

Multiplication gates. For every group of $m\ell$ multiplication gates with left input labels $\boldsymbol{\alpha}$, right input labels $\boldsymbol{\beta}$, and output labels $\boldsymbol{\gamma}$, the functionality stores $([\![\boldsymbol{\lambda_\alpha}]\!], [\![\boldsymbol{\lambda_\beta}]\!], [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*)$.

[a] Values stored as in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ are kept in a separate dictionary than these from $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1, 1)$.

Multiplication triple generation. For our preprocessing we will require uniformly random multiplication triples $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!]_*)$, with $\boldsymbol{c} = \boldsymbol{a} \star \boldsymbol{b}$. To this end, we show how to extend the techniques from [CP17], which are set in the standard (non-packed) secret-sharing setting, to the packed secret-sharing regime we use in our work. We choose the techniques from [CP17] since, in contrast to other approaches such as [DN07], no "degree-$2t$ computations" are needed, and instead all shares are either degree $t + (\ell - 1)$, or $t + 2(\ell - 1)$. This is crucial for us, where we require reconstruction to either succeed, or identify a large set of corrupt parties. We adapt the techniques from [CP17] to our setting below.

Let $m \in \Theta(n)$ be a batching parameter and $\ell = \Theta(n)$ be a packing parameter. The goal of this section is to show how the parties can preprocess

[11] Having each input to be VSS'ed adds an extra factor of n with respect to the number of inputs. We present in Section C in the Supplementary Material a variant that is more suitable incase there are many more inputs than outputs.

sharings of a "packed triple" in the $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$-hybrid model, which has the form $(\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket, \llbracket \boldsymbol{c} \rrbracket_*)$, where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^{m\ell}$ are uniformly random and $\boldsymbol{c} = \boldsymbol{a} \star \boldsymbol{b}$. To this end, we will design a procedure $\pi_{\mathsf{triple\text{-}generation}}$, which is a direct adaptation of the techniques from [CP17]. We split this into two phases called "Verifiable Triple Sharing" and "Triple Extraction" which we describe next.

Verifiable Triple Sharing (Phase 1) : This procedure, denoted by $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$, enables a dedicated party P to act as a dealer and share a batched triple among all parties. The guarantee is that a corrupt dealer cannot cause the procedure to output an invalid batched triple except with negligible probability.

Procedure 10: $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}(P)$

This procedure enables a single dedicated party P to act as a dealer and *verifiably* share a triple among all parties.[a]

1. P samples a batched triple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ and obtains $\llbracket \boldsymbol{a} \rrbracket \leftarrow \mathsf{share}(\boldsymbol{a}), \llbracket \boldsymbol{b} \rrbracket \leftarrow \mathsf{share}(\boldsymbol{b}), \llbracket \boldsymbol{c} \rrbracket_* \leftarrow \mathsf{share}(\boldsymbol{c})$.
2. P samples another batched triple $(\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma})$ and obtains $\llbracket \boldsymbol{\alpha} \rrbracket \leftarrow \mathsf{share}(\boldsymbol{\alpha}), \llbracket \boldsymbol{\beta} \rrbracket \leftarrow \mathsf{share}(\boldsymbol{\beta}), \llbracket \boldsymbol{\gamma} \rrbracket_* \leftarrow \mathsf{share}(\boldsymbol{\gamma})$.
3. $e \leftarrow \mathsf{rand}()$[b]
4. Compute $\llbracket \boldsymbol{\rho} \rrbracket \leftarrow e \cdot \llbracket \boldsymbol{a} \rrbracket - \llbracket \boldsymbol{\alpha} \rrbracket$. Then $\boldsymbol{\rho} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{\rho} \rrbracket)$
5. Compute $\llbracket \boldsymbol{\sigma} \rrbracket \leftarrow \llbracket \boldsymbol{b} \rrbracket - \llbracket \boldsymbol{\beta} \rrbracket$. Then $\boldsymbol{\sigma} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{\sigma} \rrbracket)$
6. Compute $\llbracket \boldsymbol{z} \rrbracket_* \leftarrow e \cdot \llbracket \boldsymbol{c} \rrbracket_* - \llbracket \boldsymbol{\gamma} \rrbracket_* - \boldsymbol{\sigma} \cdot \llbracket \boldsymbol{\alpha} \rrbracket - \boldsymbol{\rho} \cdot \llbracket \boldsymbol{\beta} \rrbracket - \boldsymbol{\rho} \cdot \boldsymbol{\sigma}$.
7. $\boldsymbol{z} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{z} \rrbracket_*)$.
8. If $\boldsymbol{z} \neq \boldsymbol{0}$ output $(\llbracket \boldsymbol{0} \rrbracket, \llbracket \boldsymbol{0} \rrbracket, \llbracket \boldsymbol{0} \rrbracket_*)$; otherwise output $(\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket, \llbracket \boldsymbol{c} \rrbracket_*)$.

[a] If any of the sharings abort in Steps 1. and 2., the parties skip to step 8.
[b] As in $\pi_{\mathsf{Input\text{-}Sharings}}$, one single random value can be reused across all parties, and this is important for efficiency.

Lemma 11. *If P is honest, then $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ outputs $(\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket, \llbracket \boldsymbol{c} \rrbracket_*)$ with probability 1. If P is corrupt, then $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ either aborts, or outputs a valid batched triple except with $\mathtt{negl}(\kappa)$ failure probability.*

The proof appears in Section B.3 in the Supplementary Material.

Cost analysis of $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$: Steps 1, 2, 4, 5 and 7 cost $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(1)$. Step 3 costs $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$. All other steps are local. Hence the total cost of $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ is $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$.

Triple Randomness extractor Before showing Phase 2 of Triple Extraction, we present the triple extractor of [CP17].

Suppose we have a source distribution R which outputs $n = 2t + 1$ triples, $(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)$, out of which $n - t = t + 1$ are guaranteed to be uniform and independently sampled. [CP17] designed a deterministic extractor $\mathcal{E}_{\mathrm{triple}}$

which outputs $n - 2t = 1$ new triple, $(a_{\text{new}}, b_{\text{new}}, c_{\text{new}})$, such that this new triple is uniform and independent from any set of at most t source triples. We describe the extractor algorithm below:

$\underline{\mathcal{E}_{\text{triple}}((a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)) :}$

Parameter: Let $\lambda_j^{(i)} \in \mathbb{F}$ denote the j^{th} lagrange coefficient of a degree t polynomial evaluted at point $i \in \mathbb{F}$. More formally, $\lambda_j^{(i)}$ is the result of evaluating the lagrange polynomial $\frac{\Pi_{k \in [1,t+1], k \neq j}(x-k)}{\Pi_{k \in [1,t+1], k \neq j}(j-k)}$ on the point $x = i$. Similarly, $\gamma_j^{(i)} \in \mathbb{F}$ denote the j^{th} lagrange coefficient of a degree $2t$ polynomial evaluted at point $i \in \mathbb{F}$.

1. Compute an auxiliary set of values $(a'_{t+2}, b'_{t+2}), \ldots, (a'_n, b'_n)$ s.t.

$$\forall i \in [t+2, n] : \begin{cases} a'_i = \lambda_1^{(i)} \cdot a_1 + \ldots + \lambda_{t+1}^{(i)} \cdot a_{t+1} \\ b'_i = \lambda_1^{(i)} \cdot b_1 + \ldots + \lambda_{t+1}^{(i)} \cdot b_{t+1} \end{cases}$$

2. Compute an auxiliary set of values $c'_{t+2}, \ldots, c'_n$ s.t.

$$\forall i \in [t+2, n] : c'_i = a'_i \cdot b'_i$$

3. Output $(a_{\text{new}}, b_{\text{new}}, c_{\text{new}})$ s.t.

$$a_{\text{new}} = \lambda_1^{(n+1)} \cdot a_1 + \ldots + \lambda_{t+1}^{(n+1)} \cdot a_{t+1}$$

$$b_{\text{new}} = \lambda_1^{(n+1)} \cdot b_1 + \ldots + \lambda_{t+1}^{(n+1)} \cdot b_{t+1}$$

$$c_{\text{new}} = \begin{cases} \gamma_1^{(n+1)} \cdot c_1 + \ldots + \gamma_t^{(n+1)} \cdot c_t \\ + \\ \gamma_{t+1}^{(n+1)} \cdot c'_{t+1} \ldots + \gamma_{2t+1}^{(n+1)} \cdot c'_{2t+1} \end{cases}$$

Lemma 12. *Given $n = 2t + 1$ source triples, $(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)$, out of which at least $n - t$ triples are uniformly random and independent, $\mathcal{E}_{triple}$ outputs a triple $(a_{new}, b_{new}, c_{new})$ which is uniform and independent from any set of atmost t source triples.*

The proof appears in Section B.3 in the Supplementary Material.

Triple Extraction (Phase 2) : Consider n valid triples, $(\llbracket \boldsymbol{a}_1 \rrbracket, \llbracket \boldsymbol{b}_1 \rrbracket, \llbracket \boldsymbol{c}_1 \rrbracket_*), \ldots, (\llbracket \boldsymbol{a}_n \rrbracket, \llbracket \boldsymbol{b}_n \rrbracket, \llbracket \boldsymbol{c}_n \rrbracket_*)$, out of which atmost t are known and fixed by the adversary and the remaining are uniformly random and independent. Procedure $\pi_{\mathsf{triple\text{-}extraction}}$ below deterministically extracts a new uniformly random triple $(\llbracket \boldsymbol{a}_{\text{new}} \rrbracket, \llbracket \boldsymbol{b}_{\text{new}} \rrbracket, \llbracket \boldsymbol{c}_{\text{new}} \rrbracket))$ which will be uniform and independent from the view of the adversary. This is done by executing $\mathcal{E}_{\text{triple}}$ randomness extractor shown above securely. The observation is that Step 1 and Step 3 in $\mathcal{E}_{\text{triple}}$ involve only linear operations on the secrets hence they can be computed using add and mult instructions provided by $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$. For securely computing Step 2 in $\mathcal{E}_{\text{triple}}$,

which involves multiplying two secret values, we will use a sub-routine π_{Beaver} which will perform the classic beaver multiplication of two secret batched values by consuming some input batched triples as preprocessing material.

Procedure 11:

proc:tripleextraction $\pi_{\mathsf{triple\text{-}extraction}}((\llbracket \boldsymbol{a_1} \rrbracket, \llbracket \boldsymbol{b_1} \rrbracket, \llbracket \boldsymbol{c_1} \rrbracket), \ldots, (\llbracket \boldsymbol{a_n} \rrbracket, \llbracket \boldsymbol{b_n} \rrbracket, \llbracket \boldsymbol{c_n} \rrbracket))$
This procedure uses n batched triples, $(\llbracket \boldsymbol{a_1} \rrbracket, \llbracket \boldsymbol{b_1} \rrbracket, \llbracket \boldsymbol{c_1} \rrbracket), \ldots, (\llbracket \boldsymbol{a_n} \rrbracket, \llbracket \boldsymbol{b_n} \rrbracket, \llbracket \boldsymbol{c_n} \rrbracket)$, stored in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, and computes a new batched triple, $(\llbracket \boldsymbol{a}_{\text{new}} \rrbracket, \llbracket \boldsymbol{b}_{\text{new}} \rrbracket, \llbracket \boldsymbol{c}_{\text{new}} \rrbracket))$, stored in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$.
Public Parameters: Let $\lambda_j^{(i)} \in \mathbb{F}$ denote the j^{th} lagrange coefficient of a degree t polynomial evaluted at point $i \in \mathbb{F}$. More formally, $\lambda_j^{(i)}$ is the result of evaluating the lagrange polynomial $\frac{\Pi_{k \in [1,t+1], k \neq j}(x-k)}{\Pi_{k \in [1,t+1], k \neq j}(j-k)}$ on the point $x = i$. Similarly, $\gamma_j^{(i)} \in \mathbb{F}$ denote the j^{th} lagrange coefficient of a degree $2t$ polynomial evaluted at point $i \in \mathbb{F}$.

1. Compute an auxiliary set of values $(\llbracket \boldsymbol{a'_{t+2}} \rrbracket, \llbracket \boldsymbol{b'_{t+2}} \rrbracket), \ldots, (\llbracket \boldsymbol{a'_n} \rrbracket, \llbracket \boldsymbol{b'_n} \rrbracket)$ s.t.

$$\forall i \in [t+2, n] : \begin{cases} \llbracket \boldsymbol{a'_i} \rrbracket = \lambda_1^{(i)} \cdot \llbracket \boldsymbol{a_1} \rrbracket + \cdots + \lambda_{t+1}^{(i)} \cdot \llbracket \boldsymbol{a_{t+1}} \rrbracket \\ \llbracket \boldsymbol{b'_i} \rrbracket = \lambda_1^{(i)} \cdot \llbracket \boldsymbol{b_1} \rrbracket + \cdots + \lambda_{t+1}^{(i)} \cdot \llbracket \boldsymbol{b_{t+1}} \rrbracket \end{cases}$$

   where the $\lambda_j^{(i)}$'s are Lagrange coefficients of a degree t polynomial.
2. Compute an auxiliary set of values $\llbracket \boldsymbol{c'_{t+2}} \rrbracket_*, \ldots, \llbracket \boldsymbol{c'_n} \rrbracket_*$ s.t.

$$\forall i \in [t+2, n] : \llbracket \boldsymbol{c'_i} \rrbracket_* = \pi_{\mathsf{Beaver}}((\llbracket \boldsymbol{a'_i} \rrbracket, \llbracket \boldsymbol{b'_i} \rrbracket), (\llbracket \boldsymbol{a_i} \rrbracket, \llbracket \boldsymbol{b_i} \rrbracket, \llbracket \boldsymbol{c_i} \rrbracket))$$

3. Output $(\llbracket \boldsymbol{a}_{\text{new}} \rrbracket, \llbracket \boldsymbol{b}_{\text{new}} \rrbracket, \llbracket \boldsymbol{c}_{\text{new}} \rrbracket_*)$ s.t.

$$\llbracket \boldsymbol{a}_{\text{new}} \rrbracket = \lambda_1^{(n+1)} \cdot \llbracket \boldsymbol{a_1} \rrbracket + \ldots + \lambda_{t+1}^{(n+1)} \cdot \llbracket \boldsymbol{a_{t+1}} \rrbracket$$
$$\llbracket \boldsymbol{b}_{\text{new}} \rrbracket = \lambda_1^{(n+1)} \cdot \llbracket \boldsymbol{b_1} \rrbracket + \ldots + \lambda_{t+1}^{(n+1)} \cdot \llbracket \boldsymbol{b_{t+1}} \rrbracket$$
$$\llbracket \boldsymbol{c}_{\text{new}} \rrbracket_* = \begin{cases} \gamma_1^{(n+1)} \cdot \llbracket \boldsymbol{c_1} \rrbracket + \ldots + \gamma_{t+1}^{(n+1)} \cdot \llbracket \boldsymbol{c_{t+1}} \rrbracket \\ + \\ \gamma_{t+2}^{(n+1)} \cdot \llbracket \boldsymbol{c'_{t+2}} \rrbracket_* \ldots \gamma_{2t+1}^{(n+1)} \cdot \llbracket \boldsymbol{c'_{2t+1}} \rrbracket_* \end{cases}$$

Lemma 13 (Triple Extraction Lemma[CP17]). *Given n valid batched triples, $(\llbracket \boldsymbol{a_1} \rrbracket, \llbracket \boldsymbol{b_1} \rrbracket, \llbracket \boldsymbol{c_1} \rrbracket), \ldots, (\llbracket \boldsymbol{a_n} \rrbracket, \llbracket \boldsymbol{b_n} \rrbracket, \llbracket \boldsymbol{c_n} \rrbracket))$, out of which the adversary knows and fixes at most t batched triples and the remaining $n-t$ batched triples are uniform and independent, $\pi_{\mathsf{triple\text{-}extraction}}$ will output a uniform triple $(\llbracket \boldsymbol{a}_{new} \rrbracket, \llbracket \boldsymbol{b}_{new} \rrbracket, \llbracket \boldsymbol{c}_{new} \rrbracket_*)$ which will be independent from the view of the adversary.*

Proof. Since $\pi_{\mathsf{triple\text{-}extraction}}$ is line-by-line evaluating securely the extractor from Section 5.1, it follows from Lemma 12 that the output triple is uniformly random and correct. □

Cost analysis of $\pi_{\mathsf{triple\text{-}extraction}}$: Step 2 performs $O(n)$ invocations of π_{Beaver} where each invocation costs $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(1)$. Hence, the total cost of Step

2 is $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$. Since the remaining steps are local, the overall cost of $\pi_{\mathsf{triple\text{-}extraction}}$ is $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$.

Procedure 12: $\pi_{\mathsf{Beaver}}((\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket), (\llbracket \boldsymbol{u} \rrbracket, \llbracket \boldsymbol{v} \rrbracket, \llbracket \boldsymbol{w} \rrbracket))$

This procedure takes two batched inputs $(\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket)$ along with a batched triple $(\llbracket \boldsymbol{u} \rrbracket, \llbracket \boldsymbol{v} \rrbracket, \llbracket \boldsymbol{w} \rrbracket)$ stored in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, and computes an output $\llbracket \boldsymbol{c} \rrbracket_*$ stored in $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ s.t. $\boldsymbol{c} = \boldsymbol{a} \cdot \boldsymbol{b}$.

