Detect, Pack and Batch: Perfectly-Secure MPC with Linear Communication and Constant Expected Time

Ittai Abraham* Gilad Asharov† Shravani Patil‡ Arpita Patra§

April 19, 2023

Abstract

We prove that perfectly-secure optimally-resilient secure Multi-Party Computation (MPC) for a circuit with C gates and depth D can be obtained in $\mathcal{O}((Cn + n^4 + Dn^2)\log n)$ communication complexity and $\mathcal{O}(D)$ expected time. For $D \ll n$ and $C \geq n^3$, this is the **first** perfectly-secure optimal-resilient MPC protocol with **linear** communication complexity per gate and **constant** expected time complexity per layer.

Compared to state-of-the-art MPC protocols in the player elimination framework [Beerliova and Hirt TCC'08, and Goyal, Liu, and Song CRYPTO'19], for $C > n^3$ and $D \ll n$, our results significantly improve the run time from $\Omega(n+D)$ to expected $\mathcal{O}(D)$ while keeping communication complexity at $\mathcal{O}(Cn \log n)$.

Compared to state-of-the-art MPC protocols that obtain an expected $\mathcal{O}(D)$ time complexity [Abraham, Asharov, and Yanai TCC'21], for $C > n^3$, our results significantly improve the communication complexity from $\mathcal{O}(Cn^4 \log n)$ to $\mathcal{O}(Cn \log n)$ while keeping the expected run time at $\mathcal{O}(D)$.

One salient part of our technical contribution is centered around a new primitive we call *detectable secret sharing*. It is perfectly-hiding, weakly-binding, and has the property that either reconstruction succeeds, or $\mathcal{O}(n)$ parties are (privately) detected. On the one hand, we show that detectable secret sharing is sufficiently powerful to generate multiplication triplets needed for MPC. On the other hand, we show how to share p secrets via detectable secret sharing with communication complexity of just $\mathcal{O}(n^4 \log n + p \log n)$. When sharing $p \geq n^4$ secrets, the communication cost is amortized to just $\mathcal{O}(1)$ field elements per secret.

Our second technical contribution is a new Verifiable Secret Sharing protocol that can share p secrets at just $\mathcal{O}(n^4 \log n + pn \log n)$ word complexity. When sharing $p \geq n^3$ secrets, the communication cost is amortized to just $\mathcal{O}(n)$ filed elements per secret. The best prior required $\Omega(n^3)$ communication per secret.

*VMWare Research. iabraham@vmware.com

†Department of Computer Science, Bar-Ilan University, Israel. Gilad.Asharov@biu.ac.il. Sponsored by the Israel Science Foundation (grant No. 2439/20), by JPM Faculty Research Award, and by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 891234.

‡Indian Institute of Science, Bangalore, India. shravanip@iisc.ac.in. Supported by DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020-2025.

§Indian Institute of Science, Bangalore, India. arpita@iisc.ac.in. Supported by DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020-2025, Google India Faculty Award, and JPM Faculty Research Award.

Contents

1 Introduction

In the setting of secure multiparty computation (MPC), n distrustful parties jointly compute a function on their inputs while keeping their inputs private. Security should be preserved even in the presence of an external entity that controls some parties and coordinates their behavior. We consider in this paper the most demanding setting: perfect security with optimal resilience. Perfect security means that the adversary is all-powerful and that the protocol has zero probability of error. Optimal resilience means that the number of corruptions is at most $t < n/3$. Such protocols come with desirable properties: They guarantee adaptive security (with some caveats [18, 5]) and remain secure under universal composition [36].

The seminal protocols of Ben-Or, Goldwasser, and Wigderson [13], and Chaum, Crépeau and Damgård [20] led the foundations of this setting. Since then, there are, in general, two families of protocols:

1. **Efficient but slow:** These protocols [34, 12, 32] ([12] test-of-time award) have $\mathcal{O}(n \log n)$ communication complexity per multiplication gate. Still, the running time of these protocols is at least $\Theta(n)$ rounds, even if the depth of the circuit is much smaller $D \ll n$. Specifically:

   > **Theorem 1.1.** *For an arithmetic circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\mathcal{O}(n^5 \log n + Cn \log n)$ bits communication complexity and $\Omega(n + D)$ expected number of rounds.*

   The protocol requires $\mathcal{O}(n^3 \log n + Cn \log n)$ bits of point-to-point communication and n *sequential* invocations of broadcast of $\mathcal{O}(\log n)$ bits each, with $\Omega(n + D)$ rounds. Using the broadcast implementation of [1], this becomes the complexity of Theorem 1.1. Alternatively, using the implementation of [15, 24], the protocol can be more efficient, but even more slower: $\mathcal{O}(n^3 \log n + Cn \log n)$ bits communication complexity and $\Omega(n^2 + D)$ number of rounds.

2. **Fast but not efficient:** This line of protocols [13, 20, 31, 25, 7, 2] run at $\mathcal{O}(D)$ expected number of rounds, but require $\Omega(n^4 \log n)$ communication complexity per multiplication gate.

   > **Theorem 1.2.** *For an arithmetic circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\Omega(Cn^4 \log n)$ communication complexity and $\mathcal{O}(D)$ expected number of rounds.*

   In the broadcast hybrid model, the protocol requires $\mathcal{O}(n^3 \log n)$ bits of communication complexity over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast, in $\mathcal{O}(D)$ number of rounds. Theorem 1.2 reports the communication complexity using the broadcast implementation of [1]. Using [15, 24] for implementing the broadcast, the number of rounds is increased to $\Omega(n + D)$.

Our Main Result

Our main result is that it is possible to simultaneously achieve the best of both families. For the first time, we provide a perfectly-secure, optimally-resilient MPC protocol that has **both** $\mathcal{O}(n \log n)$ communication complexity per multiplication gate and $\mathcal{O}(D)$ expected time complexity.

Theorem 1.3 (Main Result)**.** *For a circuit with C multiplication gates and depth D there exists a perfectly-secure, optimally-resilient MPC protocol with $\mathcal{O}((Cn + Dn^2 + n^4)\log n)$ communication complexity and $\mathcal{O}(D)$ expected number of rounds.*

In the broadcast-hybrid model, the total communication complexity over point-to-point is $\mathcal{O}((Cn + Dn^2 + n^4)\log n)$, and each party has to broadcast at most $\mathcal{O}(n^2 \log n)$ bits. Using [1] for implementing the broadcast, we obtain Theorem 1.3. Compared to [12, 32], for $D \ll n$, our result provides up to an $\mathcal{O}(n)$ improvement in round complexity while keeping the same linear communication complexity (and also improving the communication complexity for $C \in o(n^4)$). Compared to [2], for $C > n^3$, our result provides an $\mathcal{O}(n^3)$ improvement in the communication complexity while keeping the same $\mathcal{O}(D)$ expected round complexity.

We remark that in many practical settings, a large set of parties may want to compute a shallow depth circuit in a robust manner. For instance, consider a network with 200ms latency and channels of 1Gbps, and consider a highly parallel circuit with 1M gates, depth $D = 10$, and $n = 200$ parties. Then, the round complexity of our protocol is $\mathcal{O}(D)$, which results in a delay of $10 \cdot 200\text{ms} = 2$ seconds. The delay associated with the communication complexity is smaller: each party sends or receives $(C+Dn+n^3)\log n$ bits, which over 1Gbps channel results in a delay of 0.08 seconds. In [32], the delay due to the round complexity is $\mathcal{O}(n + D)$, which results in a delay of $210 \cdot 200\text{ms} = 42$ seconds, and each party sends or receives $(C + n^4)\log n$ bits which over 1Gbps results in a delay of ≈ 14 seconds. If we use [15, 24] to implement the broadcast, then the round complexity becomes $\mathcal{O}(n^2 + D)$ which is ≈ 8000 seconds. The improvement in the round complexity is significant in this scenario. Of course, these are only coarse estimations that do not even take into account the hidden constants in the $\mathcal{O}$ notation.