1. $\llbracket \widetilde{\boldsymbol{a}} \rrbracket \leftarrow \llbracket \boldsymbol{a} \rrbracket - \llbracket \boldsymbol{u} \rrbracket$. Then $\widetilde{\boldsymbol{a}} \leftarrow \mathsf{reconstruct}(\llbracket \widetilde{\boldsymbol{a}} \rrbracket)$.
2. $\left\llbracket \widetilde{\boldsymbol{b}} \right\rrbracket \leftarrow \llbracket \boldsymbol{b} \rrbracket - \llbracket \boldsymbol{v} \rrbracket$. Then $\widetilde{\boldsymbol{b}} \leftarrow \mathsf{reconstruct}(\left\llbracket \widetilde{\boldsymbol{b}} \right\rrbracket)$.
3. Output $\llbracket \boldsymbol{c} \rrbracket_*$ where $\llbracket \boldsymbol{c} \rrbracket_* \leftarrow \llbracket \boldsymbol{w} \rrbracket + \widetilde{\boldsymbol{a}} \cdot \llbracket \boldsymbol{v} \rrbracket + \widetilde{\boldsymbol{b}} \cdot \llbracket \boldsymbol{u} \rrbracket + \widetilde{\boldsymbol{a}} \cdot \widetilde{\boldsymbol{b}}$

Cost analysis of π_{Beaver}*:* Step 1 and Step 2 each cost $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(1)$ whereas Step 3 is local. Hence, the overall cost of π_{Beaver} is $\mathsf{P2P}(O(n^3))$, $O(n^3) \times \mathsf{BC}(1)$.

Triple Generation: Our overall procedure for batched triple generation $\pi_{\mathsf{triple\text{-}generation}}$ will simply combine the individual phases of Verifiable Triple Sharing (Phase 1) and Triple Extraction (Phase 2) in the following natural way. Each party will act as a dealer in an invocation of $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ and share a batched triple. Then we invoke $\pi_{\mathsf{triple\text{-}extraction}}$ on all the n shared batched triples to extract a new batched triple.

Procedure 13: $\pi_{\mathsf{triple\text{-}generation}}$

This procedure generates a single batched triple $(\llbracket \boldsymbol{a}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{b}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{c}_{\mathrm{new}} \rrbracket_*)$ which will be uniform and independent from the view of corrupt parties.

1. For all $i \in [n]$ in parallel, compute $(\llbracket \boldsymbol{a_i} \rrbracket, \llbracket \boldsymbol{b_i} \rrbracket, \llbracket \boldsymbol{c_i} \rrbracket) \leftarrow \pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}(P_i)$.
2. Compute $(\llbracket \boldsymbol{a}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{b}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{c}_{\mathrm{new}} \rrbracket_*) \leftarrow \pi_{\mathsf{triple\text{-}extraction}} \begin{pmatrix} (\llbracket \boldsymbol{a_1} \rrbracket, \llbracket \boldsymbol{b_1} \rrbracket, \llbracket \boldsymbol{c_1} \rrbracket) \\ \cdots \\ (\llbracket \boldsymbol{a_n} \rrbracket, \llbracket \boldsymbol{b_n} \rrbracket, \llbracket \boldsymbol{c_n} \rrbracket) \end{pmatrix}$
3. Output $(\llbracket \boldsymbol{a}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{b}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{c}_{\mathrm{new}} \rrbracket_*)$.

Lemma 14. $\pi_{\mathsf{triple\text{-}generation}}$ *outputs a batched triple* $(\llbracket \boldsymbol{a}_{new} \rrbracket, \llbracket \boldsymbol{b}_{new} \rrbracket, \llbracket \boldsymbol{c}_{new} \rrbracket_*)$ *which is uniform and independent from the view of an adversary corrupting at most* t *parties except with* $\mathtt{negl}(\kappa)$ *failure probability.*

Proof. From Lemma 11, the triples provided by $n - t$ honest parties are random, and the t triples provided by the adversary are correct (except with $\mathtt{negl}(\kappa)$ probability). Hence, from Lemma 13, the returned triple $(\llbracket \boldsymbol{a}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{b}_{\mathrm{new}} \rrbracket, \llbracket \boldsymbol{c}_{\mathrm{new}} \rrbracket_*)$ is correct and uniformly random towards the adversary. □

Cost analysis of $\pi_{\mathsf{triple\text{-}generation}}$***:*** Step 1 performs $O(n)$ invocations of $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ where each invocation costs $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$. Thus, the total cost of Step 1 is $\mathsf{P2P}(O(n^5))$, $O(n^5) \times \mathsf{BC}(1)$. We note that this can be reduced to $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$ by using a single call of $\mathsf{rand}()$ across all the $O(n)$ parallel invocations of $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$. Step 2 costs $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$ and Step 3 is local. So the overall cost of $\pi_{\mathsf{triple\text{-}generation}}$ is $\mathsf{P2P}(O(n^4))$, $O(n^4) \times \mathsf{BC}(1)$. Since $\pi_{\mathsf{triple\text{-}generation}}$ outputs a single batched triple containing $O(m\ell) = O(n^2)$ triples, the amortized communication cost per triple generation is $\mathsf{P2P}(O(n^2))$, $O(n^2) \times \mathsf{BC}(1)$.

Useful procedures. Before we describe the protocol that instantiates $\mathcal{F}_{\mathsf{Prep}}$, we describe a few useful procedures. The first, $\pi_{\mathsf{Input\text{-}Sharings}}(P_i)$ (Procedure 14), enables the parties to obtain random sharings of the form $(\llbracket r \cdot \mathbf{1} \rrbracket, \langle r \rangle)$, where P_i knows r. This will be important for providing inputs, with the VSS part enabling restarting without input modification. The procedure follows along the same lines as $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, Procedure 4.2, which lets P_i distribute these sharings and the parties check them via random linear combinations. The second procedure, which we denote by $\pi_{\mathsf{Rand\text{-}Sharings}}$ (Procedure 15), allows the parties to obtain $\llbracket r \cdot \mathbf{1} \rrbracket$, where $r \in \mathbb{F}$ is uniformly random and unknown to any party. This first uses ideas as in the first procedure to let each party distribute one such sharing correctly, and then, similarly to $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$ (Procedure 4.2), we can use standard techniques based on Vandermonde matrices to extract uniformly random sharings.

Procedure 14: $\pi_{\mathsf{Input\text{-}Sharings}}(P_i)$

This procedure lets P_i share a pair $(\llbracket r \cdot \mathbf{1} \rrbracket, \langle r \rangle)$, where $r \in \mathbb{F}$ is random.

1. P_i samples $r, r' \in \mathbb{F}$ and calls $\llbracket r \cdot \mathbf{1} \rrbracket \leftarrow \mathsf{share}(r \cdot \mathbf{1})$ and $\llbracket r \cdot \mathbf{1} \rrbracket \leftarrow \mathsf{share}(r' \cdot \mathbf{1})$, and also $\langle r \rangle \leftarrow \mathsf{share}(r)$ and $\langle r' \rangle \leftarrow \mathsf{share}(r')$ (the first two with $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$, the second with $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1, 1)$).[a]
2. The parties then call $\beta \leftarrow \mathsf{rand}()$[b] and compute $\llbracket \boldsymbol{z} \rrbracket \leftarrow \beta \cdot \llbracket r \cdot \mathbf{1} \rrbracket - \llbracket r' \cdot \mathbf{1} \rrbracket$, as well as $\langle z' \rangle \leftarrow \beta \cdot \langle r \rangle - \langle r' \rangle$
3. Parties call $\boldsymbol{z} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{z} \rrbracket)$ and $z' \leftarrow \mathsf{reconstruct}(\langle z' \rangle)$. If $\boldsymbol{z} \neq z' \cdot \mathbf{1}$, then output $(\llbracket \mathbf{0} \rrbracket, \langle 0 \rangle)$. Else, output $(\llbracket r \cdot \mathbf{1} \rrbracket, \langle r \rangle)$

[a] The parties abort if any of these sharings abort, as we know P_i is corrupted in this case.

[b] One single random value can be reused across all parties, and this is important for efficiency.

Procedure 15: $\pi_{\mathsf{Rand\text{-}Sharings}}$

This procedure has each party share batches of $\llbracket r \cdot \mathbf{1} \rrbracket$, and outputs batches of $n - t$ such random sharings. Initialize $T = \emptyset$.

1. For every party P_i, the parties do the following:

(a) P_i samples $r_i, r'_i \in \mathbb{F}$ and calls $[\![r_i \cdot \mathbf{1}]\!] \leftarrow \mathsf{share}(r_i \cdot \mathbf{1})$ and $[\![r_i \cdot \mathbf{1}]\!] \leftarrow \mathsf{share}(r'_i \cdot \mathbf{1})$.[a]
(b) The parties then call $\beta \leftarrow \mathsf{rand}()$[b] and compute $[\![\boldsymbol{z}_i]\!] \leftarrow \beta \cdot [\![r_i \cdot \mathbf{1}]\!] - [\![r'_i \cdot \mathbf{1}]\!]$.
(c) Parties call $\boldsymbol{z}_i \leftarrow \mathsf{reconstruct}([\![\boldsymbol{z}_i]\!])$. If $\boldsymbol{z}_i \neq z_i \cdot \mathbf{1}$ for some $z_i \in \mathbb{F}$, add P_i to the set T.

2. For $P_i \in T$ set $r_i := 0$ and store $[\![r_i \cdot \mathbf{1}]\!]$ for this party.
3. All parties then compute and output

$$([\![s_1 \cdot \mathbf{1}]\!], \ldots, [\![s_{n-t} \cdot \mathbf{1}]\!])^\intercal \leftarrow \boldsymbol{M} \cdot ([\![r_1 \cdot \mathbf{1}]\!], \ldots, [\![r_n \cdot \mathbf{1}]\!]).$$

[a] If the parties abort during this step, they use a canonical sharing of all-zeros in place of the failed sharing, as we know the dealer is corrupted.
[b] One single random value can be reused across all parties, and this is important for efficiency.

The following two Lemmas are proven similarly to Lemma 10, we omit their proof.

Lemma 15. *Except with probability* $\mathtt{negl}(\kappa)$*, the output* $[\![r \cdot \mathbf{1}]\!], \langle r \rangle$ *produced by* $\pi_{\mathsf{Input\text{-}Sharings}}(P_i)$ *is correct, and for an honest* P_i*, the secret* r *is distributed randomly given the corrupted parties' shares.*

Lemma 16. *Except with probability* $\mathtt{negl}(\kappa)$*, the outputs* $([\![s_1 \cdot \mathbf{1}]\!], \ldots, [\![s_{n-t} \cdot \mathbf{1}]\!])$ *produced by* $\pi_{\mathsf{Rand\text{-}Sharings}}$ *are correct, and are distributed randomly given the corrupted parties' shares.*

Preprocessing protocol. We are finally ready to present our protocol for instantiating $\mathcal{F}_{\mathsf{Prep}}$. This is given in Π_{Prep}, Protocol 16 below.

Protocol 16: Π_{Prep}

The protocol makes use of two functionalities $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1)$, and every command regarding packed DSS is forwarded to these functionalities. For the other commands:

Input groups. For every input wire α associated to party P_i, call $([\![\lambda_\alpha \cdot \mathbf{1}]\!], \langle \lambda_\alpha \rangle) \leftarrow \pi_{\mathsf{Input\text{-}Sharings}}(P_i)$ for $i \in [n]$.[a]

Sampling random masks. For every wire α that is either an output of a multiplication gate, or an input wire, the parties call $[\![\lambda_\alpha \cdot \mathbf{1}]\!] \leftarrow \pi_{\mathsf{Rand\text{-}Sharings}}()$. After this note that, locally, they can compute $[\![\lambda_\gamma \cdot \mathbf{1}]\!]$ for every wire γ that is the output of an addition gate by adding the corresponding shares. This means they have $[\![\lambda_{\alpha_i} \cdot \mathbf{1}]\!]$ for *every* circuit wire α.

Output groups. For an output group $\boldsymbol{\alpha}$, the parties take the sharings $[\![\lambda_{\alpha_i} \cdot \mathbf{1}]\!]$ for $i \in [m\ell]$ from the previous step and output $[\![\boldsymbol{\lambda_\alpha}]\!]_* = \sum_{i=1}^{m\ell} \mathbf{1}_i \cdot [\![\lambda_{\alpha_i} \mathbf{1}]\!]$.

Multiplication groups. For a multiplication group with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, the parties do the following:

1. Call $\pi_{\mathsf{triple\text{-}generation}}$ to obtain $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{a} \star \boldsymbol{b}]\!]_*)$

2. The parties proceed as before, obtaining $[\![\boldsymbol{\lambda_\alpha}]\!]_*$ and $[\![\boldsymbol{\lambda_\beta}]\!]_*$. Similarly, they get $[\![\boldsymbol{\lambda_\gamma}]\!]_*$
3. Locally compute $[\![\boldsymbol{d}]\!]_* \leftarrow [\![\boldsymbol{\lambda_\alpha}]\!]_* - [\![\boldsymbol{a}]\!]$ and $[\![\boldsymbol{e}]\!]_* \leftarrow [\![\boldsymbol{\lambda_\beta}]\!]_* - [\![\boldsymbol{b}]\!]$
4. Call $\boldsymbol{d} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{d}]\!]_*)$ and $\boldsymbol{e} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{e}]\!]_*)$
5. Locally compute $[\![\boldsymbol{\lambda_\alpha}]\!] \leftarrow \boldsymbol{d} + [\![\boldsymbol{a}]\!]$, $[\![\boldsymbol{\lambda_\beta}]\!] \leftarrow \boldsymbol{e} + [\![\boldsymbol{b}]\!]$, and

$$[\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_* \leftarrow \boldsymbol{d} \cdot [\![\boldsymbol{a}]\!] + \boldsymbol{e} \cdot [\![\boldsymbol{b}]\!] + \boldsymbol{d} \star \boldsymbol{e} + [\![\boldsymbol{a} \star \boldsymbol{b}]\!]_* - [\![\boldsymbol{\lambda_\gamma}]\!]_* ,$$

and output $([\![\boldsymbol{\lambda_\alpha}]\!], [\![\boldsymbol{\lambda_\beta}]\!], [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*)$

Abort. Note that, if any of the steps above results in abort, then a set T of corrupt parties with $|T| > t - 2(\ell - 1)$ is identified. In this case the parties output this set.

[a] If the parties abort during this step, P_i's inputs will be disregarded, as we know they are corrupted.

Theorem 3. *Π_{Prep} UC-realizes $\mathcal{F}_{\mathsf{Prep}}$ in the $(\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m), \mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1))$-hybrid model, with probability $1 - \mathtt{negl}(\kappa)$.*

The proof appears in Section B.4 in the Supplementary Material.

Communication complexity. We now calculate the communication cost of Π_{Prep} by calculating the cost of different parts:

1. Input groups: This step invokes $\pi_{\mathsf{Input\text{-}Sharings}}$ k times where k is the number of input wires. Each invocation of $\pi_{\mathsf{Input\text{-}Sharings}}$ costs $\mathsf{P2P}(O(n^3)), O(n^3) \times \mathsf{BC}(1)$ (assuming the cost of $\mathsf{rand}()$ is amortized across n parties). Therefore, the total cost of this step is $\mathsf{P2P}(O(|C|n^3)), O(|C|n^3) \times \mathsf{BC}(1)$.
2. Sampling random masks: This step invokes $\pi_{\mathsf{Rand\text{-}Sharings}}$ $O(|C|/n)$ times. Each invocation of $\pi_{\mathsf{Rand\text{-}Sharings}}$ costs $\mathsf{P2P}(O(n^4)), O(n^4) \times \mathsf{BC}(1)$. (assuming the cost of $\mathsf{rand}()$ is amortized across n parties). Therefore, the total cost of this step is $\mathsf{P2P}(O(|C|n^3)), O(|C|n^3) \times \mathsf{BC}(1)$.
3. Multiplication groups: Let $k = |C|/n^2$ be the number of multiplication groups. This step invokes $\pi_{\mathsf{triple\text{-}generation}}$ k times where each invocation costs $\mathsf{P2P}(O(n^4)), O(n^4) \times \mathsf{BC}(1)$. Also, it performs a beaver multiplication (same as π_{Beaver}) k times where each multiplication costs $\mathsf{P2P}(O(n^3)), O(n^3) \times \mathsf{BC}(1)$. Therefore, the total cost of this step is $\mathsf{P2P}(O(|C|n^2)), O(|C|n^2) \times \mathsf{BC}(1)$.

Summing up all the above costs, the overall communication cost of Π_{Prep} is $\mathsf{P2P}(O(|C|n^3)), O(|C|n^3) \times \mathsf{BC}(1)$.