Main Technical Result

Our main result is obtained via several advances in building blocks for perfectly secure optimally resilient MPC. In our view, the most important and technically involved contribution is a new primitive called *Detectable Secret Sharing*. This is a secret sharing with the following properties: (1) Secrecy: The corrupted parties cannot learn anything about the secrets after the sharing phase for an honest dealer; (2) Binding: After the sharing phase (even if the dealer is corrupted), the secret is well defined by the shares of the honest parties; (3) Reconstruction or detection: Reconstruction ends up in the well-defined secret, or it might fail (even if the dealer is honest). However, in the case of failure, there is a (private) detection of $\mathcal{O}(n)$ corrupted parties. Moreover, successful sharing and reconstruction are guaranteed if the dealer has already detected more than $t/2$ corrupted parties before the respective phases.

We show that despite the possible failure of the reconstruction, Detectable Secret Sharing suffices for obtaining our end result for MPC. Most importantly, we obtain a highly efficient construction for this primitive:

Theorem 1.4 (informal)**.** *There exists a detectable secret sharing protocol that allows sharing p secrets (of $\log n$ bits each) with malicious security and optimal resilience with $\mathcal{O}(n^4 \log n + p \log n)$ communication complexity and expected constant number of rounds.*

For $p \geq n^4$, this is $\mathcal{O}(1)$ field elements per secret (which is also a field element)! This matches packed semi-honest secret sharing as in [30]. The theorem holds for a single dealer; for n dealers, each sharing p secrets in parallel, we get $\mathcal{O}(1)$ field elements per secret starting from $p \geq n^3$.

Stated differently, we show a detectable secret sharing protocol that can pack $\mathcal{O}(n^2)$ secrets (of size $\log n$ each) at the cost of $\mathcal{O}(n^2 \log n)$ communication complexity for private channels and each party broadcasts at most $\mathcal{O}(n \log n)$ bits, with a strictly constant number of rounds. There are at least two striking features of our new detectable secret sharing: *packing*, and *batching*. First, to the best of our knowledge, this is the first protocol in the malicious setting that can pack $\mathcal{O}(n^2)$ secrets at the cost of $\mathcal{O}(n^2)$ communication complexity – so an amortized cost of $\mathcal{O}(1)$ per secret over point-to-point channels. Second, our scheme allows batching – m independent instances with the same dealer require $\mathcal{O}(mn^2 \log n)$ over point-to-point channels but just $\mathcal{O}(n \log n)$ broadcast per party in all m instances combined. To the best of our knowledge, this is the first protocol that requires a fixed broadcast cost independent of the batching parameter m. By setting $m = p/n^2$ and combining with the recent broadcast implementation of Abraham, Asharov, Patil, and Patra [1], we obtain Theorem 1.4 in the point-to-point channel model and no broadcast.

Note that this primitive is formally incomparable with weak-secret sharing [38] (where reconstruction needs the help of the dealer but is guaranteed to succeed when the dealer is honest). On the one hand, our notion seems weaker as there is no guaranteed validity (no guaranteed reconstruction in case of an honest dealer). On the other hand, it is not strictly weaker since our notion ensures mass detection in case of a reconstruction failure. For comparison, the best known weak-secret sharing [2] requires $\mathcal{O}(n^4 \log n)$ for sharing $\mathcal{O}(n)$ secrets (i.e., $\mathcal{O}(n^3)$ per secret).

Verifiable secret sharing. We also derive (and use) a "strong" secret sharing (i.e., honest parties always succeed to reconstruct), i.e., in the standard verifiable secret sharing [21] setting:

Theorem 1.5 (informal)**.** *There exists a protocol that allows to secret share p secrets (of $\log n$ bits each) with malicious security and optimal resilience with $\mathcal{O}(n^4 \log n + p \cdot n \log n)$ communication complexity and expected constant number of rounds.*

For $p \geq n^3$, this is an overhead of $\mathcal{O}(n)$ per secret. Previously, the best known [1] in this setting packs $\mathcal{O}(n)$ secrets with $\mathcal{O}(n^4 \log n)$ communication complexity (an overhead of $\mathcal{O}(n^3)$ per secret). This is an improvement of $\mathcal{O}(n^2)$ over the state-of-the-art. In comparison, the starting point is the VSS of BGW and Feldman [13, 28] requires $\mathcal{O}(n^2 \log n)$ point-to-point and $\mathcal{O}(n^2 \log n)$ broadcast, for sharing just a single secret. This results in $\mathcal{O}(n^4 \log n)$ communication complexity over point-to-point channels and no broadcast, for sharing just a single secret (an overhead of $\mathcal{O}(n^4)$).

Detection. The line of work of [34, 12, 32] in perfectly-secure MPC is based on the *player elimination framework* (introduced by Hirt, Maurer and Przydatek [34]). The protocol identifies a set of parties in which it is guaranteed that one of the players among the set is corrupted, excludes the entire set, and restarts the protocol. The important aspect here is that all parties agree on the set, and that honest parties are also "sacrificed" along the way. In each iteration, the number of parties being excluded is constant. This is a slow process that leads to the $\mathcal{O}(n)$ rounds overhead.

Instead of globally eliminating a set of parties, our approach is to have each party maintain a local set of conflicted parties, with no global agreement among parties on who is malicious. Each party can decide which parties to mark as conflicted while it shares its own secret(s). When an honest party marks enough corrupt parties as conflicted, its sharing will always be successful. Moreover, whenever there is a failure in sharing or reconstruction, then there is a mass detection – $\mathcal{O}(n)$ corruptions are identified, either publicly or privately.

To elaborate further, our MPC protocol uses three kinds of detections:

1. *Global detection* – wherein a set of parties is excluded from the computation. Unlike [34, 12, 32], in our case, honest parties are never discarded (e.g., "discard the dealer").
2. *Public individual detection* – wherein each party has its own conflict set that is publicly known to all (see, e.g., Step 2b in Protocol 4.2). While a similar mechanism, referred to as 'dispute control' has been used in [11, 14, 33], these works achieve *statistical* security in the honest majority setting with $\mathcal{O}(n)$ rounds overhead similar to the player-elimination framework;
3. *Private (local) detection* – wherein each party has its private conflict set that it excludes from its local computation. Specifically, an honest party may locally identify a set conflicts (with corrupted parties) without a mechanism to prove that it has done so honestly. In our protocol, it can identify $\mathcal{O}(n)$ such conflicts simultaneously in case private reconstruction towards it fails (see, e.g., Step 7 in Protocol 5.2). This allows an honest party to locally discard the communication from $\mathcal{O}(n)$ corrupt parties, eventually ensuring a successful reconstruction.

1.1 Related Work

Broadcast. Our communication complexity takes into account the cost of broadcast. In the setting of perfect security, there are two families of protocols for implementing the broadcast: once again – efficient and slow, or fast but less efficient. The former [15, 24] takes $\mathcal{O}(n)$ rounds and $\mathcal{O}(n^2 + pn)$ for broadcasting a message of p bits. The latter [1] (built upon Feldman and Micali [27], and Katz and Koo [35]) takes $\mathcal{O}(1)$ expected number of rounds and $\mathcal{O}(n^4 + pn)$ communication complexity for broadcasting a message of size p bits, i.e., this is optimal for $p > n^3 \log n$. Note that when broadcasting a message of size p, then since each party is supposed to receive p bits, the minimal possible communication complexity is pn. Moreover, n parties broadcasting messages of size p bits each takes $\mathcal{O}(n^4 + pn^2)$, i.e., optimal for $p > n^2 \log n$. We also remark that containing strict $\mathcal{O}(1)$ number of rounds is impossible [29].

Shunning. Our notion of detectable secret sharing can be viewed as a synchronous analog of the notion of shunning, in which parties either succeed in their asynchronous verifiable secret sharing or some detection event happens. In the context of asynchronous verifiable secret sharing, shunning was first suggested by Abraham, Dolev, and Halpern [3] and later improved and extended to shunning $\mathcal{O}(n)$ parties by Bangalore, Choudhury, and Patra [8, 9]. However, unlike our detectable secret sharing, none of these works attain $\mathcal{O}(1)$ amortized communication cost per secret.

2 Technical Overview

In this section, we provide a technical overview of our work. We start in Section 2.1 with an overview of our main technical result – our detectable and verifiable secret sharing schemes. In Section 2.2 we overview our MPC result. Most of the building blocks are based on previous works, and we highlight in the overview the steps where we made significant improvements. In Section 2.3 we overview another step in the protocol, triplet secret sharing.

2.1 Detectable and Verifiable Secret Sharing

We start this overview with the most basic verifiable secret sharing protocol – the one by BGW [13]. See also [6, 4] for further details. To share a secret s, the dealer chooses a bivariate polynomial $S(x, y) = \sum_{k=0}^{t} \sum_{\ell=0}^{t} s_{k,\ell} \cdot x^k y^\ell$ of degree t in both x and y under the constraint that $S(0, 0) =$

$s_{0,0} = s$. The share of each party P_i is the pair of degree-t univariate polynomials $S(x,i), S(i,y)$. The goal of the verification step is to verify that the shares of all honest parties indeed lie on a unique bivariate polynomial $S(x,y)$. Let us briefly recall the sharing phase:

1. **Sharing:** The dealer sends the share $(f_i(x), g_i(y)) = (S(x,i), S(i,y))$ to each party P_i.
2. **Pairwise checks:** P_i sends to each P_j the two points $(f_i(j), g_i(j)) = (S(j,i), S(i,j)) = (g_j(i), f_j(i))$. If P_i did not receive from P_j the points it expects to see (i.e., that agree with $f_i(x), g_i(y)$), then it publicly broadcasts a complaint $\mathsf{complaint}(i, j, f_i(j), g_i(j))$.
3. **Publicly resolving the complaints:** The dealer checks all complaints; if some party P_i publicly complains with values that are different than what the dealer has sent it, then the dealer makes the share of P_i public – i.e., it broadcasts $\mathsf{reveal}(i, S(x,i), S(i,y))$.
4. If a party P_j sees that (1) all polynomials that the dealer made public agree with its private shares; (2) its share was not made public; (3) if two parties P_k and P_ℓ both complaint on each other, then the dealer must open one of them. If all those conditions hold, then P_j is happy with its share, and votes to accept the dealer. If the shares of P_j were made public, then it re-assigns $f_j(x), g_j(y)$ to the publicly revealed ones.
5. If $2t+1$ parties votes to accept the shares, then each party output its share. Otherwise, the dealer is discarded.

Observe that if the dealer is honest, then during the verification phase the corrupted parties do not learn anything new. Specifically, a party always broadcasts a complaint with the values that it received from the dealer, and the dealer makes a share public only if the public complaint does not contain the values that the dealer has sent that party. Therefore, an honest dealer never makes the shares of another honest party public. Moreover, all honest parties are happy, and accept the shares.

If the dealer is corrupted, then $2t+1$ parties that voted to accept the dealer implies that we have a set $J \subseteq [n]$ of at least $t+1$ honest parties that are happy with their shares and that their shares were never made public. The shares of those $t+1$ honest parties fully determine a bivariate polynomial of degree-t in both variables. If some honest party P_j initially held a share that does not agree with this bivariate polynomial, i.e., does not agree with some P_k for $k \in J$, then it must be that P_j and P_k both publicly complained, and that the share of P_j was made public with some new share that agrees with S (if it does not agree with S, then at least one party in J would have not voted to accept). Therefore, at the end, all honest parties hold shares of a well-defined bivariate polynomial.

To reconstruct the bivariate polynomial, each party sends to each other party its pair of polynomials. Since the underlying polynomial is of degree-t, the adversary controls at most t parties, we must have $n - t \geq 2t + 1$ correct points and at most t errors. The Reed-Solomon decoding procedure guarantees that the t errors can be identified and corrected.

Our improvements. The above scheme for verifiable secret sharing requires $\mathcal{O}(n^2 \log n)$ communication over the point-to-point channels, and also the broadcast of $\mathcal{O}(n^2 \log n)$ bits. This results in total communication complexity of $\mathcal{O}(n^4 \log n)$ over point-to-point for sharing a single secret. The work of [1] has the same complexity for sharing $\mathcal{O}(n)$ secrets.

For the same communication complexity, we show how to do detectable secret sharing for $\mathcal{O}(n^4)$ secrets or to do (standard) verifiable secret sharing for $\mathcal{O}(n^3)$ secrets. Looking ahead, we improve the basic scheme in the following aspects, each giving a factor of $\mathcal{O}(n^2)$ improvement for our detectable secret sharing:

(1) Packing: The bivariate polynomial $S(x, y)$ in the basic construction contains only a single secret, located at $S(0, 0)$. This is the best possible when sharing a bivariate polynomial of degree-t in both x and y: The t shares of the corrupted parties, together with the secret, fully determine the bivariate polynomial. In our detectable secret sharing scheme, the dealer shares a bivariate polynomial of degrees greater than t in *both* x and y. This allows planting $\mathcal{O}(n^2)$ secrets. The verification that all parties hold shares on the same bivariate polynomial is much more challenging because the degrees of all the univariate polynomials are greater than t. Nevertheless, we obtain binding with asymptotically the same cost as the basic scheme, therefore we already obtain an improvement of $\mathcal{O}(n^2)$ over the basic scheme.

Moreover, once the degree in both dimensions is greater than t, then reconstruction might fail because the underlying codeword is of degree greater than t, and the parties cannot necessarily correct the errors if the adversary does not provide correct shares. Nevertheless, Reed-Solomon decoding guarantees that the honest parties can (efficiently) *identify* whether there is a unique decoding or not. We use this property to also *detect sufficiently many* corrupted parties. This suffices for constructing a detectable secret sharing scheme.

For (standard) verifiable secret sharing, we must make the degree in at least one of the dimensions to be at most t, to allow to always succeed in correcting errors. This allows us to pack "only" $\mathcal{O}(n)$ secrets and not $\mathcal{O}(n^2)$.

(2) Batching: The verification step of [13] requires broadcasting $\mathcal{O}(n^2)$ field elements by the dealer, and $\mathcal{O}(n)$ field elements by each party. Hence m independent instances (with the same dealer) require broadcasting of $\mathcal{O}(mn^2)$ field elements. First, we *balance* the protocol such that each party broadcasts at most $\mathcal{O}(n)$ field elements, including the dealer. Second, by designing a sharing protocol that is tailored for achieving cheap batching, *the broadcast cost for m independent instances remains the same as a single instance, i.e., it requires each party to broadcast $\mathcal{O}(n \log n)$ bits in all m executions combined.* By setting $m = \mathcal{O}(n^2)$ and implementing the broadcast over point-to-point, we get a detectable secret sharing of $\mathcal{O}(n^4)$ secrets (each is a field element of size $\mathcal{O}(\log n)$) at the cost of $\mathcal{O}(n^4 \log n)$ communication over the point-to-point channels. This is the second $\mathcal{O}(n^2)$ improvement over the basic scheme.

Our batched and packed detectable secret sharing protocol. For our discussion, assume that the dealer first chooses a polynomial $S(x, y)$ of degree $t + t/4$ in x and degree $t + t/4$ in y. We will use different parameters in the actual construction later,[1] but we choose $t + t/4$ for simplicity of exposition in this overview. Like the basic scheme, the view of the adversary consists of the pair of the univariate polynomials $S(x, i), S(i, x)$, for every $i \in I$, where $I \subseteq [n]$ is the set of indices of the corrupted parties (of cardinality at most t). This means that the adversary receives at most $2t(t+t/4+1)-t^2$ values, and therefore the dealer can still plant $(t/4+1)^2 \in \mathcal{O}(n^2)$ secrets in $S(x, y)$, which is fully determined by $(t+t/4+1)^2$ values. Concretely, it can plant for every $a \in \{0, \ldots, t/4\}$ and $b \in \{0, \ldots, t/4\}$ a secret at location $S(-a, -b)$.

Looking ahead, to allow *batching*, the dealer will choose m different bivariate polynomials $S_1(x, y), \ldots, S_m(x, y)$, and all the parties will verify all the m instances simultaneously. To accept the shares, all instances must end up successfully. We follow the following two design principles:

1. *Broadcast is expensive; Each broadcast must be utilized in all m instances, not just in one instance.*
2. *Detection: Whenever a party is detected as an obstacle for achieving agreement (a*

[1] Our actual parameters are further optimized to pack more secrets.

foe), we should make it a "friend", or more precisely, we neutralize its capacity to obstruct further, and utilize it to achieve agreement on a later stage.