Remark 2 (On function-dependent/independent preprocessing.). As in [Esc+22], we can easily make our offline phase function-independent without affecting our asymptotic communication in the online phase. For this, the offline phase consists only of generating sharings of the form $[\![r \cdot \mathbf{1}]\!]$ and $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{a} \star \boldsymbol{b}]\!]_*)$ (which is function-independent), and the part of Π_{Prep} that turns these into the function-dependent $([\![\boldsymbol{\lambda_\alpha}]\!], [\![\boldsymbol{\lambda_\beta}]\!], [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*)$ is moved to the online phase Π_{MPC}. Crucially, the communication complexity of these steps is $O(n|C|)$.

5.2 Online Phase

Finally, we present Π_{MPC}, Protocol 17, which instantiates $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model. This corresponds to the online phase, and at a high level it proceeds by maitaining the following invariant. For a wire α and a given assignment to the circuit inputs, let us denote by v_α the value held by wire α. The protocol maintains that, for every wire α, the parties have the values $\mu_\alpha := v_\alpha - \lambda_\alpha$ in the clear. This is ensured all the way up to the outputs, point in which the parties can reconstruct the associated masks and obtain the outputs. A major difference with respect to Turbopack [Esc+22] is that, in our case, we need to handle the case in which any of the steps that involve reconstructions—either in the offline or online phase—result in abort. For this, we let the parties restart the computation, kicking out the identified corrupt parties, which guarantees the new corruption threshold is 1/3. The parties make use of the $t < n/3$ MPC functionality for this $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, but before doing that they use the initial VSS'ed masks $\langle \lambda_\alpha \rangle$ to ensure that the inputs provided to $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ are consistent with these from the initial execution that resulted in abort.

Protocol 17: Π_{MPC}

This protocol makes use of $\mathcal{F}_{\mathsf{Prep}}$ and $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$.

Preprocessing. The parties call $\mathcal{F}_{\mathsf{Prep}}$ to obtain:
- $[\![\boldsymbol{\lambda_\alpha}]\!]_*$ for every output group $\boldsymbol{\alpha}$
- $([\![\boldsymbol{\lambda_\alpha}]\!], [\![\boldsymbol{\lambda_\beta}]\!], [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*)$ for every multiplication group with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$.
- For every input group $\boldsymbol{\alpha}$ assigned to a party P_i, this party knows $\boldsymbol{\lambda_\alpha}$, and the parties have $\langle \lambda_{\alpha_1} \rangle, \ldots, \langle \lambda_{\alpha_{m\ell}} \rangle$

Input Gates. For a group of input gates $\boldsymbol{\alpha}$ owned by a party P_i, this party, who knows $\boldsymbol{\lambda_\alpha}$ from the preprocessing, and also knows its input $\boldsymbol{v_\alpha}$, broadcasts $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$.

Addition Gates. For a group of addition gates with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$, the parties locally add $\boldsymbol{\mu_\gamma} \leftarrow \boldsymbol{\mu_\alpha} + \boldsymbol{\mu_\beta}$.

Multiplication Gates. For a group of multiplication gates with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$, the parties proceed as follows:

1. Locally compute

$$[\![\boldsymbol{\mu_\gamma}]\!]_* \leftarrow \boldsymbol{\mu_\alpha} \cdot [\![\boldsymbol{\lambda_\beta}]\!] + \boldsymbol{\mu_\beta} \cdot [\![\boldsymbol{\lambda_\alpha}]\!] + \boldsymbol{\mu_\alpha} \star \boldsymbol{\mu_\beta} + [\![\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}]\!]_*$$

2. Call $\boldsymbol{\mu_\gamma} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\mu_\gamma}]\!]_*)$.

Output Gates. Given a group of output wires $\boldsymbol{\alpha}$, call $\boldsymbol{\lambda_\alpha} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\lambda_\alpha}]\!]_*)$, and return the output $\boldsymbol{v_\alpha} = \boldsymbol{\lambda_\alpha} + \boldsymbol{\mu_\alpha}$.

Abort and restart. If any of the calls above results in abort, a set T of corrupt parties with $|T| > t - 2(\ell - 1)$ is identified. The new set of parties is $\{P_1, \ldots, P_n\} \setminus T$, where $n' = n - |T|$ and $t' = t - |T|$, and they execute the following.

- For every input wire α that belongs to $P_i \in T$, the parties call $\lambda_\alpha \leftarrow \mathsf{reconstruct}(\langle \lambda_\alpha \rangle)$ and set $v_\alpha = \mu_\alpha + \lambda_\alpha$.
- For every $P_i \notin T$, let $\alpha_1, \ldots, \alpha_M$ be the input wires that belongs to P_i. The parties do the following:
  1. P_i inputs λ_{α_j} in $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, obtaining $[\lambda_{\alpha_j}]$, for $j \in [M]$.
  2. P_i samples $r \leftarrow_{\$} \mathbb{F}$ and calls $\langle r \rangle \leftarrow \mathsf{share}(r)$.[a] P_i also inputs r in $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, obtaining $[r]$.
  3. Parties call $c_1, \ldots, c_M \leftarrow \mathsf{ttrand}()$
  4. Parties compute $\langle z \rangle \leftarrow \langle r \rangle + \sum_{j=1}^{M} c_j \cdot \langle \lambda_{\alpha_j} \rangle$ and call $z \leftarrow \mathsf{reconstruct}(\langle z \rangle)$
  5. Use $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ to compute the following function.
     - The inputs are $[\lambda_{\alpha_1}], \ldots, [\lambda_{\alpha_M}], [r]$ as above
     - The function first computes $[z'] \leftarrow [r] + \sum_{j=1}^{M} c_j \cdot [\lambda_{\alpha_j}]$, and outputs 1 if $z' = z$, and 0 otherwise.
  6. If the output is 0, the parties call $\lambda_{\alpha_j} \leftarrow \mathsf{reconstruct}(\langle \lambda_{\alpha_j} \rangle)$ and set $v_{\alpha_j} = \mu_{\alpha_j} + \lambda_{\alpha_j}$, for $j \in [M]$. The party P_i is added to T.
- The parties then use $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ to compute the following function and return its outputs:
  - The (secret) inputs are, for each $P_i \notin T$, $[\lambda_{\alpha_1}], \ldots, [\lambda_{\alpha_M}]$ as above.
  - For every such values, the function first computes $[v_{\alpha_j}] = \mu_{\alpha_j} + [\lambda_{\alpha_j}]$ (recall that μ_{α_j} is public, as it is broadcast in the input phase).
  - Using these inputs, together with the public inputs v_{α_j} for $P_i \in T$, compute the circuit C. These outputs are the outputs of the function.

[a] If this sharing aborts, the parties skip to step 6., since P_i must be corrupted.

We prove the following in Section B.4 in the Supplementary Material.

Theorem 4. *Π_{MPC} UC-realizes $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model, with probability $1 - \mathtt{negl}(\kappa)$.*

Communication complexity. We now calculate the communication cost of Π_{MPC} by calculating the cost of different parts:

1. Input gates: This involves each party broadcasting a batch of inputs per input group that it owns. Across all parties and all input groups possible, the cost of this step is bounded by $\mathsf{P2P}(O(|C|n + n^4)), O(|C|n + n^4) \times \mathsf{BC}(1)$.
2. Addition gates: This step is local so there is no communication cost.
3. Multiplication gates: Let $k = |C|/n^2$ be the total number of groups of multiplication gates in the circuit. For each group, we invoke a single $\mathsf{reconstruct}$ which requires $\mathsf{P2P}(O(n^3)), O(n^3) \times \mathsf{BC}(1)$. Hence, the overall cost of this step is $\mathsf{P2P}(O(|C|n)), O(|C|n) \times \mathsf{BC}(1)$.
4. Output gates: Let $k = |C|/n$ be the total number of groups of output gates in the circuit. For each group, we invoke a single $\mathsf{reconstruct}$ which requires $\mathsf{P2P}(O(n^3)), O(n^3) \times \mathsf{BC}(1)$. Hence, the overall cost of this step is $\mathsf{P2P}(O(|C|n)), O(|C|n) \times \mathsf{BC}(1)$.
5. Abort and restart: Let c_I be the number of input wires. The cost of this step is $\mathsf{P2P}(O(|C|n + c_I n^3)), O(c_I n^3) \times \mathsf{BC}(1)$.

Combining all the costs, we get that the overall communication cost of Π_{MPC} in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model is $\mathsf{P2P}(O(|C|n+c_I n^3+n^4)), O(|C|n+c_I n^3+n^4)\times\mathsf{BC}(1)$. Assuming $C >> c_I \cdot n^2$, we get communication cost of $\mathsf{P2P}(O(|C|n+n^4)), O(|C|n+n^4)\times\mathsf{BC}(1)$.

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.

References

[AAY21] Ittai Abraham, Gilad Asharov, and Avishay Yanai. "Efficient Perfectly Secure Computation with Optimal Resilience". In: *TCC 2021: 19th Theory of Cryptography Conference, Part II*. Ed. by Kobbi Nissim and Brent Waters. Vol. 13043. Lecture Notes in Computer Science. Raleigh, NC, USA: Springer, Heidelberg, Germany, 2021, pp. 66–96. DOI: 10.1007/978-3-030-90453-1_3.

[Abr+23] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. "Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time". In: *Advances in Cryptology – EUROCRYPT 2023, Part II*. Ed. by Carmit Hazay and Martijn Stam. Vol. 14005. Lecture Notes in Computer Science. Lyon, France: Springer, Heidelberg, Germany, 2023, pp. 251–281. DOI: 10.1007/978-3-031-30617-4_9.

[AKP23] Benny Applebaum, Eliran Kachlon, and Arpita Patra. "The Round Complexity of Statistical MPC with Optimal Resiliency". In: *Cryptology ePrint Archive* (2023).

[AL17] Gilad Asharov and Yehuda Lindell. "A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation". In: *Journal of Cryptology* 30.1 (Jan. 2017), pp. 58–151. DOI: 10.1007/s00145-015-9214-4.

[Bea91] Donald Beaver. "Secure Multiparty Protocols and Zero-Knowledge Proof Systems Tolerating a Faulty Minority". In: *Journal of Cryptology* 4.2 (Jan. 1991), pp. 75–122. DOI: 10.1007/BF00196771.

[Bea92] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. "Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2012, pp. 663–680. DOI: 10.1007/978-3-642-32009-5_39.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[BTH06] Zuzana Beerliová-Trubíniová and Martin Hirt. "Efficient Multi-party Computation with Dispute Control". In: *TCC 2006: 3rd Theory of Cryptography Conference*. Ed. by Shai Halevi and Tal Rabin. Vol. 3876. Lecture Notes in Computer Science. New York, NY, USA: Springer, Heidelberg, Germany, 2006, pp. 305–328. DOI: 10.1007/11681878_16.

[BTH08] Zuzana Beerliová-Trubíniová and Martin Hirt. "Perfectly-Secure MPC with Linear Communication Complexity". In: *TCC 2008: 5th Theory of Cryptography Conference*. Ed. by Ran Canetti. Vol. 4948. Lecture Notes in Computer Science. San Francisco, CA, USA: Springer, Heidelberg, Germany, 2008, pp. 213–230. DOI: 10.1007/978-3-540-78524-8_13.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. "Multiparty Unconditionally Secure Protocols (Extended Abstract)". In: *20th Annual ACM Symposium on Theory of Computing*. Chicago, IL, USA: ACM Press, 1988, pp. 11–19. DOI: 10.1145/62212.62214.

[CP17] Ashish Choudhury and Arpita Patra. "An Efficient Framework for Unconditionally Secure Multiparty Computation". In: *IEEE Transactions on Information Theory* 63.1 (2017), pp. 428–468. DOI: 10.1109/TIT.2016.2614685.

[Cra+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. "Efficient Multiparty Computations Secure Against an Adaptive Adversary". In: *Advances in Cryptology – EUROCRYPT'99*. Ed. by Jacques Stern. Vol. 1592. Lecture Notes in Computer Science. Prague, Czech Republic: Springer, Heidelberg, Germany, 1999, pp. 311–326. DOI: 10.1007/3-540-48910-X_22.

[DLN19] Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. "Communication Lower Bounds for Statistically Secure MPC, With or Without Preprocessing". In: *Advances in Cryptology – CRYPTO 2019,*

Part II. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2019, pp. 61–84. DOI: 10.1007/978-3-030-26951-7_3.

[DN07] Ivan Damgård and Jesper Buus Nielsen. "Scalable and Unconditionally Secure Multiparty Computation". In: *Advances in Cryptology – CRYPTO 2007*. Ed. by Alfred Menezes. Vol. 4622. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2007, pp. 572–590. DOI: 10.1007/978-3-540-74143-5_32.

[EF21] Daniel Escudero and Serge Fehr. "On Fully-Secure Honest Majority MPC Without n^2 Round Overhead". In: *Progress in Cryptology - LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America*. Ed. by Patrick Longa and Carla Ràfols. Vol. 12912. Lecture Notes in Computer Science. Bogotá, Colombia: Springer, Heidelberg, Germany, 2021, pp. 47–66. DOI: 10.1007/978-3-031-44469-2_3.

[Esc+22] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "TurboPack: Honest Majority MPC with Constant Online Communication". In: *ACM CCS 2022: 29th Conference on Computer and Communications Security*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. Los Angeles, CA, USA: ACM Press, 2022, pp. 951–964. DOI: 10.1145/3548606.3560633.

[FY92] Matthew K. Franklin and Moti Yung. "Communication Complexity of Secure Computation (Extended Abstract)". In: *24th Annual ACM Symposium on Theory of Computing*. Victoria, BC, Canada: ACM Press, 1992, pp. 699–710. DOI: 10.1145/129712.129780.

[GLS19] Vipul Goyal, Yanyi Liu, and Yifan Song. "Communication-Efficient Unconditional MPC with Guaranteed Output Delivery". In: *Advances in Cryptology – CRYPTO 2019, Part II*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11693. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2019, pp. 85–114. DOI: 10.1007/978-3-030-26951-7_4.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. "Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography". In: *17th ACM Symposium Annual on Principles of Distributed Computing*. Ed. by Brian A. Coan and Yehuda Afek. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998, pp. 101–111. DOI: 10.1145/277697.277716.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. "Guaranteed Output Delivery Comes Free in Honest Majority MPC". In: *Advances in Cryptology – CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2020, pp. 618–646. DOI: 10.1007/978-3-030-56880-1_22.

[HMP00] Martin Hirt, Ueli M. Maurer, and Bartosz Przydatek. "Efficient Secure Multi-party Computation". In: *Advances in Cryptology – ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. Vol. 1976. Lecture Notes in Computer Science. Kyoto, Japan: Springer, Heidelberg, Germany, 2000, pp. 143–161. DOI: 10.1007/3-540-44448-3_12.

[IK00] Yuval Ishai and Eyal Kushilevitz. "Randomizing polynomials: A new representation with applications to round-efficient secure computation". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. IEEE. 2000, pp. 294–304.

[IK02] Yuval Ishai and Eyal Kushilevitz. "Perfect constant-round secure computation via perfect randomizing polynomials". In: *Automata, Languages and Programming: 29th International Colloquium, ICALP 2002 Málaga, Spain, July 8–13, 2002 Proceedings 29*. Springer. 2002, pp. 244–256.

[Ish+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. "Secure Protocol Transformations". In: *Advances in Cryptology – CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, 2016, pp. 430–458. DOI: 10.1007/978-3-662-53008-5_15.

[PR10] Arpita Patra and C. Pandu Rangan. *Communication and Round Efficient Information Checking Protocol*. 2010. arXiv: 1004.3504 [cs.CR].

[RB89] Tal Rabin and Michael Ben-Or. "Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract)". In: *21st Annual ACM Symposium on Theory of Computing*. Seattle, WA, USA: ACM Press, 1989, pp. 73–85. DOI: 10.1145/73007.73014.

[Sha79] Adi Shamir. "How to Share a Secret". In: *Communications of the Association for Computing Machinery* 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

Supplementary Material

A Honest-Majority Fully-Secure MPC

Rabin and Ben-Or [RB89] are the first in showing that, as a feasibility result, G.O.D. is possible with polynomial communication if $n = 2t + 1$, assuming a broadcast channel and allowing for statistical (also known as unconditional) security. In that work, the authors introduce a verifiable secret-sharing (VSS) tool that enhances plain Shamir secret-sharing [Sha79] so that cheating parties who modify their shares can be identified. [RB89] follows the "BGW paradigm" from [BGW88], in which the parties have degree-t VSS shares of each secret, and for multiplications they locally multiply these shares, obtaining degree-$2t$ shares, which are reshared using degree-t towards the other parties. Some corrupt parties may misbehave in this step, which can be detected thanks to the use of VSS in conjunction to certain "zero-knowledge" subprotocols for verifying products. However, since the product has degree $2t$ and there are $2t + 1$ parties, *all* of the parties' contributions are needed in order to make progress, so disqualifying a party prevents the parties from continuing. To address this, before sharing the product of their shares, each party also distributes VSS-shares of each of their individual shares (proving in "zero-knowledge" they do so correctly), which enables the parties to make public the shares of parties that are disqualified, allowing them to continue with the computation. Overall, this only adds a tiny overhead to each multiplication layer, keeping the total number of rounds bellow $O(\mathsf{depth}(C))$, independent of n. Later, the work of [Bea91] optimized the "zero-knowledge" component in [RB89], resulting in an asymptotically similar result.

Cramer et al. [Cra+99] present a protocol with complexity $O(|C|n^5 + n^4) + O(|C|n^5)\mathsf{BC}$, which is at least a factor of $\kappa^2 \cdot n$ better than [Bea91; RB89], where κ is the statistical security parameter. In addition, [RB89] requires communication for each addition gate (since their VSS is *not linear*), while [Cra+99] avoids such communication by designing a linearly homomorphic VSS. For multiplications, [Cra+99] does not use the BGW template, but instead uses the simpler and more efficient GRR [GRR98]. GRR is particularly suitable for $t < n/3$, or for $t < n/2$ with computational assumptions (they use homomorphic commitments), but [Cra+99] notes that it can be made to work with statistical security using their VSS, as long as a given party can VSS-share a product of two of its shares while proving it did so correctly. The authors of [Cra+99] present a protocol for this, but crucially, in contrast to [RB89], each party's share is *not* VSS-shared across the other parties as part of the protocol (which is part of the optimizations in [Cra+99]). This means that, in case a cheater is disqualified, the parties cannot simply reconstruct its share to be able to finish a given multiplication without its help/ Instead, the parties must reconstruct this corrupt party's inputs (which are actually VSS-shared), and redo the computation up to this point. As a result, the computation may end up being redone $t = O(n)$ times, so the number of

rounds becomes $O(\mathsf{depth}(C) \cdot n)$ in the worst case (this factor of n also affects communication).

With the goal of improving communication complexity, Beerliová-Trubíniová and Hirt [BTH06] introduce a novel technique called *dispute-control*, resulting in protocols with $O(|C|n^2 + n^5) + O(n^3)\mathsf{BC}$ communication. These massive improvements w.r.t. [Cra+99] are due to several aspects. First, [BTH06] deviates from the BGW/GRR paradigm and adopts multiplication triples [Bea92] for the first time in the G.O.D. honest majority literature, which results in an online phase that only requires degree-t reconstructions, which is arguably simpler than the "zero-knowledge"-type techniques involved in prior works. In particular, the VSS needed in [BTH06] can be made more lightweight; the main bottlenecks are pushed to an offline phase, which is in charge of, among other things, producing multiplication triples. Second, [BTH06] does not aim at fully disqualifying parties—*i.e.* identifying cheaters—but instead settles with identifying *disputed parties*, which are pairs of parties with at least one of them being corrupt, with no certainty on who is the cheater. Dispute-control is then the technique that enables the parties to recover using this seemingly weaker notion: the protocol is rerun while ensuring that either it succeeds, or a *new* disputed pair is identified; since there are at most $O(n^2)$ possible pairs of parties, this process is repeated at most $O(n^2)$ times, point in which the protocol is guaranteed to succeed. Naively this adds a factor of n^2 to both communication and round complexity, but the authors make use of an idea stemming from [HMP00], which consists of splitting the circuit into "segments", performing the recovery steps at the end of each segment instead of at the end of the whole protocol. This way, if cheating is detected, only the current segment must be repeated, and by choosing the number of multiplications per segment appropriately one can ensure that the potential $O(n^2)$ repetitions do not affect communication asymptotically. Crucially for our work, however, these repetitions *do affect* the number of rounds by adding $O(n^2)$ extra rounds, resulting on $O(\mathsf{depth}(C) + n^2)$ in the worst case.

While [BTH06] achieved quadratic communication (and the authors even conjectured this was optimal), the groundbreaking work by Ben-Sasson, Fehr, and Ostrovsky [BFO12] showed that *linear communication* was possible, obtaining acomplexity of $O(|C|n + \mathsf{depth}(C)n^2 + n^7) + O(n^3)\mathsf{BC}$, which is $O(|C|n + n^7)$ if $\mathsf{depth}(C) = O(|C|/n)$, or equivalently, the (average) width is $|C|/\mathsf{depth}(C) = \Omega(n)$. The current state-of-the-art is [GSZ20], which improves the concrete constants in the O-notation from [BFO12], on top of removing the $\Omega(n)$ width requirement, achieving a communication of $5.5|C|n$ in the optimistic case, and $7.5|C|n$ in the worst case. This work, however, still uses dispute control and hence suffers from the extra n^2 overhead in terms of rounds.

A recent work that deviates from the identify-then-rerun paradigm and still achieves linear communication is [EF21], which proposes a protocol *in the preprocessing model* with online linear communication $O(|C|n + \mathsf{depth}(C)n^3) + 0 \cdot \mathsf{BC}$, which is $O(|C|n)$ if the width is $\Omega(n^2)$. The protocol has the advantage of not requiring a broadcast channel in every round. Unfortunately, instantiating the preprocessing with a protocol that does not add $\Omega(n)$ rounds, like [RB89],

would lead to a very inefficient (yet polynomial) communication complexity in the online phase.

B Several Missing Proofs

B.1 Proofs from Section 3

Theorem 5 (Theorem 1, restated). *$\Pi_{\mathsf{batch\text{-}IC}}$ UC-realizes $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for any $\ell = \mathtt{poly}(\kappa)$ with probability $1 - \mathtt{negl}(\kappa)$.*

Proof. We first provide a simulator $\mathcal{S}$:

1. For initialization:
   (a) If the dealer D is honest, then for every corrupted party P_i, $\mathcal{S}$ samples random α_i and sends it to P_i; otherwise, $\mathcal{S}$ receives from the adversary α_i for each honest party P_i.
   (b) Then for each $\tau \in [T]$:
      i. If D is honest and INT is corrupted, $\mathcal{S}$ samples random degree-$(t+2\ell-2)$ polynomials $o_1(x), o_2(x)$ such that $o_1(-j+1) = o_2(-j+1) = 0$ for $j \in [\ell]$, then sends $(o_1(x), o_2(x))$ to the adversary, as well as $o_{1,i} \leftarrow o_1(\alpha_i), o_{2,i} \leftarrow o_2(\alpha_i)$ for all other corrupted parties P_i.
      ii. If both the dealer D and INT are honest, then for each corrupted party P_i, $\mathcal{S}$ samples random $o_{1,i}$ and $o_{2,i}$ and sends them to the adversary.
      iii. If D is corrupted and INT is honest, then $\mathcal{S}$ receives from the adversary $o_1(x)$ and $o_2(x)$.
      iv. If both D and INT are corrupt, then $\mathcal{S}$ receives from the adversary $o_{1,i}$ and $o_{2,i}$ for each honest party P_i.
   (c) Next:
      i. If D is honest and INT is corrupted, then if INT broadcasts $\mathsf{Corr} = 1$, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, $\mathcal{S}$ receives $(\beta, o(x))$ from the adversary.
      ii. If both the dealer D and INT are honest, then $\mathcal{S}$ samples random β and random degree-$(t + 2\ell - 2)$ polynomial $o(x)$ such that $o(\alpha_i) = \beta \cdot o_{1,i} - o_{2,i}$ for all corrupted parties P_i and $o(-j + 1) = 0$ for all $j \in [\ell]$, then broadcasts $(\beta, o(x))$.
      iii. If D is corrupted and INT is honest, then if $o_1(x), o_2(x)$ are not degree-$(t+2\ell-2)$ polynomials such that $o_1(-j+1) = o_2(-j+1) = 0$ for $j \in [\ell]$, then $\mathcal{S}$ broadcasts $\mathsf{Corr} \leftarrow 1$ and sends Corr to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, $\mathcal{S}$ samples random β and broadcasts $(\beta, o(x))$, where $o(x) \leftarrow \beta \cdot o_1(x) - o_2(x)$.
      iv. If both D and INT are corrupt, if INT broadcasts $\mathsf{Corr} = 1$, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, $\mathcal{S}$ receives $(\beta, o(x))$ from the adversary.
   (d) Then:

i. If D is honest and INT is corrupted, $\mathcal{S}$ checks if $o(x) = \beta \cdot o_1(x) - o_2(x)$, and if so, stores $o_\tau(x) \leftarrow o_1(x)$, otherwise, broadcasts $\mathsf{Corr} \leftarrow 1$ and sends Corr to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.
ii. If both the dealer D and INT are honest, then $\mathcal{S}$ stores $o_{\tau,i} \leftarrow o_{1,i}$ for each corrupt party.
iii. If D is corrupted and INT is honest, then if D broadcasts $\mathsf{Corr} = 1$, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, $\mathcal{S}$ stores $o_\tau(x) \leftarrow o_1(x)$.
iv. If both D and INT are corrupt, if INT broadcasts $\mathsf{Corr} = 1$, $\mathcal{S}$ forwards it to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, if $o(\alpha_i) \neq \beta \cdot o_{1,i} - o_{2,i}$, $\mathcal{S}$ sets $\mathsf{dealerbad}_i \leftarrow 1$,

2. For signing, if $\mathcal{S}$ input $\mathsf{Corr} = 1$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ during initialization, then for honest dealer, $\mathcal{S}$ just broadcasts $\boldsymbol{s}$; otherwise, we simulate as follows:
   (a) First:
      i. If the dealer D is honest and intermediary INT is corrupted, $\mathcal{S}$ first receives $\boldsymbol{s}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, then samples random degree-$(t + \ell - 1)$ polynomial $f(x)$ such that $f(-j+1) = s^j$ for all $j \in [\ell]$ and random degree-$(t + \ell - 1)$ polynomial $r(x)$, then sends $f(x)$ and $r(x)$ to the adversary, as well as $v_{\mathsf{sid},i} \leftarrow f(\alpha_i)$ and $r_{\mathsf{sid},i} \leftarrow r(\alpha_i)$ for all other corrupted parties P_i.
      ii. If both the dealer D and intermediary INT are honest, then for each corrupted party P_i, $\mathcal{S}$ samples random $v_{\mathsf{sid},i}$ and $r_{\mathsf{sid},i}$ and sends them to the adversary.
      iii. If the dealer D is corrupted and the intermediary INT is honest, then $\mathcal{S}$ receives from the adversary $f(x)$ and $r(x)$, sets $s^j \leftarrow f(-j+1)$ for $j \in [\ell]$, and sends $(\mathsf{sign}, (s^1, \ldots, s^\ell), \mathsf{sid})$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.
      iv. Finally, if both the dealer D and intermediary INT are corrupted, then $\mathcal{S}$ receives from the adversary $v_{\mathsf{sid},i}$ and $r_{\mathsf{sid},i}$ for all (at least $t+1$) honest parties P_i, and sends $(\mathsf{sign}, (0, \ldots, 0), \mathsf{sid})$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.
   (b) Next:
      i. If the dealer D is honest and intermediary INT is corrupted, then $\mathcal{S}$ receives $(\beta, b(x))$ from the adversary.
      ii. If both the dealer D and intermediary INT are honest, then $\mathcal{S}$ samples random β and random degree-$(t + \ell - 1)$ polynomial $b(x)$ such that $b(\alpha_i) = \beta \cdot v_{\mathsf{sid},i} + r_{\mathsf{sid},i}$ for all corrupted parties P_i, then broadcasts $(\beta, b(x))$.
      iii. If the dealer D is corrupted and the intermediary INT is honest, then $\mathcal{S}$ samples random β and broadcasts $(\beta, b(x))$, where $b(x) \leftarrow \beta \cdot f(x) + r(x))$.
      iv. Finally, if both the dealer D and intermediary INT are corrupted, then $\mathcal{S}$ receives $(\beta, b(x))$ from the adversary.
   (c) Finally:
      i. If D is honest and INT is corrupted, then if INT was corrupted after the broadcast of $(\beta, b(x))$, $\mathcal{S}$ just sets $h_{\mathsf{sid}}(x) \leftarrow f(x)$; otherwise, $\mathcal{S}$ checks that $b(x) = \beta \cdot f(x) + r(x)$, and if so, sets $h_{\mathsf{sid}}(x) \leftarrow f(x)$, otherwise broadcasts $\boldsymbol{s}$ and sets $h_{\mathsf{sid}}(x) \leftarrow g(x)$, where $g(x)$ is the degree $\ell - 1$ polynomial such that $g(-j+1) = s^j$ for $j \in [\ell]$.

ii. If both the dealer D and intermediary INT are honest, then $\mathcal{S}$ does nothing.

iii. If D is corrupted and INT is honest, then if D broadcasts $\boldsymbol{s}'$, $\mathcal{S}$ sets $h_{\mathsf{sid}}(x) \leftarrow g(x)$, where $g(x)$ is the degree-ℓ polynomial such that $g(-j+1) = (s')^j$ for $j \in [\ell]$; else it sets $h_{\mathsf{sid}}(x) \leftarrow f(x)$ and $s_j \leftarrow f(-j+1)$, for $j \in [\ell]$. Finally, when $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ asks, $\mathcal{S}$ sends $(s^1, \ldots, s^\ell)$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.

iv. Finally, if both the dealer D and intermediary INT are corrupted, if D broadcasts $\boldsymbol{s}$, $\mathcal{S}$ sets $h_{\mathsf{sid}}(x) \leftarrow g(x)$, where $g(x)$ is the degree-ℓ polynomial such that $g(-j+1) = s^j$ for $j \in [\ell]$. Otherwise, $\mathcal{S}$ sets $h_{\mathsf{sid}}(x) \leftarrow 0$ and for each honest party P_i such that $b(\alpha_i) \neq \beta \cdot v_{\mathsf{sid},i} + r_{\mathsf{sid},i}$, $\mathcal{S}$ sets $\mathsf{dealerbad}_i \leftarrow 1$. Then, when $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ asks, $\mathcal{S}$ sends $(h(-\ell+1), \ldots, h(0))$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.

3. For revealing, we simulate as follows:
   (a) First, $\mathcal{S}$ first receives $\boldsymbol{s}_{\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.
   (b) If INT is corrupted, then $\mathcal{S}$ receives $h(x) = \sigma_{\mathsf{sid}} + o_\tau(x)$ from the adversary. Then:
      i. If D is honest, if $h(x) = h_{\mathsf{sid}}(x) + o_\tau(x)$ stored by $\mathcal{S}$, $\mathcal{S}$ broadcasts accept on behalf of all honest parties P_i and then sends continue to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Otherwise, $\mathcal{S}$ broadcasts reject on behalf of all honest parties P_i and sends reject to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$.
      ii. If D is corrupted, then $\mathcal{S}$ broadcasts accept on behalf of all honest parties P_i if $v_{\mathsf{sid},i} + o_{\tau,i} = h(\alpha_i)$ or $\mathsf{dealerbad}_i = 1$. Then if at least $t+1$ parties (honest or corrupted) broadcasted accept, then $\mathcal{S}$ sends $(h(-\ell+1), \ldots, h(0))$ to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; otherwise, it sends reject.
   (c) If INT is honest:
      i. If D is honest, $\mathcal{S}$ first receives $\boldsymbol{s}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, then samples degree-$(t+2\ell-2)$ polynomial $h_{\mathsf{sid}}(x)$ such that $h_{\mathsf{sid}}(-j+1) = s_j$ for $j \in [\ell]$ and $h_{\mathsf{sid}}(\alpha_i) = v_{\mathsf{sid},i} + o_{\tau,i}$ for all corrupted parties P_i. (If D is corrupted, $h_{\mathsf{sid}}(x), o_\tau(x)$ should already be stored)
      ii. Then $\mathcal{S}$ broadcasts $h_{\mathsf{sid}}(x)$ on behalf of INT and then accept on behalf of all honest parties P_i.
4. For adding, there is no communication to be simulated, but:
   (a) If both D and INT are honest, then $\mathcal{S}$ sets $v_{\mathsf{sid}_3,i} \leftarrow v_{\mathsf{sid}_1,i} + v_{\mathsf{sid}_2,i}$ for each corrupted party P_i.
   (b) Otherwise, $\mathcal{S}$ sets $h_{\mathsf{sid}_3}(x) \leftarrow h_{\mathsf{sid}_1}(x) + h_{\mathsf{sid}_2}(x)$.
5. For multiplication by public vector, there is also no communication to be simulated, but letting $u(x)$ be the degree-$(\ell-1)$ polynomial such that $u(-j+1) = u^j$ for $j \in [\ell]$:
   (a) If both D and INT are honest, then $\mathcal{S}$ sets $v_{\mathsf{sid}',i} \leftarrow v_{\mathsf{sid},i} \cdot u(\alpha_i)$ for each corrupted party P_i.
   (b) Otherwise, $\mathcal{S}$ sets $h_{\mathsf{sid}'}(x) \leftarrow h_{\mathsf{sid}}(x) \cdot u(x)$.
6. For a corruption of the dealer D, to build their state, $\mathcal{S}$ first samples random α_i for all honest parties (in addition to those already sampled for corrupted parties), then if INT was not already corrupted, $\mathcal{S}$ receives $\boldsymbol{s}_{\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for each originally signed sid and interpolates degree-$(t+\ell-1)$ polynomial

$h_{\mathsf{sid}}(x)$ such that $h_{\mathsf{sid}}(-j+1) = s_j$ for $j \in [\ell]$ and $h_{\mathsf{sid}}(\alpha_i) = v_{\mathsf{sid},i}$ for all corrupted parties P_i.