We focus on sharing of one instance for now, while keeping these design principles in mind. Along the way, we also discuss how to keep the broadcasts of the dealer low for all m instances simultaneously, and we will show how to reduce the broadcasts of other parties later on. We follow a similar structure to that of the basic scheme:

1. **Sharing:** The dealer sends $f_i(x), g_i(y)$ to each party P_i.
2. **Pairwise checks:** Each pair of parties exchange sub-shares. In case of a mismatch, a party broadcasts a complaint $\mathsf{complaint}(i, j, f_i(j), g_i(j))$.

The dealer now has to resolve the complaints. In the basic protocol, when the dealer identifies party P_i as corrupted, the dealer simply broadcasts the "correct" $(S(x, i), S(i, y))$ so that everyone can verify that the shares are consistent. However, this leads to $\mathcal{O}(n^2)$ values being broadcasted, and $\mathcal{O}(mn^2)$ values in the batched case. Instead, in our protocol, the dealer just marks P_i as corrupted and adds it to a set $\mathsf{CONFLICTS} \subset [n]$ which is initially empty. It broadcasts the set $\mathsf{CONFLICTS}$. This set should be considered as "parties that had false complaints" from an honest dealer's perspective. There are three cases to consider:

1. The dealer is discarded: This might happen, e.g., if two parties complained on each other and none of them is in $\mathsf{CONFLICTS}$. In this case, it is clear that the dealer is corrupted, and all parties can just discard it.
2. If the dealer is not discarded and $|\mathsf{CONFLICTS}| > t/4$, then we have **large conflict**. The dealer identified a large set of conflicts (note that if the dealer is honest, then $\mathsf{CONFLICTS}$ contains only corrupted parties). Instead of publicly announcing the polynomials $f_i(x), g_i(y)$ of the identified corrupted parties, the dealer simply restarts the protocol. In the new iteration, the shares of parties in $\mathsf{CONFLICTS}$ are publicly set to 0. That is, it chooses a new random bivariate polynomial $S(x, y)$ that hides the same secrets as before, this time under the additional constraints that $S(x, i) = S(i, y) = 0$ for every $i \in \mathsf{CONFLICTS}$.
   The dealer does not broadcast the shares of parties in $\mathsf{CONFLICTS}$; all the parties know that they are 0s. When each party receives its new pair of shares $f_j(x), g_j(y)$, it also verifies that $f_j(i) = g_j(i) = 0$, and if not, it raises a complaint. Parties in $\mathsf{CONFLICTS}$ cannot raise any complaints. Furthermore, observe that the outcome of "large conflict" might occur only $\mathcal{O}(1)$ times; if the dealer tries to exclude more than t parties total, then the dealer is publicly discarded.
   When batching over m instances, we choose the shares of the set $\mathsf{CONFLICTS}$ to be 0 in all instances. Thus, the dealer uses a broadcast of $\mathcal{O}(n \log n)$ bits, i.e., the set $\mathsf{CONFLICTS}$, and by restarting the protocol it made the shares of parties in $\mathsf{CONFLICTS}$ public in all m executions. Thus, we get the same effect as broadcasting $m|\mathsf{CONFLICTS}|$ pairs of polynomials (i.e., broadcasting $\mathcal{O}(m \cdot n^2 \log n)$ bits). This follows exactly our first design principle.
3. If $|\mathsf{CONFLICTS}| \leq t/4$ then the dealer proceeds with the protocol. It has to reconstruct the f and g polynomials of all parties in $\mathsf{CONFLICTS}$.

Before we proceed, let's highlight what guarantees we have so far: when the dealer is honest, then all the parties in $\mathsf{CONFLICTS}$ are corrupted. Moreover, if in some iteration there were more than $t/4$ identified conflicts by the dealer, those corrupted parties are eliminated, and they have

shares that all parties know (i.e., 0) and are consistent with the shares of the honest parties. This turns a "foe" into a "friend", as our second design principle.

When the dealer is corrupted, then all parties that are not in CONFLICTS have shares that define a unique bivariate polynomial, and we have binding. Specifically, if the shares of two honest parties do not agree with each other, then they both complain on each other, and the dealer must include one of them in CONFLICTS. Therefore, all honest parties that are not in CONFLICTS (assuming that the dealer was not publicly discarded) hold shares that are consistent with each other. Moreover, there is one more important property: Honest parties that were excluded in previous iterations (and now their shares are 0) also hold shares that are consistent with the honest parties that are not in CONFLICTS. In particular, if we indeed proceed, then there are at most $t/4$ honest parties who do not hold shares on the polynomial. This means that there are $2t + 1 - t/4$ honest parties that have shares on the bivariate polynomial – not only do we have binding, but we also have some redundancy! This redundancy will be crucial for our next step as we show below.

However, there might still be up to $t/4$ honest parties (in CONFLICTS) that do not have shares on the correct polynomial. The rest of the protocol is devoted to reconstructing their shares. We call this phase reconstruction of the shares of honest parties in CONFLICTS. However, before proceeding to the reconstruction, we first describe how to batch over m instances.

Batching Complaints. Consider sharing m instances simultaneously with the same dealer. In the above description, we already described how the dealer's broadcast is just the set CONFLICTS, which require $\mathcal{O}(n \log n)$ bits, independent of m. However, the broadcast of other parties depends on m. Specifically:

1. A party P_i broadcasts $\mathsf{complaint}(i, j, f_i(j), g_i(j))$ when it receives a wrong share from some party P_j.
2. A party P_i broadcast complaint if the share it received do not agree with the parties that are publicly 0. Recall that in that case, the dealer must include P_i in CONFLICTS.

It suffices to complain in only one of the instances, say the one with the lexicographically smallest index. This follows our first design principle. If two parties P_i and P_j do not agree in $\ell < m$ of the instances, they will both file a joint complaint with the same minimal index. Thus, we have a joint complaint, and in order to not be discarded, the dealer must include either i or j in CONFLICTS. Thus, we still have the guarantee that if two honest parties are not in CONFLICTS then their shares must be consistent, now in *all* m executions.

Likewise, if some party P_i receives from the dealer private shares where on points of some parties that were excluded it does not receive 0s, it essentially requires to be part of CONFLICTS. Thus, there is no need to make m requests, it suffices to make just one such request.

Reconstruction of the shares of honest parties in CONFLICTS. Going back to the last step of the sharing process, each party P_j in CONFLICTS wishes to reconstruct its pair of polynomials $(f_j(x), g_j(y))$. Towards that end, each party P_k that are not in CONFLICTS send to P_j, privately, the values $(f_j(k), g_j(k))$. P_j therefore is guaranteed to receive $2t + 1 - t/4$ correct points. However, the polynomials are of degree $t + t/4$, and we need $2t + t/4$ "correct values" to eliminate t errors. This means that if the adversary introduces more than $t/2$ incorrect values, P_j does not have unique decoding. If this is the case, then P_j broadcasts a complaint $\mathsf{complaint}(j)$, insisting that its shares will be publicly reconstructed. As we will see, when batching over m executions, it is enough to make one public complaint in one execution, say the lexicographically smallest one, let's denote it as $\beta \in [m]$. Resolving this instance will help to resolve all other m instances.

Upon receiving $\mathsf{complaint}(j, \beta)$, each party P_k broadcasts $\mathsf{reveal}(k, j, f_k(j), g_k(j))$ for the βth instance. Thus, we will have at least $2t+1-t/4$ correct values that are public. Moreover, corrupted parties might now reveal values that are different than what they have previously sent privately, and we might already have unique decoding. In any case, with each value that was broadcasted and is wrong, the dealer adds the identity of the party that broadcasted the wrong value into a set Bad. It then broadcasts the set Bad, and all parties can check that when excluding parties in Bad then all other values define a unique polynomial, and all public points (excluding Bad) lie on this polynomial. Otherwise, the dealer is publicly discarded. Note that it is enough to broadcast one set Bad for all party $j \in \mathsf{CONFLICTS}$ and for all m instances. If $|\mathsf{Bad}| > t/2$, we restart the protocol, again giving shares 0 to parties in Bad (as long as the total number of parties that the dealer excluded does not exceed t).