7. For corruption of the intermediary INT, to build their state, if D was not already corrupted, $\mathcal{S}$ receives $\boldsymbol{s}_{\mathsf{sid}}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for each originally signed sid and interpolates degree-$(t+\ell-1)$ polynomial $h_{\mathsf{sid}}(x)$ such that $h_{\mathsf{sid}}(-j+1) = s_j$ for $j \in [\ell]$ and $h_{\mathsf{sid}}(\alpha_i) = v_{\mathsf{sid},i}$ for all corrupted parties P_i.
8. For a corruption of some party P_i, to build their state, $\mathcal{S}$ samples random α_i, and if D and INT are honest, $\mathcal{S}$ samples random $v_{\mathsf{sid},i}$, otherwise, $\mathcal{S}$ sets $v_{\mathsf{sid},i} \leftarrow h_{\mathsf{sid}}(\alpha_i)$.

Now we show that the real world is distributed identically to the ideal world, except with probability $\mathtt{negl}(\kappa)$. For initialization, it is clear that the real and ideal worlds are distributed identically for distributing the α_i. Then, for $\tau \in [T]$: In the first step, if the dealer is honest it is clear that the two worlds are identical: if the INT is corrupted, then the polynomials $o_1(x)$ and $o_2(x)$ that $\mathcal{S}$ sends to INT are distributed the same way; if INT is honest then the t points on degree-$(t+2\ell-2)$ polynomials $o_2(x)$ and $o_2(x)$ that the corrupted parties receive are distributed randomly and independently of the $\ell-1$ evaluations set to 0. In the second step, if INT is honest and D is corrupted, then $\mathcal{S}$ behaves exactly as INT in the real world; if both INT and D are honest, then $\mathcal{S}$'s broadcast of random degree-$(t+2\ell-2)$ polynomial $o(x)$ consistent with the corrupted parties' points and $o(-j+1) = 0$ for $j \in [\ell]$ is identical to the real world, since the probability that any other point $o(i)$ is equal to some value is exactly equal to the probability that $o_2(i) = \beta \cdot o_1(i) - o(i)$, which is random, constrained to $o_2(x)$ being degree-$(t+2\ell-2)$ (Note that this preserves the randomness in $o_1(x)$). In the third step, if D is honest and INT was corrupted and broadcasted $o(x) \neq \beta \cdot o_1(x) - o_2(x)$, then it is clear that in both the real world and ideal world, D broadcasts $\mathsf{Corr} = 1$; otherwise, in both worlds, D does nothing and the honest parties do not set $\mathsf{dealerbad} \leftarrow 1$. The same holds if both D and INT are honest (i.e., the honest parties do nothing). If D is corrupted and INT is honest, it is clear that the honest parties set $\mathsf{dealerbad}_\tau \leftarrow 1$ if D did not broadcast $\mathsf{Corr} = 1$ properly. If both D and INT are corrupted, $\mathcal{S}$ sets $\mathsf{dealerbad}_i$ for each honest party P_i in the same way as in the real world.

For signing, in the first step, if the dealer is honest it is clear that the two worlds are identical: if the INT is corrupted, then the polynomials $f(x)$ and $r(x)$ that $\mathcal{S}$ sends to INT are distributed the same way, since $\mathcal{S}$ gets $\boldsymbol{s}$ from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$; if INT is honest then the t points on degree-$(t+\ell-1)$ polynomials $f(x)$ and $r(x)$ that the corrupted parties receive are distributed randomly and independently of $\boldsymbol{s}$. If the dealer is corrupted and INT is honest, then $\mathcal{S}$ can clearly extract some $\boldsymbol{s}$ from $f(x)$ received to input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. In the second step, if INT is honest and D is corrupted, then $\mathcal{S}$ behaves exactly as INT in the real world; if both INT and D are honest, then $\mathcal{S}$'s broadcast of random degree-$(t+\ell-1)$ polynomial $b(x)$ consistent with the corrupted parties' points is identical to the real world, since the probability that any other point $b(i)$ is equal to some value is exactly equal to the probability that $r(i) = b(i) - \beta \cdot f(i)$, which is random, constrained to $r(x)$ being degree-$(t+\ell-1)$. In the third step, if D is honest and INT was

corrupted and broadcasted $b(x) \neq \beta \cdot f(x) + r(x)$, then it is clear that in both the real world and ideal world, D broadcasts $\boldsymbol{s}$ and the honest parties will set their $v_{\mathsf{sid},i}$ according to $h_{\mathsf{sid}}(x)$; otherwise, in both worlds, D does nothing and the honest parties keep their $v_{\mathsf{sid},i}$ according to $h_{\mathsf{sid}}(x)$. The same holds if both D and INT are honest (i.e., the honest parties do nothing). If D is corrupted and INT is honest, it is clear that $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ the correct signed $\boldsymbol{s}$ and the honest parties set their $v_{\mathsf{sid},i}$ according to $h_{\mathsf{sid}}(x)$. If both D and INT are corrupted, $\mathcal{S}$ sets $\mathsf{dealerbad}_i$ for each honest party P_i in the same way as in the real world (Note that in this case, it does not matter what $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ for $\boldsymbol{s}$, nor what $h_{\mathsf{sid}}(x)$ is; it will be corrected eventually during the reveal phase).

For adding and multiplying, if both D and INT are honest, it is clear that the honest parties set their new v_i in the same way for both worlds. Otherwise (except if both D and INT are corrupted, which will be handled in the reveal phase), it is clear that the honest parties in the real world will set their new v_i according to the new $h(x)$ computed by $\mathcal{S}$ in the ideal world.

For the reveal phase, if INT is corrupted: then if D is also corrupted, it is clear that $\mathcal{S}$ behaves exactly as in the real world on behalf of the honest parties and if at least $t+1$ parties accept, $\mathcal{S}$ is able to extract $\boldsymbol{s}$ to input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ from $h(x)$, which the honest parties output in both worlds; If D is honest, then in the real world, the adversary does not know the random α_i of honest parties and so by the Schwartz-Zippel Lemma, the probability that $h(\alpha_i) = v_i = h_{\mathsf{sid}}(\alpha_i) + o_\tau(\alpha_i)$ for any honest P_i, where $h(x)$ is broadcast by INT and $h_{\mathsf{sid}}(x), o_\tau(x)$ are the polynomials which the honest parties' shares are based on is negligible in κ (if $|\mathbb{F}|$ is super-polynomial in κ). So, with all but negligible probability, if $h(x) \neq h_{\mathsf{sid}}(x) + o_\tau(x)$, then in both worlds, the parties reject; otherwise they accept and output the same $\boldsymbol{s}$ (in part because $o_\tau(-j+1) = 0$ for $j \in [\ell]$). If INT is honest: if D is also honest, it is clear that $\mathcal{S}$ samples $h_{\mathsf{sid}}(x)$ based on the corrupted parties' t points $v_{\mathsf{sid},i} + o_{\tau,i}$ and the ℓ points of $\boldsymbol{s}$ received from $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ that is distributed identically to the real world. This is because in the real world, the probability that any other point $h_{\mathsf{sid}}(i)$ is equal to some value is exactly equal to the probability that $o_\tau(i) = h_{\mathsf{sid}}(i) - v_{\mathsf{sid},i}$, which is random. If D is corrupted, then it is clear that $\mathcal{S}$ sets $h_{\mathsf{sid}}(x) \leftarrow h_{\mathsf{sid}}(x) + o_\tau(x)$ as INT would in the real world. Furthermore (if D is corrupted), since INT chooses β independently of $f(x)$ and $r(x)$, for any honest party P_i, if D gives it $v_{\mathsf{sid},i} \neq f(\alpha_i)$, then the probability that $\beta \cdot v_{\mathsf{sid},\alpha_i} + r_{\mathsf{sid},i} = \beta \cdot f(\alpha_i) + r(\alpha_i)$ is equal to the probability that $\beta = (r(\alpha_i) - r_{\mathsf{sid},i})/(v_{\mathsf{sid},\alpha_i} - f(\alpha_i))$, which is negligible in κ (if $|\mathbb{F}|$ is super-polynomial in κ). A similar thing can be said for the $\beta, o_\tau(x), o_{\tau,i}$ from the initialization phase. Thus, if for $h_{\mathsf{sid}}(x)$ broadcasted by INT, $h_{\mathsf{sid}}(\alpha_i) \neq v_{\mathsf{sid},i} + o_{\tau,i}$, then with all-but-negligible probability, for some other sid' or τ, P_i would have set $\mathsf{dealerbad} \leftarrow 1$. Therefore, in the real world, when INT is honest, the honest parties accept with all-but-negligible probability; in the ideal world in this case, the honest parties always accept. The parties in the two worlds output the same $\boldsymbol{s}$, since $\mathcal{S}$ extracts $\boldsymbol{s}$ from the originally input signed values (which $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ uses to compute the final vector for sid), and the honest parties in the real world do the same from $h_{\mathsf{sid}}(x)$ in the protocol, where

$o_\tau(x)$ must satisfy $o_\tau(-j+1) = 0$ for $j \in [\ell]$ since otherwise, honest INT would have broadcasted $\mathsf{Corr} = 1$.

For similar reasons as above, corruptions are simulated perfectly. Therefore, the ideal and real worlds are distributed identically except with probability $\mathtt{negl}(\kappa)$. □

B.2 Proofs from Section 4

Lemma 17 (Lemma 3, restated). *If the dealer P_i is honest in $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, then all honest parties finish $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$ without aborting.*

Proof. This follows easily from the properties of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Indeed, an honest dealer P_i will only give the other parties P_j points $z_\eta^{j1}, \ldots, z_\eta^{jn}$ and $z_\eta^{1j}, \ldots, z_\eta^{nj}$ consistent with a degree-(d_x, t) bivariate polynomial for $\eta \in [m]$, and input vectors of those points to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. Therefore, other honest parties will never reveal their points in step 3 and corrupt parties will only be able to reveal those points that P_i input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, which will always be consistent. Thus, honest parties never abort from step 3. Also in steps 6 and 7 parties will only be able to reveal those points that P_i input to $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, which will always be consistent with each other. Thus, honest parties never abort from reveals in step 6 and step 7. This concludes the proof. □

Lemma 18 (Lemma 7, restated). *If $\pi_{\mathsf{Packed\text{-}DSS\text{-}Coins}}$ does not abort with some set T of corrupted parties, then it outputs $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m$ that are random and unknown to the adversary.*

Proof. For each $i \in [n]$, let $F_{i,\eta}(x, y)$, $\eta \in [m]$, be those bivariate polynomials defined by the sharing $[\![\boldsymbol{\beta}_i]\!]$, according to Lemma 4. Also, let $F_\eta(x, y) \leftarrow \sum_{i=1}^{n} F_{i,\eta}(x, y)$, $\eta \in [m]$, be the resulting bivariate polynomials defined by $[\![\boldsymbol{\beta}]\!]$, that are interpolated according to Corollary 2. For any $j \in \mathsf{Hon}$, we have that

$$F_{j,\eta}(x, y) = F_\eta(x, y) - \sum_{i \in [n] \setminus \{j\}} F_{i,\eta}(x, y).$$

Therefore, the probability that $F_\eta(x, y)$ is some given polynomial is equal to the probability that $F_{j,\eta}(x, y)$ is some given polynomial. For this P_j, for $\eta \in [m]$, we have by Lemma 2 that the corrupted parties' points $\{(F_{j,\eta}(k, i), F_{j,\eta}(i, k)\}_{k \in \mathsf{Corr}, i \in [n]}$ are consistent with any other $F'_{j,\eta}(x, y)$ chosen in the same fashion (with other random $\boldsymbol{\beta}'_{j,1}, \ldots, \boldsymbol{\beta}'_{j,m}$). Therefore $F_{j,\eta}(x, y)$ is random to the adversary and thus so are $\boldsymbol{\beta}'_{j,1}, \ldots, \boldsymbol{\beta}'_{j,m}$, and $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m$. □

Lemma 19 (Lemma 8, restated). *If P_i inputs sharings $([\![\boldsymbol{s}_1]\!]_*, [\![\boldsymbol{s}_2]\!]_*) \neq ([\![\mathbf{0}_1]\!]_*, [\![\mathbf{0}_2]\!]_*)$ to $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, then with probability at least $1 - \mathtt{negl}(\kappa)$, all honest parties output either (abort, T), for $|T| > t - 2\ell + 2$ where each $j \in T$ corresponds to a corrupt P_j, or abort. Otherwise, all honest parties output either (abort, T), for $|T| > t - 2\ell + 2$ where each $j \in T$ corresponds to a corrupt P_j, or $[\![\mathbf{0}_1]\!]_*$.*

Proof. We know from Lemma 6 that if the honest parties output (abort, T) such that $|T| > t - 2\ell + 2$, then for each $j \in T$, $P_j \in \mathsf{Corr}$. Otherwise, also from Lemma 6, we know that the opened $\boldsymbol{e}_\eta = \beta \cdot \boldsymbol{s}_{1,\eta} - \boldsymbol{s}_{2,\eta}$. Thus, if $\boldsymbol{s}_{1,\eta} = \boldsymbol{s}_{2,\eta} = \boldsymbol{0}$, then $\boldsymbol{e}_\eta = \boldsymbol{0}$ for $\eta \in [m]$, and so the honest parties will output $[\![\boldsymbol{0}_1]\!]_{t+2\ell-2,t}$.

Otherwise, first note that if $\boldsymbol{e}_\eta = \boldsymbol{0} = \boldsymbol{s}_{1,\eta}$, then it must be that $\boldsymbol{s}_{2,\eta} = \boldsymbol{0}$. If $\boldsymbol{s}_{1,\eta} \neq \boldsymbol{0}$, then $\boldsymbol{e}_\eta = \boldsymbol{0}$ if and only if $s^l_{2,\eta}/s^l_{1,\eta} = \beta$ for every $l \in [\ell]$. Since β is chosen randomly after the runs of $\pi_{\mathsf{Packed\text{-}DSS\text{-}Share}}$, in which $\boldsymbol{s}_{1,\eta}, \boldsymbol{s}_{2,\eta}$ are defined according to Lemma 4, the probability that $\boldsymbol{s}_{1,\eta} \neq \boldsymbol{0}$ and $\boldsymbol{s}_{2,\eta}$ are chosen such that $s^l_{2,\eta}/s^l_{1,\eta} = \beta$ for any $l \in [\ell]$ is negligible in κ. □

Lemma 20 (Lemma 9, restated). *$\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ run on an honest party P_i's sharings does not give the adversary any more information on $[\![\boldsymbol{0}_1]\!]_*$.*

Proof. By Lemma 2, both sharings $[\![\boldsymbol{0}_1]\!]_*$ and $[\![\boldsymbol{0}_2]\!]_*$ could have been distributed using, for $\eta \in [m]$, any random degree-$(t+\ell-1, t)$ bivariate polynomials $F_{1,\eta}(x, y)$ and $F_{2,\eta}(x, y)$ consistent with the corrupt parties' shares and $F_{1,\eta}(-l + 1, 0) = F_{2,\eta}(-l + 1, 0) = 0$ for every $l \in [\ell]$. Since the adversary receives $F_\eta(x, y) \leftarrow \beta \cdot F_{1,\eta}(x, y) + F_{2,\eta}(x, y)$, the probability that $F_{1,\eta}(x, y)$ is any given degree-$(t + \ell - 1, t)$ bivariate polynomial consistent with the corrupt parties' shares and $F_{1,\eta}(-l + 1, 0) = 0$ for every $l \in [\ell]$, is equal to the probability that $F_{2,\eta}(x, y) = F_\eta(x, y) - \beta \cdot F_{1,\eta}(x, y)$, which is the same distribution as before $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ for $F_{2,\eta}(x, y)$. □

Lemma 21 (Lemma 10, restated). *For each $i \in [n-t]$, the output $[\![\boldsymbol{0}_1]\!]_*, \ldots, [\![\boldsymbol{0}_{n-t}]\!]_*$ are distributed randomly given the corrupted parties' shares.*

Proof. Let $\{F_{i,\eta}(x, y)\}_{\eta\in[n]}$ be the degree-$(t + 2\ell - 2, t)$ bivariate polynomials defined by the honest parties' shares of $[\![\boldsymbol{0}_{i,1}]\!]_*$ according to Lemma 4. For each $j \in [n]$, let $G_{j,\eta}(x, y)$ for $\eta \in [m]$ be those bivariate polynomials defined by the honest parties' shares of $[\![\boldsymbol{0}_j]\!]_*$, according to Lemma 5. Since $\boldsymbol{M}$ is super-invertible, we have that $\boldsymbol{M}^{\mathsf{Hon}}$ is invertible. Thus, we can write that

$$(F_{h_1,\eta}(x, y), \ldots, F_{h_{n-t},\eta}(x, y))^\intercal = (\boldsymbol{M}^{\mathsf{Hon}})^{-1} \cdot ((G_{1,\eta}(x, y), \ldots, G_{n-t,\eta}(x, y))^\intercal - \boldsymbol{M}^{\mathsf{Corr}} \cdot ((F_{c_1,\eta}(x, y), \ldots, G_{c_t,\eta}(x, y))^\intercal)),$$

where $\mathsf{Hon} = \{h_1, \ldots, h_{n-t}\}$ and $\mathsf{Corr} = \{c_1, \ldots, c_t\}$.

Therefore, the probability that any given $((G_{1,\eta}(x, y), \ldots, G_{n-t,\eta}(x, y))$ are determined by the honest parties' shares is equal to the probability that any given $(F_{h_1,\eta}(x, y), \ldots, F_{h_{n-t},\eta}(x, y))$ are defined by the honest parties' shares. For $h \in \mathsf{Hon}$, we have from Lemma 2 that the corrupted parties' shares $\{(F_{h,\eta}(j, k), F_{h,\eta}(k, j))\}_{j\in\mathsf{Corr},k\in[n]}$ are consistent with any other degree-$(t + 2\ell - 2, t)$ bivariate polynomial $F'_{h,\eta}(x, y)$ chosen at random under the constraints that $F'_{h,\eta}(-l+1, 0) = 0$ for $l \in [\ell]$ and $F'_{h,\eta}(j, k) = F_{h,\eta}1, h(j, k), F'_{h,\eta}(k, j) = F_{h,\eta}(k, j)$, for $j \in \mathsf{Corr}, k \in [n], \eta \in [m]$. Thus, the same holds for $((G_{1,\eta}(x, y), \ldots, G_{n-t,\eta}(x, y))$ and so the $[\![\boldsymbol{0}_j]\!]_*$ are distributed randomly given the corrupted parties' shares. □

Theorem 6 (Theorem 2, restated). *$\Pi_{\mathsf{Packed\text{-}DSS}}$ UC-realizes $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ in the $\mathcal{F}_{\mathsf{batch\text{-}IC}}$-hybrid model for any $\ell \leq t/2$ and any $m = \mathtt{poly}(\kappa)$, with probability $1 - \mathtt{negl}(\kappa)$.*

Proof. We first provide a simulator $\mathcal{S}$:

1. For initialization, $\mathcal{S}$ emulates every instance of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$. We will describe the simulation of the random zero sharings in the simulation of the reconstruction phase below.
2. For sharing, we simulate as follows:
   (a) First:
      i. If the sharing party P_i is corrupt, for its emulation of each $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ instance, $\mathcal{S}$ first outputs signed to all corrupted parties then receives $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ and $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ from the adversary for $j \in \mathsf{Hon}, k \in [n]$, and sets $F_\eta(j,k)$ and $F_\eta(k,j)$ based on these, and finally outputs $\mathsf{verified}$ to all corrupted parties.
      ii. If the sharing party P_i is honest, then $\mathcal{S}$ samples random degree-$(t+\ell-1,t)$ bivariate polynomials $F_\eta(x,y)$ and for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$ $\mathcal{S}$ sends $(F_1(j,k),\ldots,F_m(j,k)$ and $(F_1(k,j),\ldots,F_m(k,j)$ for $k \in [n]$ to each corrupted party P_j, as well as signed, followed by $\mathsf{verified}$ to all other corrupted parties.
   (b) Next:
      i. If the dealer P_i was corrupt above, then for every honest party P_j, $\mathcal{S}$ checks that the $F_\eta(j,k)$ and $F_\eta(k,j)$ values corresponds to valid degree-t and degree-$(t+\ell-1)$ polynomials, respectively, and if not, for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, first sends $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ and $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ received from the adversary in the previous round first back to the adversary and then to all corrupted parties.
      ii. For honest dealer P_i, if any corrupted party P_j wants to reveals its shares, then in its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, $\mathcal{S}$ first outputs the corresponding shares to the adversary, then asks it whether it wants to reject, and if so outputs reject to all corrupted parties; otherwise, outputs the shares to all corrupted parties.
   (c) Then, if the dealer P_i was corrupt from the beginning of this sharing, if an honest party revealed shares in the previous round (or a corrupted party correctly did), then $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, abort.
   (d) Next, for every honest party P_j, $\mathcal{S}$ sends to each corrupted party P_k in its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$: if the dealer P_i was corrupted from the beginning, $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ sent by the adversary; otherwise, those values sent on behalf of the honest dealer P_i to P_k. In both cases, for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, $\mathcal{S}$ also sends to P_j: signed, followed by $\mathsf{verified}$ to all other corrupted parties. It also receives on behalf of honest P_j the same from each corrupted party P_k.
   (e) Then:
      i. For every honest party P_j, $\mathcal{S}$ checks that what it received from each corrupted P_k in the previous round is consistent with either the $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ it received from the adversary in the case of a corrupted dealer P_i or the $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ $\mathcal{S}$ sent to P_k in the case of an honest dealer P_i, and if not, for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, first sends $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ to the adversary and then to all corrupted parties.

ii. For honest dealer P_i, if any corrupted party P_j wants to reveal its shares, then in its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, $\mathcal{S}$ first outputs the corresponding shares to the adversary, then asks it whether it wants to reject, and if so outputs reject to all corrupted parties; otherwise, outputs the shares to all corrupted parties.

(f) Then, if the dealer P_i was corrupt from the beginning, if $\mathcal{S}$ hears validly revealed shares from corrupted party P_k in the last round with polynomial evaluations that are inconsistent with anything that was sent to any honest party P_j, then $\mathcal{S}$ for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, first sends $\boldsymbol{z}_{\mathsf{sid}}^{kj}$ received from P_i to the adversary and then to all corrupted parties.

(g) Then, if the dealer P_i was corrupt from the beginning, if there are any inconsistent reveals in the previous two rounds, then $\mathcal{S}$ sends to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$, abort.

(h) Finally, if the dealer was corrupt from the beginning and no abort occurred, then using the $F_\eta(k,j)$ values for $k \in [n], j \in \mathsf{Hon}$, $\mathcal{S}$ interpolates $F_\eta(x,y)$ according to Corollary 1, sets $s_\eta^l \leftarrow F_\eta(-l+1,0)$, and finally sends $(\mathsf{share}, (\boldsymbol{s}_1,\ldots,\boldsymbol{s}_m), \mathsf{sid})$ to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$. $\mathcal{S}$ also uses the interpolated $F_\eta(x,y)$ from above to compute and store the corrupted parties' shares $(\boldsymbol{z}_{\mathsf{sid}}^{j1},\ldots,\boldsymbol{z}_{\mathsf{sid}}^{jn})$. Otherwise, if the dealer is honest, $\mathcal{S}$ stores those shares which it sent to the corrupted parties at the beginning.

3. For reconstruction, we simulate as follows:

(a) First, $\mathcal{S}$ receives $(\boldsymbol{s}_1,\ldots,\boldsymbol{s}_m)$ from $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$.

(b) Then, if $\mathsf{isMult} = 0$:

i. $\mathcal{S}$ interpolates $F_\eta(j,0)$ using $F_\eta(j,1),\ldots,F_\eta(j,n)$ stored for corrupt P_j, interpolates $F_\eta(x,0)$ using $F_\eta(j,0)$ and $F_\eta(-l+1,0) \leftarrow s_\eta^l$ just received, and interpolates $F_\eta(k,y)$, for $k \in [n]$, using $F_\eta(k,0)$ and $F_\eta(k,j)$ stored for corrupt P_j.

ii. Then for its emulation of $\mathcal{F}_{\mathsf{batch\text{-}IC}}$, $\mathcal{S}$ first sends $\boldsymbol{z}_{\mathsf{sid}}^{jk}$ for $k \in [n]$ according to $F_\eta(j,y)$ to the adversary, and then to all other corrupted parties.

iii. Next, $\mathcal{S}$ initializes $T = \emptyset$ and if any corrupted party P_j's broadcasted shares in the previous round do not match what $\mathcal{S}$ stored for them (or their signatures were rejected), then $\mathcal{S}$ adds j to T.

iv. If $|T| > t - \ell$ (or $|T| > t - 2\ell$ if multiplication with a public vector occurred), then $\mathcal{S}$ sends T to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$; otherwise, $\mathcal{S}$ tells $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ to continue.

(c) If $\mathsf{isMult} = 1$ (Note the first 5 steps actually run during the initialization phase):

i. $\mathcal{S}$ first simulates as in step 2 for the sharings of $(\mathbf{0}_{i,1},\ldots,\mathbf{0}_{i,m})$ and $(\mathbf{0}_{i,m+1},\ldots,\mathbf{0}_{i,2m})$ for $i \in [n]$ in $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$, except with degree $t + 2\ell - 2$ and for corrupted dealers, it remembers the interpolated $F_\eta(x,y)$, and for honest dealers, it samples $F_\eta(x,y)$ consistent with $F_\eta(-l+1,0) = 0$ for $l \in [\ell]$.

ii. For sampling β, $\mathcal{S}$ first simulates as in step 2 for the random sharings and then for reconstruction simulates as above in the $\mathsf{isMult} = 0$ case, but with random secrets $(\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_m)$.

iii. Then, to simulate $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$, the simulator computes $F_\eta(x,y) \leftarrow \beta \cdot F^1_\eta(x,y) + F^2_\eta(x,y)$ for $\eta \in [n]$.
iv. Then $\mathcal{S}$ simulates reconstruction as above in the $\mathsf{isMult} = 0$ case, starting at step 3(b)ii.
v. If for any corrupt dealer P_i, they did not share $(\mathbf{0}_{i,1}, \ldots, \mathbf{0}_{i,m})$ and $(\mathbf{0}_{i,m+1}, \ldots, \mathbf{0}_{i,2m})$, then $\mathcal{S}$ ignores their zero sharings.
vi. Finally, for reconstruction of the re-randomized sharings, $\mathcal{S}$ samples random $F_\eta(x,y)$ consistent with (re-randomized) $F_\eta(j,k) = \boldsymbol{z}^{jk}$ stored for corrupt P_j and $F_\eta(-l+1, 0) = s^l_\eta$ for $l \in [\ell]$ and simulates as above in the $\mathsf{isMult} = 0$ case, starting at step 3(b)ii.

4. For adding sid_1 and sid_2, there is no communication to be simulated, but $\mathcal{S}$ stores for every corrupted party P_j, $\boldsymbol{z}^{jk}_{\mathsf{sid}_3} \leftarrow \boldsymbol{z}^{jk}_{\mathsf{sid}_1} + \boldsymbol{z}^{jk}_{\mathsf{sid}_2}$ for $k \in [n]$.
5. For multiplication with public vectors, there is no communication to be simulated, but letting $u_\eta(x)$ for $\eta \in [m]$ be the degree-$(\ell-1)$ polynomials such that $u_\eta(-l+1) = u^l_\eta$ for $l \in [\ell]$, $\mathcal{S}$ computes and stores for every corrupt party P_j, $\boldsymbol{z}^{jk}_{\mathsf{sid}'} \leftarrow \boldsymbol{z}^{jk}_{\mathsf{sid}} * (u_1(j), \ldots, u_m(j))$, for $k \in [n]$.
6. For corruption of a party P_i:
   (a) For all sharings in which P_i was the dealer, $\mathcal{S}$ receives from $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$ $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m)$, and interpolates each $F_\eta(x,y)$ as above using the t corrupted parties' shares originally sent from $\mathcal{S}$.
   (b) For all other sharings sid, $\mathcal{S}$ computes P_i's shares $\boldsymbol{z}^{ij}_{\mathsf{sid}}$ and $\boldsymbol{z}^{ji}_{\mathsf{sid}}$ based on the polynomials $(F_1(x,y), \ldots, F_m(x,y))$ sampled/computed for sid.

Now we show that the real world is distributed ε-close to the ideal world, for some $\varepsilon = \mathtt{negl}(\kappa)$.

For sharing, if the dealer P_i is honest, then it follows from Lemma 2 that the corrupted parties' shares are distributed identically in the ideal world as in the real world. Everything else simulated for the sharing phase is based directly on these shares, if P_i is honest, and those shares given to honest parties by P_i, if they are corrupt. Therefore, sharing is distributed identically in the ideal world as in the real world.

For reconstruction, if $\mathsf{mult} = 0$, the polynomials $F_1(x,y), \ldots, F_m(x,y)$ are determined uniquely by the corrupt parties' shares and $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ by Corollary 1. Thus, honest parties' shares are distributed identically in the ideal world and real world. Since honest parties' shares define corrupted parties' shares $\boldsymbol{z}^{j1}, \ldots, \boldsymbol{z}^{jn}$, because they correspond to degree-t polynomials, $t+1$ of which are from honest parties, whether or not corrupted parties are marked as corrupt is the same in the real and ideal worlds. Finally, because honest parties' shares define all sharings, according to Lemma 6, the vectors $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m$ output in the real and ideal worlds are the same. If $\mathsf{mult} = 1$, for the sharing phase of the zero sharings, we have that the ideal and real worlds are distributed identically as above, again using Lemma 2. We also have that for the reconstruction of $[\![\boldsymbol{\beta}]\!]$, it is defined by the corrupted parties' shares and random $\boldsymbol{\beta}$, according to Lemma 7, and so the ideal and real worlds are distributed identically. Since $\pi_{\mathsf{Check\text{-}Zero\text{-}DSS}}$ is based on the above zero sharings sharings, it is clear that the real and ideal worlds are identical for this part, as well. Also, by Lemma 8, the probability

that parties ignore some P_i's sharings who misbehaved when generating zero sharings in the ideal world, also not in the real world is $1 - \mathtt{negl}(\kappa)$. Finally, for reconstruction of the re-randomized sharings, by Lemma 9, we have that all of the 0-sharings output by $\pi_{\mathsf{Packed\text{-}Zero\text{-}DSS}}$ correspond to random polynomials $F_{1,\eta}(x, y)$ constrained to the corrupted parties' shares and $F_{1,\eta}(-l+1, 0) = 0$ for $l \in [\ell]$. Therefore, adding such random $F_{1,\eta}(x, y)$ to $F_{2,\eta}(x, y)$ constrained to the corrupted parties' shares and $F_{2,\eta}(-l+1, 0) = s_\eta^l$ for $l \in [\ell]$, results in random $F_\eta(x, y)$ constrained to the sum of the corrupted parties' shares from $F_{1,\eta}(x, y)$ and $F_{2,\eta}(x, y)$ and $F_\eta(-l+1, 0) = s_\eta^l$ for $l \in [\ell]$, which is exactly what $\mathcal{S}$ samples.

Thus, we have concluded that the real and ideal worlds are distributed identically, except with probability $\mathtt{negl}(\kappa)$. □

B.3 Proofs From Section 5.1

Lemma 22 (Lemma 11, restated). *If P is honest, then $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ outputs $(\llbracket \boldsymbol{a} \rrbracket, \llbracket \boldsymbol{b} \rrbracket, \llbracket \boldsymbol{c} \rrbracket_*)$ with probability 1. If P is corrupt, then $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ either aborts, or outputs a valid batched triple except with $\mathtt{negl}(\kappa)$ failure probability.*

Proof. The claim for honest P follows by inspection. Suppose that the procedure does not abort, and that $\boldsymbol{c} \neq \boldsymbol{a} \star \boldsymbol{b}$. By plugging in the values for $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$, one can verify that the reconstructed $\boldsymbol{z}$ is equal to

$$\begin{aligned}\boldsymbol{z} &= e \cdot \boldsymbol{c} - \boldsymbol{\gamma} - (\boldsymbol{b} - \boldsymbol{\beta}) \star \boldsymbol{\alpha} - (e \cdot \boldsymbol{a} - \boldsymbol{\alpha}) \star \boldsymbol{\beta} - (\boldsymbol{b} - \boldsymbol{\beta}) \star (e \cdot \boldsymbol{a} - \boldsymbol{\alpha}) \\ &= e \cdot (\boldsymbol{c} - \boldsymbol{a} \star \boldsymbol{b}) - (\boldsymbol{\gamma} - \boldsymbol{\alpha} \star \boldsymbol{\beta}) \\ &= e \cdot \boldsymbol{u} - \boldsymbol{v}.\end{aligned}$$

Since $\boldsymbol{u} = \boldsymbol{c} - \boldsymbol{a} \star \boldsymbol{b} \neq \boldsymbol{0}$, there exists $i \in [m\ell]$ such that $u_i \neq 0$. If $z_i = 0$, then $e \cdot u_i - v_i = 0$ and therefore $e = v_i / u_i$. However, e is sampled at random *after* the values u_i, v_i are chosen by the adversary, and hence can only happen with probability $\leq 1/|\mathbb{F}| = \mathtt{negl}(\kappa)$.

Note that, if multiple calls to $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ share the same random challenge e, then we need to perform a union bound over the number of calls in order to bound the failure probability. This is because if the challenge e is "bad" even w.r.t. one of the calls (a challenge is "bad" w.r.t. a tuple $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ if $\boldsymbol{z} = 0$ even though $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ is an incorrect triple), then the adversary will be able to make that incorrect triple accepted without detection. Having said that, note that union bounding over $t = O(n)$ calls would still provide us $\mathtt{negl}(\kappa)$ failure probability for $n = \mathtt{poly}(\kappa)$ and $|\mathbb{F}| = O(2^\kappa)$. □

Lemma 23 (Label 12, restated). *Given $n = 2t+1$ source triples, $(a_1, b_1, c_1), \ldots, (a_n, b_n, c_n)$, out of which at least $n - t$ triples are uniformly random and independent, $\mathcal{E}_{triple}$ outputs a triple $(a_{new}, b_{new}, c_{new})$ which is uniform and independent from any set of atmost t source triples.*

Proof (Sketch). We provide the high-level intuition here and refer the readers to [CP17] for the formal version. The idea behind the extractor is to construct three

(implicit) polynomials $A(x), B(x), C(x)$ of degree $t, t, 2t$ respectively such that $A(x) \cdot B(x) = C(x)$ and then output an evaluation on these triple of polynomials. This is done by constraining $A(i) = a_i$, $B(i) = b_i$ for all $i \in [t+1]$. Since $A(x)$ and $B(x)$ are supposed to be degree t polynomials, these $t+1$ constraints completely determine $A(x)$ and $B(x)$. Now in order to determine $C(x)$, one could simply take the product of $A(x)$ and $B(x)$ as formal polynomials. However, looking ahead, this approach will not be MPC friendly as it requires too many multiplications.