At this point, if we did not restart and the dealer was not discarded, then it must be that P_j can reconstruct its polynomials $f_j(x), g_j(y)$ *in all* m *instances*. First, in the βth instance (that was publicly resolved), we know that we have $2t+1-t/4$ public points that are "correct" and that the dealer could have excluded at most $t/2$ parties. Therefore, there are more than $t + t/4 + 1$ correct points even if the dealer excludes up to $t/2$ honest parties (recall that it cannot exclude more than $t/2$). Those correct points uniquely determine a polynomial of degree $t+t/4$, and therefore, since all points after excluding parties in Bad lie on one unique polynomial, it must be that this polynomial is the correct one.

Using the information learned in the resolved instance party P_j can uniquely decode all other m instances. Specifically, there is no unique decoding in a particular instance only if P_j received more than $t/2$ wrong private shares. When going publicly, some parties might announce different values than what they first told P_j privately. P_j can compare between the polynomial reconstructed in the βth instance to the initial values it received privately from the parties, and identify all parties that sent it wrong shares. Denote this set as $\mathsf{localBad}_j$. It must hold that this set contains more than $t/2$ corrupted parties. Now, in each one of the other instances, ignore all parties in $\mathsf{localBad}_j$. This implies that the remaining values are of distance at most $t/2$ from a correct word, i.e., they contain at most $t/2$ errors. Moreover, it is guaranteed that honest parties are not eliminated, and we still have at least $2t + 1 - t/4$ correct points. Therefore, P_j guarantees to have unique decoding in all m instances.

Detectable and Robust Reconstruction. So far, we described the sharing procedure. While we do not use the reconstruction of detectable secret sharing directly (we will use private reconstruction, and parties never reconstruct all secrets), we briefly describe it for completeness. To reconstruct polynomials $S_1(x, y), \ldots, S_m(x, y)$ that were shared with the same dealer, we follow a similar step as reconstruction towards parties in $\mathsf{CONFLICTS}$, but with reconstructing all polynomials: Each party sends (privately) the f-shares, the parties try to privately reconstruct g_i-polynomials for all $i \in [n]$, and interpolate the bivariate polynomials from the g_i-polynomials. If some party does not succeed in uniquely reconstructing some g_i-polynomial, then it asks to go public. For each party P_j, it is enough to publicly reconstruct one g_i-polynomial that it did not succeed to reconstruct privately, and from that, P_j can reconstruct all other shares (by ignoring the new privately detected parties).

However, as before, the adversary can cause the reconstruction to fail. When it does so, the dealer is guaranteed to detect more than $t/2$ corruptions. Moreover, if the dealer already detected at least $t/2$ corruptions during the sharing phase, then those parties cannot fail the reconstruction, and reconstruction is guaranteed. Note that the cost of the reconstruction is $\mathcal{O}(mn^2 \log n)$ over point-

to-point channels, plus each party has to broadcast at most $\mathcal{O}(n \log n)$ bits, again, independent of m.

Reconstruction for VSS. Recall that for VSS, we set the degree of y in each bivariate polynomial to t. This implies that all parties can reconstruct all g-polynomials using Reed-Solomon error correction and we never have to resolve complaints publicly. Moreover, the adversary can never cause any failure. The cost is therefore $\mathcal{O}(mn^2 \log n)$ over point-to-point channels, and VSS robust reconstruction is always guaranteed.

We refer the reader to Section 4 for our packed secret sharing scheme for a single polynomial, and to Section 5 for the batched version.

2.2 Our MPC Protocol

Our MPC protocol follows the following structure: an offline phase in which the parties generate Beaver triplets [10], and an online phase in which the parties compute the circuit while consuming those triples.

Beaver triplets generation. Our goal is to distribute shares of random secret values a, b and c, such that $c = ab$. If the circuit contains C multiplication gates, then we need C such triplets. Towards that end, we follow the same steps as in [23], and generate such triplets in two stages:

1. **Triplets with a dealer:** Each party generates shares of a_i, b_i, c_i such that $c_i = a_i \cdot b_i$. We generate all the triplets in parallel using expected $\mathcal{O}(1)$ rounds. We will elaborate on this step below in Section 2.3. Our main contribution is in improving this step. In our protocol, each party acts as a dealer to generate mn triplets. This step requires an overall cost of $\mathcal{O}(n^4 \log n + mn^3 \log n)$ point-to-point communication for all the parties together. Later, these mn^2 triplets will be used for generating $\mathcal{O}(mn^2)$ triplets overall. Looking ahead, we will use $m = C/n^2$ and this step costs $\mathcal{O}(n^4 \log n + Cn \log n)$.
   Previously, the best known [23] used $\mathcal{O}(n^3 \log n)$ point-to-point and $\mathcal{O}(n^3 \log n)$ broadcast for generating just a single triplet for one dealer. That is, for $\mathcal{O}(mn^2)$ triplets this is $\mathcal{O}(mn^5 \log n)$ broadcast which costs at least $\Omega(mn^6 \log n)$ over point-to-point. We therefore improve in a factor of $\mathcal{O}(n^3)$.
2. **Triplets with no dealer:** Using triplet extraction of [23], we can extract from a total of C triplets with a dealer, $\mathcal{O}(C)$ triplets where no party knows the underlying values. That is, if n parties generate C/n triplets each, then we have a total of C triplets and we can extract from it $\mathcal{O}(C)$ triplets. This step costs $\mathcal{O}(n^2 \log n + Cn \log n)$.

Putting it all together, for generating C triplets we pay a total of $\mathcal{O}(n^4 \log n + Cn \log n)$ and constant expected number of rounds.

The MPC protocol then follows the standard structure where each party shares its input, and the parties evaluate the circuit gate-by-gate, or more exactly, layer-by-layer. In each multiplication gate, the parties have to consume one multiplication triple. Using the method of [23], if the ith layer of the circuit contains C_i multiplications (for $i \in [D]$, where D is the depth of the circuit), the evaluation costs $\mathcal{O}(n^2 \log n + C_i \cdot n \log n)$. Summing over all layers, this is $\sum_{i \in [D]} (n^2 + nC_i) \log n = (Dn^2 + Cn) \log n$. Together with the generation of the triplets, we get the claimed $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ cost as in Theorem 1.3. We refer the reader to Section 8 for further details on our MPC protocol.

2.3 Multiplication Triplets with a Dealer

As mentioned, a building block which we improve in a factor of $\mathcal{O}(n^3)$ over the state-of-the-art is multiplication triplets with a dealer. The goal is that given a dealer, to distribute shares of secret values $\vec{a}, \vec{b}, \vec{c}$ such that for every i it holds that $c_i = a_i b_i$. Towards this end, the dealer plants $\vec{a}$ into some bivariate polynomial $A(x, y)$ using our verifiable secret sharing scheme. It plants $\vec{b}$ into $B(x, y)$ and $\vec{c}$ into $C(x, y)$ in a similar manner. Note that we use verifiable secret sharing here, since we want to output the triplets shared via degree-t polynomials (which is utilized by our MPC protocol). So we can plant only $\mathcal{O}(n)$ values in each one of them. Then, the dealer has to prove, using a distributed zero-knowledge protocol, that indeed $c_i = a_i b_i$ for every i. The zero-knowledge proof uses sharing and computations on the coefficients of the polynomials used for sharing $\vec{a}, \vec{b}, \vec{c}$, i.e., if we shared $\mathcal{O}(M)$ triplets, then the zero-knowledge involves sharing of $\mathcal{O}(Mn)$ values. However, since the dealer is involved in the sharing and the reconstruction of those values, we do not need full-fledged secret sharing scheme, and we can use the lighter detectable secret sharing. This scheme enables us to share $\mathcal{O}(Mn)$ values at the same cost of "strong" verifiable secret sharing of $\mathcal{O}(M)$ values.

In a more detail, after verifiable sharing A, B and C, the dealer needs to prove that for every $a \in \{0, \ldots, t/4\}$ it holds that $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$. Towards that end, for every $a \in \{0, \ldots, t/4\}$ it considers the polynomial

$$E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y) = e_{-a,0} + e_{-a,1} y + \ldots + e_{-a,2t} y^{2t}$$

and its goal is to show that the degree-$2t$ polynomial $E_{-a}(y)$ evaluates to 0 on each $y \in \{0, \ldots, -t/4\}$. The dealer secret-shares all the coefficients $(e_{-a,k})$ for $a \in \{0, \ldots, t/4\}$ and $k \in \{0, \ldots, 2t\}$ using our detectable secret sharing scheme, by packing them into several bivariate polynomials $E(x, y)$. Note that there are $\mathcal{O}(n^2)$ coefficients to share, and each polynomial $E(x, y)$ can pack $(t/4 + 1)^2$ secrets.[2] Thus, we actually share a constant number (precisely 8) of polynomials to share all the coefficients.