Therefore, [CP17] propose a different approach which is MPC friendly. The first observation is that we already know t points on $C(x)$, namely $C(i) = c_i$ for all $i \in [t+1]$. Therefore, in order to determine it completely, we need t additional points on $C(x)$. This is done in the following way: we evaluate $A(x)$ and $B(x)$ at t auxiliary points, namely at $i \in [t+2, n]$ to derive $a'_i = A(i)$ and $b'_i = B(i)$ and then use these to derive t additional points on $C(x)$ namely $c'_i = a'_i \cdot b'_i$ for all $i \in [t+2, n]$. Looking ahead, when we run the extractor securely using MPC, these t multiplications will be performed by consuming the remaining t source triples $(a_{t+2}, b_{t+2}, c_{t+2}), \ldots, (a_n, b_n, c_n)$ via Beaver multiplication. Now given the $t+1$ original points $c_1, \ldots, c_{t+1}$ and the newly derived t points $c'_{t+2}, \ldots, c'_n$, we can completely determine the polynomial $C(x)$.

Having determined $A(x), B(x), C(x)$, the last step is to simply output an evaluation on these polynomials at a consistent input point. We do this by simply outputting $(a_{\text{new}}, b_{\text{new}}, c_{\text{new}}) = (A(n+1), B(n+1), C(n+1))$ which ensures that the output triple is uniform and independent from any subset of $\leq t$ source triples. □

B.4 Proofs from Section 5

Theorem 7 (Theorem 3, restated). *Π_{Prep} UC-realizes $\mathcal{F}_{\mathsf{Prep}}$ in the $(\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$, $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1))$-hybrid model, with probability $1 - \mathtt{negl}(\kappa)$.*

Proof. We provide a simulator $\mathcal{S}$. The calls to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1)$ are all emulated internally by $\mathcal{S}$.

Input groups. For an input group with labels $\boldsymbol{\alpha}$ belonging to party P_i, simulate $\pi_{\mathsf{Input\text{-}Sharings}}$ as follows depending on whether P_i is honest or corrupt:

- If P_i is corrupt, the adversary sends $m_1 = r \cdot \mathbf{1}$ and $m_2 = r$ to $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1)$ respectively, which are emulated by $\mathcal{S}$. This process is repeated for r'. If these are not well formed (*i.e.* $m_1 \neq m_2 \cdot \mathbf{1}$), then $\mathcal{S}$ stores $(\llbracket \mathbf{0} \rrbracket, \langle 0 \rangle)$. Else, $\mathcal{S}$ stores $(\llbracket \boldsymbol{r} \rrbracket, \langle r \rangle)$, which by Lemma 15 follows the same distribution as the real world, except with probability $\mathtt{negl}(\kappa)$.
- If P_i is honest, emulate the interaction of $\pi_{\mathsf{Input\text{-}Sharings}}$, that is, inform the adversary of the calls to share, and sample a random β as part of the call to $\mathsf{rand}()$. Then, emulate $\boldsymbol{z} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{z} \rrbracket)$ and $z' \leftarrow \mathsf{reconstruct}(\langle z' \rangle)$ by opening a random $\boldsymbol{z} = z' \cdot \mathbf{1}$. This is indistinguishable from the real world since there this corresponds to $z = \beta \cdot r - r'$, which is uniformly random since r' is. For this emulated party, $\mathcal{S}$ stores the sid $\llbracket r \cdot \mathbf{1} \rrbracket, \langle r \rangle$ internally, with $r \in \mathbb{F}$ sampled uniformly at random.

Sampling random masks. For every wire α that is either an output of a multiplication gate, or an input wire, $\mathcal{S}$ simulates the call $[\![\lambda_\alpha \cdot \mathbf{1}]\!] \leftarrow \pi_{\mathsf{Rand\text{-}Sharings}}()$ as follows. First, simulate step (1) in $\pi_{\mathsf{Rand\text{-}Sharings}}$ as done above, with the difference that if the "dealer" P_i is corrupt, $\mathcal{S}$ adds P_i to the set T, and as in the procedure, P_i stores $[\![0 \cdot \mathbf{1}]\!], \langle 0 \rangle$ for parties in T. Finally, $\mathcal{S}$ computes internally $([\![s_1 \cdot \mathbf{1}]\!], \ldots, [\![s_{n-t} \cdot \mathbf{1}]\!])^\intercal \leftarrow \boldsymbol{M} \cdot ([\![r_1 \cdot \mathbf{1}]\!], \ldots, [\![r_n \cdot \mathbf{1}]\!])$ as in the real world. As in the real world $\mathcal{S}$ assigns them internally to $[\![\lambda_\alpha \mathbf{1}]\!]$ for every wire α that is either an output of a multiplication gate, or an output wire. Note that, by Lemma 16, except with probability $\mathtt{negl}(\kappa)$ these sharings are well formed and they are uniformly random in the real world, following the same distribution as the ideal world. At this point $\mathcal{S}$ can internally compute sid's $[\![\lambda_\alpha \cdot \mathbf{1}]\!]$ for every wire α.

Output groups. As in the real world, for an output group $\boldsymbol{\alpha}$, $\mathcal{S}$ takes the internally stored "sharings" $[\![\lambda_{\alpha_i} \cdot \mathbf{1}]\!]$ for $i \in [m\ell]$ from the previous step and computes $[\![\boldsymbol{\lambda_\alpha}]\!]_* = \sum_{i=1}^{m\ell} \mathbf{1}_i \cdot [\![\lambda_{\alpha_i} \mathbf{1}]\!]$.

Multiplication groups. For a multiplication group with input wires $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and output wires $\boldsymbol{\gamma}$, $\mathcal{S}$ emulates $\pi_{\mathsf{triple\text{-}generation}}$ as follows:

- Emulate $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ (step 1 in $\pi_{\mathsf{triple\text{-}generation}}$) for honest P_i:
  1. Emulate $\mathsf{rand}()$ (step 3 in $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$) by sending a random e as in the real world
  2. Emulate $\boldsymbol{\rho} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\rho}]\!])$ and $\boldsymbol{\sigma} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\sigma}]\!])$ by reconstructing random $\boldsymbol{\rho}$ and $\boldsymbol{\sigma}$. This follows the same distribution as in the real world since $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are uniformly random and unknown to the adversary.
  3. Emulate $\boldsymbol{z} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{z}]\!]_*)$ by opening $\boldsymbol{z} = \mathbf{0}$. This is the same view as in the real world where it can be checked that, for an honest P_i, $\boldsymbol{z}$ equals zero.
- Emulate $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$ (step 1 in $\pi_{\mathsf{triple\text{-}generation}}$) for honest P_i:
  1. Receive $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ from the adversary as part of the emulation of $\mathcal{F}_{\mathsf{Prep}}$ (which extends $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$).
  2. Emulate $\mathsf{rand}()$ (step 3 in $\pi_{\mathsf{verifiable\text{-}triple\text{-}sharing}}$) by sending a random e as in the real world
  3. As in the real world, emulate the reconstruction of $[\![\boldsymbol{\rho}]\!] \leftarrow e \cdot [\![\boldsymbol{a}]\!] - [\![\boldsymbol{\alpha}]\!]$, $\boldsymbol{\rho} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\rho}]\!])$ and $\boldsymbol{z} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{z}]\!]_*)$. If $\boldsymbol{z} \neq \mathbf{0}$ then $\mathcal{S}$ stores $([\![\mathbf{0}]\!], [\![\mathbf{0}]\!], [\![\mathbf{0}]\!]_*)$; otherwise it stores $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{c}]\!]_*)$. Note this is the exact same interaction as the real world.
- Emulate $\pi_{\mathsf{triple\text{-}extraction}}$ (step 2 in $\pi_{\mathsf{triple\text{-}generation}}$)
  1. The only interaction is in step 2 of $\pi_{\mathsf{triple\text{-}extraction}}$, where $[\![\boldsymbol{c'_i}]\!]_* = \pi_{\mathsf{Beaver}}(([\![\boldsymbol{a'_i}]\!], [\![\boldsymbol{b'_i}]\!]), ([\![\boldsymbol{a_i}]\!], [\![\boldsymbol{b_i}]\!], [\![\boldsymbol{c_i}]\!]))$ is called for $i \in [t+2, n]$. If P_i is honest this simulation consists of opening a random $\widetilde{\boldsymbol{a}}_i$ and $\widetilde{\boldsymbol{b}}_i$ as the reconstructions of steps 1 and 2 in π_{Beaver}, which follows the same distribution as the real world since $[\![\boldsymbol{a_i}]\!], [\![\boldsymbol{b_i}]\!]$ are uniformly random and unknown to the adversary. If P_i is corrupt then $\mathcal{S}$ simply plays the protocol as in the real world.
- From the above, $\mathcal{S}$ obtains a triple $([\![\boldsymbol{a}]\!], [\![\boldsymbol{b}]\!], [\![\boldsymbol{a} \star \boldsymbol{b}]\!]_*)$. From Lemma 14, except with probability $\mathtt{negl}(\kappa)$, this triple is uniformly random and unknown to the adversary in the real world, and the same will hold in the ideal world.

Now, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ in the calls $\boldsymbol{d} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{d} \rrbracket_*)$ and $\boldsymbol{e} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{e} \rrbracket_*)$ in step (4) by reconstructing random values $\boldsymbol{d}, \boldsymbol{e}$. This interaction looks the same as in the ideal world since, except with probability $\mathtt{negl}(\kappa)$, the vectors $\boldsymbol{a}, \boldsymbol{b}$ are uniformly random and unknown to the adversary in the real world. Finally, $\mathcal{S}$ stores $(\llbracket \boldsymbol{\lambda_\alpha} \rrbracket, \llbracket \boldsymbol{\lambda_\beta} \rrbracket, \llbracket \boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma} \rrbracket_*)$. It can be checked that these follow the same distribution as in the ideal world: from the above, except with probability $\mathtt{negl}(\kappa)$, the vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ in the real world satisfy $\boldsymbol{a} \star \boldsymbol{b} = \boldsymbol{c}$, and $\llbracket \boldsymbol{c} \rrbracket$ looks uniform to the adversary, which ensures that the linear combination in step (5)—which can be easily checked to be equal to $\boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma}$—is freshly uniform in the real world as well.

Abort. If in any of the $\mathsf{reconstruct}$ calls from above the adversary sends a set T with $|T| > t - 2(\ell - 1)$ and instructs an abort, $\mathcal{S}$ forwards the corresponding call to $\mathcal{F}_{\mathsf{Prep}}$ (which, recall, extends $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1, 1)$), which itself sends T to the actual honest parties. Since the interaction is indistinguishable in both worlds, in the real world the adversary would also send the *same* set T, which the actual functionalities $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$ and $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1, 1)$ that are used in the real world would forward to the honest parties. Hence, the two worls are indistinguishable. □

Theorem 8 (Theorem 4, restated). *Π_{MPC} UC-realizes $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep}}$-hybrid model, with probability $1 - \mathtt{negl}(\kappa)$.*

Proof. We present a simulator $\mathcal{S}$. This simulator emulates internally honest parties, as well as the functionalities $\mathcal{F}_{\mathsf{Prep}}$ and $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, together with the broadcast channel.

Preprocessing. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{Prep}}$ by storing internally the following sid's. Crucially, the different λ_* symbols below, with the exception of corrupt input wires, do *not* represent specific values, and rather they represent placeholders that $\mathcal{S}$ populates only when needed.

- $\llbracket \boldsymbol{\lambda_\alpha} \rrbracket_*$ for every output group $\boldsymbol{\alpha}$
- $(\llbracket \boldsymbol{\lambda_\alpha} \rrbracket, \llbracket \boldsymbol{\lambda_\beta} \rrbracket, \llbracket \boldsymbol{\lambda_\alpha} \star \boldsymbol{\lambda_\beta} - \boldsymbol{\lambda_\gamma} \rrbracket_*)$ for every multiplication group with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$.
- For every input group $\boldsymbol{\alpha}$ assigned to a *corrupt* party P_i, sample $\boldsymbol{\lambda}_\alpha \in \mathbb{F}^{m\ell}$ uniformly at random, and send $\boldsymbol{\lambda_\alpha}$ to P_i. Store $\langle \lambda_{\alpha_1} \rangle, \ldots, \langle \lambda_{\alpha_{m\ell}} \rangle$

Input gates. For a group of input gates $\boldsymbol{\alpha}$ owned by a party P_i, $\mathcal{S}$ acts differently according to the following:

- If P_i is honest, then $\mathcal{S}$ emulates the interaction by sampling a random $\boldsymbol{\mu_\alpha} \in \mathbb{F}^{m\ell}$ and emulating honest P_i sending it through the broadcast channel. This is indistinguishable from the real world, where an honest party broadcasts $\boldsymbol{\mu_\alpha} = \boldsymbol{v_\alpha} - \boldsymbol{\lambda_\alpha}$, since $\boldsymbol{\lambda_\alpha}$ is uniformly random and unknown to the adversary.
- If P_i is corrupt, $\mathcal{S}$ receives $\boldsymbol{\mu_\alpha}$ from the emulation of the broadcast channel, and then sets $\boldsymbol{v_\alpha} := \boldsymbol{\mu_\alpha} + \boldsymbol{\lambda_\alpha}$ (recall that $\boldsymbol{\lambda_\alpha}$ for corrupt inputs was sampled by $\mathcal{S}$ above). $\mathcal{S}$ then sends $(\mathsf{input}, P_i, v_{\alpha_i}, \alpha_i)$ to $\mathcal{F}_{\mathsf{MPC}}$ for $i \in [m\ell]$. Notice that this is the same input that P_i has used in the real world: the interaction so

far is indistinguishable, so in the real world P_i would broadcast $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ following the same distribution as in the ideal world, and the input that would be used in the real world would be precisely $\boldsymbol{\mu}_{\boldsymbol{\alpha}} + \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$.

Note that, once $\mathcal{S}$ has provided all corrupt parties' inputs to $\mathcal{F}_{\mathsf{MPC}}$, the functionality will return $\mathcal{S}$ the outputs v_α for every output wire v_α.

Addition gates. No simulation is required here.

Multiplication gates. For a group of multiplication gates with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$, $\mathcal{S}$ emulates the call $\boldsymbol{\mu}_{\boldsymbol{\gamma}} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{\mu}_{\boldsymbol{\gamma}} \rrbracket_*)$ (recall $\mathcal{F}_{\mathsf{Prep}}$ extends $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$, which is emulated by $\mathcal{S}$) by opening an internally sampled random value $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$. This is indistinguishable from the real world: there, it can be verified that this value would be equal to $\boldsymbol{v}_{\boldsymbol{\alpha}} \star \boldsymbol{v}_{\boldsymbol{\beta}} - \boldsymbol{\lambda}_{\boldsymbol{\gamma}}$, and given that $\boldsymbol{\lambda}_{\boldsymbol{\gamma}}$ is uniformly random and unknown to the adversary, so is $\boldsymbol{\mu}_{\boldsymbol{\gamma}}$.

Output gates. Given a group of output wires $\boldsymbol{\alpha}$, recall that $\mathcal{S}$ knows $\boldsymbol{v}_{\boldsymbol{\alpha}}$ from its interaction with $\mathcal{F}_{\mathsf{MPC}}$. Also, $\mathcal{S}$ knows $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ from the emulated interaction. $\mathcal{S}$ emulates the call $\boldsymbol{\lambda}_{\boldsymbol{\alpha}} \leftarrow \mathsf{reconstruct}(\llbracket \boldsymbol{\lambda}_{\boldsymbol{\alpha}} \rrbracket_*)$ by setting $\boldsymbol{\lambda}_{\boldsymbol{\alpha}} := \boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\mu}_{\boldsymbol{\alpha}}$. and return the output $\boldsymbol{v}_{\boldsymbol{\alpha}} = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} + \boldsymbol{\mu}_{\boldsymbol{\alpha}}$. This is the same distribution as the real world: we have seen that in the real world μ_β is always equal to $v_\beta - \lambda_\beta$ for any wire β, so in particular for this output group $\boldsymbol{\alpha}$ it holds that $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$ is $\boldsymbol{v}_{\boldsymbol{\alpha}} - \boldsymbol{\lambda}_{\boldsymbol{\alpha}}$, where $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ is the value stored in $\llbracket \boldsymbol{\lambda}_{\boldsymbol{\alpha}} \rrbracket$. In the ideal world, $\mathcal{S}$ sets $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ precisely in this way.

Abort and restart. So far, the executions in both worlds have been indistinguishable, except with probability $\mathtt{negl}(\kappa)$. If in the ideal world the adversary sends abort to (the emulation of) $\mathcal{F}_{\mathsf{Prep}}$ (which extends $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}$), along with a set T of corrupt parties with $|T| > t - 2(\ell - 1)$, the adversary would send the exact same set in the real world. In the ideal world, $\mathcal{S}$ from now on interacts only with the remaining parties $\{P_1, \ldots, P_n\} \setminus T$. There are $t' = t - |T|$ corrupt parties in this new set of $n' = n - |T|$ parties. Note that

$$\frac{t'}{n'} = \frac{t - |T|}{n - |T|} = \frac{t/n - |T|/n}{1 - |T|/n} < \frac{1/2 - |T|/n}{1 - |T|/n},$$

which is $\leq 1/3$ if and only if $|T|/n \geq 1/4$, or $|T| \geq n/4$. Now, let us compute $|T|$. Since $\ell = \lfloor \frac{n+6}{8} \rfloor$, we have that $\ell \leq \frac{n+6}{8}$. Then, taking into account that $n = 2t + 1$, we have:

$$\begin{aligned}
|T| >& t - 2(\ell - 1) \\
\geq& \frac{n-1}{2} - 2 \cdot \left(\frac{n+6}{8} - 1 \right) \\
=& \frac{n-1}{2} - \frac{n+6}{4} + 2 \\
=& \frac{n-8}{4} + 2 = \frac{n}{4}.
\end{aligned}$$

Hence, $|T| \geq n/4$ and in particular $t' < n'/3$, and hence the calls of $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ in the real world will not result in abort.

Now, $\mathcal{S}$ emulates the different steps in Π_{MPC} as follows:

- For every input wire α that belongs to $P_i \in T$, $\mathcal{S}$ emulates $\lambda_\alpha \leftarrow \mathsf{reconstruct}(\langle \lambda_\alpha \rangle)$ by using the values λ_α sampled during the input phase. As we have argued there, this leads to the same input $v_\alpha = \mu_\alpha + \lambda_\alpha$ that the corrupt party uses in the real world.
- For every $P_i \notin T$, let $\alpha_1, \ldots, \alpha_M$ be the input wires that belongs to P_i. Suppose P_i is corrupt (the case in which $P_i \notin T$ is honest is even simpler and we omit it):
  1. $\mathcal{S}$ receives from P_i the value λ'_{α_j} when emulating $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$. Note that $\mathcal{S}$ knows the actual values λ_{α_j} that P_i should have used, but a corrupt P_i can deviate from this.
  2. $\mathcal{S}$ receives r' from P_i when emulating $\langle r \rangle \leftarrow \mathsf{share}(r)$. $\mathcal{S}$ also receives r as input to $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, and it may happen that $r' \neq r$ if P_i decides to cheat.
  3. $\mathcal{S}$ emulates $c_1, \ldots, c_M \leftarrow \mathsf{ttrand}()$ by distributing such random elements.
  4. $\mathcal{S}$ emulates $z \leftarrow \mathsf{reconstruct}(\langle z \rangle)$ by setting $z := r + \sum_{j=1}^{M} c_j \cdot \lambda_{\alpha_j}$. Note this is the same that is computed in the real world.
  5. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ by outputting 1 to all parties if $r' + \sum_{j=1} c_j \cdot \lambda'_{\alpha_j} = z$, and 0 otherwise. It is easy to prove that, except with probability $\mathtt{negl}(\kappa)$, if some of the λ'_{α_j} is not equal to λ_{α_j}, then in both worlds the parties produce 0. Note that the two interactions are indistinguishable thus far and hence the adversary causes the same output in both worlds.
  6. If the output is 0, $\mathcal{S}$ emulates the call to $\lambda_{\alpha_j} \leftarrow \mathsf{reconstruct}(\langle \lambda_{\alpha_j} \rangle)$ by opening the λ_{α_j} sampled in the input phase. In the real world the parties set $v_{\alpha_j} = \mu_{\alpha_j} + \lambda_{\alpha_j}$, for $j \in [M]$, which is the same inputs as the ones used in the ideal world. The party P_i is added to T.
- $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$, sending the outputs received earlier from $\mathcal{F}_{\mathsf{MPC}}$ to all the parties. Note that these are the exact same outputs resulting in the real world since the inputs provided to $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ there are the same.

□

C An Alternative MPC Protocol

In Section 5.2 we presented our MPC protocol. As discussed there, the communication complexity—for the offline phase—per input wire is $O(n^2)$, which may become the bottleneck if the number of inputs is too large. In this section we describe an alternative that keeps the communication per input wire under $O(n)$, at the expense of having the communication per *output wire* to be $O(n^2)$. This is more suitable for the case in which there are many more inputs than outputs.

Let us begin by recalling briefly how the protocol Π_{MPC} from Section 5.2 handled the case of an abort. In case there is an abort when reconstructing a secret-shared value, several corrupt parties are identified and removed, and the protocol is restarted with a much lower corruption ratio 1/3. The tricky part is the case in which one of the failed reconstruction is that of an output value, since the adversary (which is rushing) may learn the output before aborting. In this case, we must ensure the corrupt parties do not change their inputs in the second

execution, since otherwise this would lead to leakage from the adversary learning two outputs on two sets of corrupt inputs. In Π_{MPC} we address this by asking the parties to robustly secret-share the λ_α's that define their inputs, which is more inefficient than our packed DSS but guarantees reconstructions. This in particular commits the corrupt parties to their inputs.

The alternative we propose in this section consists of ensuring the adversary cannot execute the rushing attack mentioned above, that is, even if the adversary decides to cause an abort in the final output reconstructions, it will not be able to learn any information about the outputs, and hence restarting with potentially fresh inputs is not a problem (so the inputs do not need to be VSS'ed anymore). To illustrate how we achieve this, consider an output group $\boldsymbol{\alpha}$. From the preprocessing the parties have $[\![\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]\!]_*$. We will require the preprocessing to output, additionally to this, VSS shares $\langle\lambda_{\alpha_1}\rangle, \ldots, \langle\lambda_{\alpha_\ell}\rangle$. Once this is done, our problem is solved: in the online phase, if the parties abort *before* attempting to reconstruct any output wire, then no leakage has occured and it is safe to restart (possibly with different inputs). However, if the parties successfully reach the point where the only missing operation is to reconstruct the outputs, this will *not* result in abort since the parties know the broadcasted values $\boldsymbol{\mu}_{\boldsymbol{\alpha}}$, and the associated masks $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ are somehow VSS'ed, which are guaranteed to be reconstructed regardless of the adversary's behavior. We discuss this idea in more detail below.

C.1 Modified Preprocessing

Our modified preprocessing functionality is as below.

Functionality 5: $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$

Everything is the same as $\mathcal{F}_{\mathsf{Prep}}$, Functionality 4, except for the following: **Input and output sharings.**

- For every group of $m\ell$ input gates with labels $\boldsymbol{\alpha}$ belonging to party P_i, send $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$ to P_i.
- For every group of $m\ell$ output gates with labels $\boldsymbol{\alpha}$, sample a random $\boldsymbol{\omega}_{\boldsymbol{\alpha}} \leftarrow_{\$} \mathbb{F}^{m\ell}$, store $\langle\lambda_{\alpha_j} - \omega_{\alpha_j}\rangle$ for $j \in [m\ell]$, and send $\boldsymbol{\rho}_{\boldsymbol{\alpha}} = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} - \boldsymbol{\omega}_{\boldsymbol{\alpha}}$ to all parties.

This functionality, although similar to $\mathcal{F}_{\mathsf{Prep}}$, is slightly harder to instantiate: in $\mathcal{F}_{\mathsf{Prep}}$ there is a single party that knows the λ_α that must be VSS'ed, and this party can provide such sharings (which are checked against the DSS'ed version). In our case, in $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$ the ω_α to be VSS'ed is uniformly random and unknown to any of the parties. For this we adapt the Procedures $\pi_{\mathsf{Input\text{-}Sharings}}$ and $\pi_{\mathsf{Rand\text{-}Sharings}}$, essentially moving the VSS part from $\pi_{\mathsf{Input\text{-}Sharings}}$ to $\pi_{\mathsf{Rand\text{-}Sharings}}$. We describe these in detail below for completeness.

Procedure 18: $\pi_{\mathsf{Input\text{-}Sharings\text{-}Alt}}(P_i)$

This procedure lets P_i share $[\![r \cdot \mathbf{1}]\!]$, where $r \in \mathbb{F}$ is random.

1. P_i samples $r, r' \in \mathbb{F}$ and calls $[\![r \cdot \mathbf{1}]\!] \leftarrow \mathsf{share}(r \cdot \mathbf{1})$ and $[\![r \cdot \mathbf{1}]\!] \leftarrow \mathsf{share}(r' \cdot \mathbf{1})$ (using $\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m)$).
2. The parties then call $\beta \leftarrow \mathsf{rand}()$ and compute $[\![\boldsymbol{z}]\!] \leftarrow \beta \cdot [\![r \cdot \mathbf{1}]\!] - [\![r' \cdot \mathbf{1}]\!]$.
3. Parties call $\boldsymbol{z} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{z}]\!])$. If $\boldsymbol{z} \neq z' \cdot \mathbf{1}$ for some $z' \in \mathbb{F}$, then output $[\![\mathbf{0}]\!]$). Else, output $[\![r \cdot \mathbf{1}]\!]$

Procedure 19: $\pi_{\mathsf{Rand\text{-}Sharings\text{-}Alt}}$

This procedure gives parties shares $([\![\boldsymbol{r}]\!], \langle r_1 \rangle, \ldots, \langle r_{m\ell} \rangle)$. It outputs $n - t$ such tuples. Initialize $T = \emptyset$.

1. For every party P_i, the parties do the following:
   (a) P_i samples $\boldsymbol{r}_i, \boldsymbol{r}'_i \in \mathbb{F}$ and calls $[\![\boldsymbol{r}_i]\!] \leftarrow \mathsf{share}(\boldsymbol{r}_i)$ and $[\![\boldsymbol{r}_i]\!] \leftarrow \mathsf{share}(\boldsymbol{r}'_i)$, and also $\langle r_{ij} \rangle \leftarrow \mathsf{share}(r_{ij})$ and $\langle r'_{ij} \rangle \leftarrow \mathsf{share}(r'_{ij})$ for $j \in [m\ell]$.
   (b) The parties then call $\beta \leftarrow \mathsf{rand}()$[a] and compute $[\![\boldsymbol{z}_i]\!] \leftarrow \beta \cdot [\![\boldsymbol{r}_i]\!] - [\![\boldsymbol{r}'_i]\!]$, as well as $\langle z'_{ij} \rangle \leftarrow \beta \cdot \langle r_{ij} \rangle - \langle r'_{ij} \rangle$.
   (c) Parties call $\boldsymbol{z}_i \leftarrow \mathsf{reconstruct}([\![\boldsymbol{z}_i]\!])$ and $z'_{ij} \leftarrow \mathsf{reconstruct}(\langle z'_{ij} \rangle)$ for $j \in [m\ell]$. If $\boldsymbol{z}_i \neq (z_{i,1}, \ldots, z_{i,m\ell})$, add P_i to the set T.
2. For $P_i \in T$ set $\boldsymbol{r}_i := \mathbf{0}$ and store $([\![\boldsymbol{r}]\!], \langle r_1 \rangle, \ldots, \langle r_{m\ell} \rangle)$ for this party.
3. All parties then compute and output

$$([\![\boldsymbol{s}_1]\!], \ldots, [\![\boldsymbol{s}_{n-t}]\!])^\intercal \leftarrow \boldsymbol{M} \cdot ([\![\boldsymbol{r}_1]\!], \ldots, [\![\boldsymbol{r}_n]\!])$$
$$(\langle s_{1,j} \rangle, \ldots, \langle s_{n-t,j} \rangle)^\intercal \leftarrow \boldsymbol{M} \cdot (\langle r_{1,j} \rangle, \ldots, \langle r_{n,j} \rangle) \quad \textit{for} \quad j \in [m\ell].$$

[a] One single random value can be reused across all parties, and this is important for efficiency.

Equivalent versions of Lemmas 15 and 16 can be stated and proven as well. With this at hand, the finally preprocessing protocol is the following.

Protocol 20: $\Pi_{\mathsf{Prep\text{-}Alt}}$

Everything is the same as Π_{Prep}, Protocol 16 ("sampling random masks" still uses $\pi_{\mathsf{Rand\text{-}Sharings}}$), except for the following:

Input groups. For an input group $\boldsymbol{\alpha}$ associated to party P_i, call $[\![\lambda_{\alpha_i} \cdot \mathbf{1}]\!] \leftarrow \pi_{\mathsf{Input\text{-}Sharings\text{-}Alt}}(P_i)$ for $i \in [m\ell]$.

Output groups. For an output group $\boldsymbol{\alpha}$:

1. The parties call $\pi_{\mathsf{Rand\text{-}Sharings\text{-}Alt}}()$, which results in random values $([\![\boldsymbol{\omega}_{\boldsymbol{\alpha}}]\!], \langle \omega_{\alpha_1} \rangle, \ldots, \langle \omega_{\alpha_{m\ell}} \rangle)$. Store the VSS sharings.
2. Compute $[\![\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]\!]_*$ as in $\mathcal{F}_{\mathsf{Prep}}$. Compute $[\![\boldsymbol{\rho}_{\boldsymbol{\alpha}}]\!]_* \leftarrow [\![\boldsymbol{\lambda}_{\boldsymbol{\alpha}}]\!]_* - [\![\boldsymbol{\omega}_{\boldsymbol{\alpha}}]\!]$, and call $\boldsymbol{\rho}_{\boldsymbol{\alpha}} \leftarrow \mathsf{reconstruct}([\![\boldsymbol{\rho}_{\boldsymbol{\alpha}}]\!]_*)$. If this does not abort, all parties also store $\boldsymbol{\rho}_{\boldsymbol{\alpha}}$.

This theorem is proven similarly to 3

Theorem 9. *$\Pi_{\mathsf{Prep\text{-}Alt}}$ UC-realizes $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$ in the $(\mathcal{F}_{\mathsf{Packed\text{-}DSS}}(\ell, m), \mathcal{F}_{\mathsf{Packed\text{-}DSS}}(1,1))$-hybrid model.*

C.2 Modified Online Phase

We are finally ready to describe our alternative online phase. As discussed earlier, it works almost the same as Π_{MPC}, Protocol 17, except that (1) the parties do not have VSS'ed input masks, and (2) the parties have VSS'ed output offsets that ensure correct reconstruction without aborts, once that phase is reached. Otherwise, the protocol is restarted with a smaller threshold.[12] The protocol is described below.

Protocol 21: $\Pi_{\mathsf{MPC\text{-}Alt}}$

This protocol makes use of $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$ and $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$.

Preprocessing. The parties call $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$ to obtain:
- $\langle \omega_{\alpha_1} \rangle, \ldots, \langle \omega_{\alpha_{m_\ell}} \rangle$ and public $\boldsymbol{\rho}_{\boldsymbol{\alpha}} = \boldsymbol{\lambda}_{\boldsymbol{\alpha}} - \boldsymbol{\omega}_{\boldsymbol{\alpha}}$ for every output group $\boldsymbol{\alpha}$
- $(\llbracket \boldsymbol{\lambda}_{\boldsymbol{\alpha}} \rrbracket, \llbracket \boldsymbol{\lambda}_{\boldsymbol{\beta}} \rrbracket, \llbracket \boldsymbol{\lambda}_{\boldsymbol{\alpha}} \star \boldsymbol{\lambda}_{\boldsymbol{\beta}} - \boldsymbol{\lambda}_{\boldsymbol{\gamma}} \rrbracket_*)$ for every multiplication group with inputs $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and outputs $\boldsymbol{\gamma}$.
- For every input group $\boldsymbol{\alpha}$ assigned to a party P_i, this party knows $\boldsymbol{\lambda}_{\boldsymbol{\alpha}}$.

Input, Addition and Multiplication Gates. Same as Π_{MPC}.

Output Gates. Given an output wire α, call $\omega_\alpha \leftarrow \mathsf{reconstruct}(\langle \omega_\alpha \rangle)$, and return the output $v_\alpha = \omega_\alpha + \rho_\alpha + \mu_\alpha$.

Abort and restart. If any of the calls above results in abort, a set T of corrupt parties with $|T| > t - 2(\ell - 1)$ is identified. The new set of parties is $\{P_1, \ldots, P_n\} \setminus T$, where $n' = n - |T|$ and $t' = n - |T|$. They use $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ to securely compute C, where the inputs of parties in T are set to 0. Output the result of this call.

The following theorem is proven similarly to Theorem 4.

Theorem 10. *$\Pi_{\mathsf{MPC\text{-}Alt}}$ UC-realizes $\mathcal{F}_{\mathsf{MPC}}$ in the $\mathcal{F}_{\mathsf{Prep\text{-}Alt}}$-hybrid model.*

[12] We use $\mathcal{F}_{\mathsf{MPC\text{-}t<n/3}}$ for the restart to finish within one repetiton, but actually with this variant we can keep using our own protocol even for the restarts, which is guaranteed to finish within a *constant* number of repetitions. This would not affect our desired round complexity (asymptotically).