Using linear combinations over the shares, the reconstruction protocol privately reconstructs towards P_j (for every $j \in [n]$) the evaluation of $E_{-a}(y)$ on j, i.e., $E_{-a}(j)$, for each $a \in \{0, \ldots, t/4\}$. This is performed in a similar manner to the reconstruction of shares of honest parties in CONFLICTS in our detectable secret sharing protocol. Each P_j can then verify that $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$, and if not, it can raise a public complaint. The parties can then open the shares of P_j on A, B, C publicly, and also the value $E_{-a}(j)$. If indeed $E_{-a}(j) \neq A(-a, j) \cdot B(-a, j) - C(-a, j)$, then the dealer is discarded.

Moreover, again using linear evaluations over the shares and reconstruction, the parties can obtain $E_{-a}(0)$ for every $a \in \{0, \ldots, t/4\}$ and verify that it equals 0. If indeed $E_{-a}(j) = A(-a, j) \cdot B(-a, j) - C(-a, j)$ for $2t + 1$ such js, then $E_{-a}(y) = A(-a, y) \cdot B(-a, y) - C(-a, y)$ as those are two polynomials of degree $2t$ that agree on $2t + 1$ points. Moreover, if indeed $E_{-a}(0) = 0$ for every $a \in \{0, \ldots, t/4\}$, then $C(-a, 0) = A(-a, 0) \cdot B(-a, 0)$ for every $a \in \{0, \ldots, t/4\}$, as required.

The above description is a bit oversimplified. Recall that the coefficients of E are shared using only detectable secret sharing. This means that the private reconstruction towards some P_j might fail. In that case, P_j will ask to perform public reconstruction, and the adversary learns $E_{-a}(j)$ on a point $j \notin I$. This is a leakage because the reconstruction was meant to be private and becomes

[2] Again, in the actual construction we will use different dimensions, but we keep using a bivariate polynomial with degree $t + t/4$ in both x and y for simplicity.

public. The good news is that the outcome of each such public reconstruction is that party P_j identifies at least $t/2$ corruptions in $\mathsf{localBad}_j$, and all the later reconstructions towards it must succeed.

As a result, the adversary may learn up to $n-t$ reconstructions that it was not supposed to learn. Whenever this occurs, we cannot use the entire polynomials that are involved (which pack $\mathcal{O}(n)$ triplets). If a "pack" of triplets requires a public reconstruction, we discard the whole "pack". On the positive side, this can happen at most once per party. Moreover, since the multiplication triplets are just random and do not involve secret inputs, we can just sacrifice them. This means that for generating m "packs" of triplets, we need to start with batching $\mathcal{O}(m+n)$ "packs" of triplets. This additional overhead does not affect the overall complexity, but it makes the functionalities and the protocol a bit more involved. We refer the reader to Sections 6 and 7 for further details.

Organization. The rest of this paper is organized as follows. After some Preliminaries (Section 3) we focus on our packed (Section 4) and batched (Section 5) secret sharing. We then discuss our packed (Section 6) and batched (Section 7) multiplication triplets with a dealer, and conclude with the MPC protocol in Section 8.

3 Preliminaries

3.1 Network Model and Definitions

We consider a synchronous network model where the parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$ are connected via pairwise private and authenticated channels. Additionally, for some of our protocols we assume the availability of a broadcast channel, which allows a party to send an identical message to all the parties. The distrust in the network is modelled as a *computationally unbounded* active adversary $\mathcal{A}$ which can maliciously corrupt up to t out of the n parties during the protocol execution and make them behave in an arbitrary manner. We prove security in the stand-alone model for a static adversary. We provide the definitions (which are standard) below. Owing to the results of [19], this guarantees adaptive security with inefficient simulation. We derive universal composability [17] using [36].

Our protocols are defined over a finite field $\mathbb{F}$ where $|\mathbb{F}| > n + t/2 + 1$. We denote the elements by $\{-t/2, -t/2+1, \ldots, 0, 1 \ldots, n\}$. We use $\langle v \rangle$ to denote the degree-t Shamir-sharing of a value v among parties in $\mathcal{P}$.

3.2 Security Definition

We prove the security of our protocols in the standard, standalone simulation-based security model of multiparty computation in the perfect settings [16, 6]. Let $f : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let π be an n-party protocol over private and authenticated point-to-point channels and an authenticated broadcast channel. Let $\mathcal{A}$ be the adversary with auxiliary input z, and let $\mathcal{C} \subset \mathcal{P}$ be the set of corrupted parties controlled by it. We define the real and ideal executions:

- **The real execution:** In the real model, the parties run the protocol π where the adversary $\mathcal{A}$ controls the parties in $\mathcal{C}$. The adversary is assumed to be rushing, meaning that in every round it can see the messages sent by the honest parties to the corrupted parties before it determines the message sent by the corrupted parties. The adversary cannot see the messages

sent between honest parties on the point-to-point channels. We denote by $\mathsf{Real}^{\pi}_{\mathcal{A}(z),\mathcal{C}}(\vec{x})$ the random variable consisting of the view of the adversary $\mathcal{A}$ in the execution (consisting of all the initial inputs of the corrupted parties, their randomness and all messages they received), together with the output of all honest parties.

- **The ideal execution:** The ideal model consists of all honest parties, a trusted party and an ideal adversary $\mathcal{SIM}$, controlling the same set of corrupted parties $\mathcal{C}$. The honest parties send their inputs to the trusted party. The ideal adversary $\mathcal{SIM}$ receives the auxiliary input z and sees the inputs of the corrupted parties. $\mathcal{SIM}$ can substitute any x_i with any x'_i of its choice (for the corrupted parties) under the condition that $|x'_i| = |x_i|$. Once the trusted party receives (possibly modified) inputs $(x'_1, ..., x'_n)$ from all parties, it computes $(y_1, ..., y_n) = f(x'_1, ..., x'_n)$ and sends y_i to P_i. The output of the ideal execution, denoted as $\mathsf{Ideal}^{f}_{\mathcal{SIM}(z),\mathcal{C}}(\vec{x})$ is the output of all honest parties and the output of the ideal adversary $\mathcal{SIM}$.

Definition 3.1. *We say that a protocol π is t-secure for a functionality f, if for every adversary $\mathcal{A}$ in the real world, there exists an adversary $\mathcal{SIM}$ in the ideal world such that for every $\mathcal{C} \subset \mathcal{P}$ of cardinality at most t, it must hold that*

$$\{\mathsf{Ideal}^{f}_{\mathcal{SIM}(z),\mathcal{C}}(\vec{x})\} \equiv \{\mathsf{Real}^{\pi}_{\mathcal{A}(z),\mathcal{C}}(\vec{x})\}$$

where $\vec{x}$ is chosen from $(\{0,1\}^*)^n$ such that $|x_1| = \ldots = |x_n|$.

Modular composition. We also consider standard f-hybrid model. In the f-hybrid model, the parties can have access to some trusted party that can compute some functionality f for them. See, e.g., [16, 6] for further details.

3.3 Bivariate Polynomials and Secret Embedding

A degree (l, m)-bivariate polynomial over $\mathbb{F}$ is of the form $S(x, y) = \sum_{i=0}^{l} \sum_{j=0}^{m} b_{ij} x^i y^j$ where $b_{ij} \in \mathbb{F}$. The polynomials $f_i(x) = S(x, i)$ and $g_i(y) = S(i, y)$ are called ith f and g univariate polynomials of $S(x, y)$ respectively. In our protocol, we use $(t + t/2, t + d)$-bivariate polynomials where $d \leq t/4$, and the ith f and g univariate polynomials are associated with party P_i for every $P_i \in \mathcal{P}$.

We view a list of $(t/2+1)(d+1)$ secrets $\mathsf{SECRETS}$ as a $(t/2+1) \times (d+1)$ matrix. We then say that the set $\mathsf{SECRETS}$ is *embedded* in a bivariate polynomial $S(x, y)$ of degree $(t + t/2)$ in x and $(t+d)$ in y if for every $a \in \{0, \ldots, t/2\}$ and $b \in \{0, \ldots, d\}$ it holds that $S(-a, -b) = \mathsf{SECRETS}(a, b)$.

3.4 Simultaneous Error Correction and Detection of Reed-Solomon Codes

We require the following coding-theory related results. Let C be an Reed-Solomon (RS) code word of length N, corresponding to a k-degree polynomial (containing $k + 1$ coefficients). Assume that at most t errors can occur in C. Let $\bar{C}$ be the word after introducing error in C in at most t positions. Let the distance between C and $\bar{C}$ be s where $s \leq t$. Then there exists an *efficient* decoding algorithm that takes $\bar{C}$ and a pair of parameters (e, e') as input, such that $e + e' \leq t$ and $N - k - 1 \geq 2e + e'$ hold and gives one of the following as output:

1. Correction: output C if $s \leq e$, i.e. the distance between C and $\bar{C}$ is at most e;

2. Detection: output "more than e errors" otherwise.

Note that detection does not return the error indices, rather it simply indicates error correction fails due to the presence of more than correctable (i.e. e) errors. The above property of RS codes is traditionally referred to as *simultaneous error correction and detection*. In fact the bounds, $e + e' \leq t$ and $N - k - 1 \geq 2e + e'$, are known to be necessary. We cite:

Theorem 3.2 ([22, 37])**.** *Let C be an Reed-Solomon (RS) code word of length N, corresponding to a k-degree polynomial (containing $k+1$ coefficients). Let $\bar{C}$ be a word of length N such that the distance between C and $\bar{C}$ is at most t. Then RS decoding can correct up to e errors in $\bar{C}$ to reconstruct C and detect the presence of up to $e + e'$ errors in $\bar{C}$ if and only if $N - k - 1 \geq 2e + e'$ and $e + e' \leq t$.*

A couple of corollaries follows from the above theorem that we will use in our work.

Corollary 3.3. *Let C and $\bar{C}$ be as in Theorem 3.2 with $N = 2t + 1 + d + t/2$ and $k = t + d$ for any $d > 0$.*

1. *Then RS decoding can correct up to $t/2$ errors in $\bar{C}$, or detect the presence of up to t errors in $\bar{C}$.*
2. *If $t' > t/2$ errors are known in code word $\bar{C}$, then the remaining $t - t'$ errors in $\bar{C}$ can be corrected from the truncated code word C' obtained by removing the t' error points from $\bar{C}$.*

Proof. The first item follows because $N-k-1 = 2t+1+d+t/2-t-d-1 = t+t/2$, $2e+e' = t+t/2$ and $e+e' = t$ hold. The second item holds because $(N-t')-k-1 = 2t+1+d+t/2-t'-t-d-1 = t + t/2 - t'$. And so $N - t' - k - 1 = t + t/2 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. □

Corollary 3.4. *Let C and $\bar{C}$ be as in Theorem 3.2 with $N = 3t + 1$ and $k = t + t/2$.*

1. *Then RS decoding can correct up to $t/2$ errors and detect the presence of up to t errors in $\bar{C}$.*
2. *If $t' > t/2$ errors are known in $\bar{C}$, then $t - t'$ errors can be corrected from the truncated codeword C' obtained from $\bar{C}$ after removing the t' error points.*

Proof. The first item follows since $N - k - 1 = t + t/2$, $2e + e' = t + t/2$ and $e + e' = t$ hold. The second item follows since $(N - t') - k - 1 = 3t + 1 - t' - t - t/2 - 1 = t + t/2 - t'$. And so $N - t' - k - 1 = t + t/2 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. □

Corollary 3.5. *Let C and $\bar{C}$ be as in Theorem 3.2 with $N = 3t + 1$ and $k = t + d$ where $d \leq t/4$.*

1. *Then RS decoding can correct up to $t/2$ errors and detect the presence of up to t errors in $\bar{C}$.*
2. *If $t' > t/2$ errors are known in $\bar{C}$, then $t - t'$ errors can be corrected from the truncated codeword C' obtained from $\bar{C}$ after removing the t' error points.*

Proof. The first item follows since $N-k-1 = 2t-d \geq t+3t/4$, $2e+e' = t+t/2$ and $e+e' = t$ hold. The second item follows since $(N-t')-k-1 = 3t+1-t'-t-d-1 = 2t-d-t' \geq t+3t/4-t'$. And so $N - t' - k - 1 \geq t + 3t/4 - t' \geq 2(t - t')$ holds true for all $t' > t/2$. □

3.5 Parallel Broadcast

In our MPC, we use parallel broadcast that relates to the case where n parties wish to broadcast a message of size L bits in parallel, as captured in the following functionality.

Functionality 3.6: $\mathcal{F}_{\mathsf{BC}}^{\mathsf{parallel}}$

The functionality is parameterized with a parameter L.

1. Each $P_i \in \mathcal{P}$ sends the functionality its message $M_i \in \{0,1\}^L$.
2. The functionality sends to all parties the message $\{M_i\}_{i\in[n]}$.

The work of [1] presents an instantiation with the following security and complexity. Also note that, when some party has smaller message than L bits, it can pad with default values to make an L bit message.

Theorem 3.7 ([1])**.** *There exists a perfectly-secure parallel broadcast with optimal resilience of $t < n/3$, which allows n parties to broadcast messages of size L bits each, at the cost of $\mathcal{O}(n^2 L)$ bits communication, plus $\mathcal{O}(n^4 \log n)$ expected communicating bits. The protocols runs in constant expected number of rounds.*

4 Packed Secret Sharing

In this section we present our secret sharing scheme. In the introduction, we mentioned that we have two variants: regular verifiable secret sharing, and a novel detectable secret sharing. The protocol presented in this section fits the two primitives, where the difference is obtained by using different parameters in the bivariate polynomial, as we will see shortly. In this section, we still do not "batch" over multiple polynomials; the dealer share just a single polynomial. In Section 5 we provide details on the batched version. The packed secret sharing protocol consists of the following building blocks:

1. The dealer chooses a bivariate polynomial $S(x,y)$ of degree $3t/2$ in x and degree $t+d$ in y, where its secret are embedded in S. We should think of d as 0 or $t/4$. Unlike presented in Section 2.1, we have two different parameters for x and y. Looking ahead, for verifiable secret sharing, we use $d = 0$. For detectable secret sharing, we can use $d \in [1, t/4]$ (packing $\mathcal{O}((d+1)n)$ secrets).
2. The dealer tries to share $S(x,y)$ using a functionality called $\mathcal{F}_{\mathsf{ShareAttempt}}$ (see Functionality 4.1). At the end of this functionality, the sharing attempt might have the following three outcomes: (a) `discard` – the dealer is discarded; (b) (`detect`, CONFLICTS) - a large set of conflicts was detected and the protocol will be restarted; (c) `proceed`, in which case all parties also receive a set CONFLICTS (of size at most $t/2 - d$) of parties that still did not receive shares. All honest parties not in CONFLICTS hold shares that define unique bivariate polynomial of the appropriate degree. See Section 4.1 for further details.
3. The goal is now to let parties in CONFLICTS to learn their shares. Since the degrees of the bivariate polynomial is not symmetric, we first reconstruct the g-share (of degree $t+d < 3t/2$), and then the f-share (of degree $3t/2$). Reconstruction of g-polynomial is described in Section 4.2. The reconstruction of f-polynomial is similar, and is discussed in Section 4.3.

We first present the different building blocks, and then in Section 4.4 we provide the protocol (and functionality) for packed secret sharing, that uses those building blocks.

4.1 Sharing Attempt

We start with the description of the functionality.

Functionality 4.1: Sharing Attempt– $\mathcal{F}_{\mathsf{ShareAttempt}}$

The functionality is parameterized with the set of corrupted parties $I \subset [n]$.

1. All the honest parties send to $\mathcal{F}_{\mathsf{ShareAttempt}}$ a set $\mathsf{ZEROS} \subset [n]$. For an honest dealer, it holds that $\mathsf{ZEROS} \subseteq I$. $\mathcal{F}_{\mathsf{ShareAttempt}}$ sends the set ZEROS to the adversary.
2. The dealer sends a polynomial $S(x, y)$ to $\mathcal{F}_{\mathsf{ShareAttempt}}$. When either the polynomial is not of degree at most $3t/2$ in x and at most $t + d$ in y, or for some $i \in \mathsf{ZEROS}$ it holds that $S(x, i) \neq 0$ or $S(i, y) \neq 0$, $\mathcal{F}_{\mathsf{ShareAttempt}}$ executes Step 4c to discard the dealer.
3. For every $i \in I$, $\mathcal{F}_{\mathsf{ShareAttempt}}$ sends $(S(x, i), S(i, x))$ to the adversary. It receives back a set $\mathsf{CONFLICTS}$ such that $\mathsf{CONFLICTS} \cap \mathsf{ZEROS} = \emptyset$.[3] If the dealer is honest, then $\mathsf{CONFLICTS} \cup \mathsf{ZEROS} \subseteq I$. If $|\mathsf{CONFLICTS} \cup \mathsf{ZEROS}| > t$ for a corrupt dealer, then $\mathcal{F}_{\mathsf{ShareAttempt}}$ executes Step 4c to discard the dealer.
4. **Output:**
   (a) Detect: If $|\mathsf{CONFLICTS}| > t/2 - d$, then send $(\texttt{detect}, \mathsf{CONFLICTS})$ to all parties.
   (b) Proceed: Otherwise, send $(\texttt{proceed}, S(x, i), S(i, y), \mathsf{CONFLICTS})$ for every $i \notin \mathsf{CONFLICTS}$ and $(\texttt{proceed}, \perp, \perp, \mathsf{CONFLICTS})$ to every $i \in \mathsf{CONFLICTS}$.
   (c) Discard: send `discard` to all parties.

Protocol 4.2: Sharing Attempt– $\Pi_{\mathsf{ShareAttempt}}$

Common input: The description of a field $\mathbb{F}$, parameter $d < t$.
Input: All parties input $\mathsf{ZEROS} \subset [n]$. The dealer inputs a polynomial $S(x, y)$ with degree $3t/2$ in x and $t + d$ in y, such that for every $i \in \mathsf{ZEROS}$ it holds that $S(x, i) = 0$ and $S(i, y) = 0$.
The protocol:

1. **(Dealing shares):** The dealer sends $(f_i(x), g_i(y)) = (S(x, i), S(i, y))$ to P_i for $i \notin \mathsf{ZEROS}$. Each P_i for $i \in \mathsf{ZEROS}$ sets $(f_i(x), g_i(y)) = (0, 0)$.
2. **(Pairwise Consistency Checks)**:
   (a) Each $i \notin \mathsf{ZEROS}$ sends $(f_i(j), g_i(j))$ to every $j \notin \mathsf{ZEROS}$. Let (f_{ji}, g_{ji}) be the values received by P_i from P_j.
   (b) Each $i \notin \mathsf{ZEROS}$ broadcasts $\mathsf{complaint}(i, j, f_i(j), g_i(j))$ if (a) $f_{ji} \neq g_i(j)$ or $g_{ji} \neq f_i(j)$ for any $j \notin \mathsf{ZEROS}$. For $j \in \mathsf{ZEROS}$, P_i broadcasts $\mathsf{complaint}(i, j, f_i(j), g_i(j))$ if $f_i(j) \neq 0$ or $g_i(j) \neq 0$.
3. **(Conflict Resolution):**
   (a) The dealer sets $\mathsf{CONFLICTS} = \emptyset$. For each $\mathsf{complaint}(i, j, u, v)$ such that $u \neq S(j, i)$ or $v \neq S(i, j)$, the dealer adds i to $\mathsf{CONFLICTS}$. The dealer broadcasts $\mathsf{CONFLICTS}$.

[3] To ease understanding and notion, we sometimes expect to receive from the adversary some sets or inputs that satisfy some conditions. We do not necessarily verify the conditions in the functionality, and this is without loss of generality. For instance, in this step we require that the adversary sends a set $\mathsf{CONFLICTS}$ such that $\mathsf{CONFLICTS} \cap \mathsf{ZEROS} = \emptyset$. Instead, we can enforce that this is the case by resetting: $\mathsf{CONFLICTS} = \mathsf{CONFLICTS} \setminus \mathsf{ZEROS}$.

(b) Discard the dealer if any one of the following does not hold: (i) $|\mathsf{ZEROS} \cap \mathsf{CONFLICTS}| = \emptyset$; (ii) $|\mathsf{CONFLICTS} \cup \mathsf{ZEROS}| \leq t$ (iii) if some P_i broadcasted $\mathsf{complaint}(i, j, u_i, v_i)$ and P_j broadcasted $\mathsf{complaint}(j, i, u_j, v_j)$ with $u_i \neq v_j$ or $v_i \neq u_j$, then CONFLICTS should contain either i or j (or both); (iv) if some P_i broadcasted $\mathsf{complaint}(i, j, u, v)$ with $j \in \mathsf{ZEROS}$ and $u \neq 0$ or $v \neq 0$, then $i \in \mathsf{CONFLICTS}$.

4. **(Output):** Each P_i outputs `discard` when the dealer is discarded and $(\texttt{detect}, \mathsf{CONFLICTS})$ when $|\mathsf{CONFLICTS}| > t/2 - d$. Else, it outputs $(\texttt{proceed}, \perp, \perp, \mathsf{CONFLICTS})$ when $i \in \mathsf{CONFLICTS}$, and $(\texttt{proceed}, f_i(x), g_i(y), \mathsf{CONFLICTS})$ otherwise.

Lemma 4.3. *Protocol 4.2,* $\Pi_{\mathsf{ShareAttempt}}$*, perfectly-securely computes Functionality 4.1,* $\mathcal{F}_{\mathsf{ShareAttempt}}$*, in the presence of a malicious adversary, controlling at most* $t < n/3$.

Proof. The efficiency of the protocol can be verified by inspection. As for security, we prove the statement separately for the case of an honest dealer and of a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z. Initialize $\mathsf{CONFLICTS} = \emptyset$.
2. Receive from the functionality the set ZEROS and a set of polynomials $(f_i(x), g_i(y))_{i \in I}$. For every $i \in I \setminus \mathsf{ZEROS}$, send $\mathcal{A}$ the pair $(f_i(x), g_i(y))$ as coming from the dealer to P_i.
3. For every $j \notin I$, and $i \notin \mathsf{ZEROS}$, simulate P_j privately sending to P_i the pair $(f_j(i), g_j(i)) = (g_i(j), f_i(j))$.
4. Receive from the adversary values $(f_{i,j}, g_{i,j})$ for every $i \in I \setminus \mathsf{ZEROS}$ and $j \notin I$. If $f_{i,j} \neq f_i(j)$ or $g_{i,j} \neq g_i(j)$, then simulate P_j broadcasting $\mathsf{complaint}(j, i, g_i(j), f_i(j))$.
5. If the adversary broadcasts $\mathsf{complaint}(i, j, u_i, v_i)$ with $u_i \neq f_i(j)$ or $v_i \neq g_i(j)$, then add i to CONFLICTS.
6. Simulate the dealer broadcasting CONFLICTS, and send CONFLICTS to the functionality.
7. Receive from the functionality the output. If $(\texttt{detect}, \mathsf{CONFLICTS})$ received, then send it to the adversary. Otherwise, for each $i \in I \setminus \mathsf{CONFLICTS}$, send $(\texttt{proceed}, f_i(x), g_i(y), \mathsf{CONFLICTS})$ and send $(\texttt{proceed}, \perp, \perp, \mathsf{CONFLICTS})$ to each $i \in \mathsf{CONFLICTS}$.

It is clear that since the protocol as well as the simulation is deterministic, the adversary's view in the real execution and ideal execution are identical. It thus remains to be shown that the output of the honest parties is the same in both these executions.

In the ideal execution, all the honest parties including the dealer hold the same set ZEROS with which they invoke the functionality. Moreover, an honest dealer always sends a valid $(3t/2, t + d)$-bivariate polynomial $S(x, y)$ such that $S(x, i) = 0$ and $S(i, y) = 0$ holds for each $i \in \mathsf{ZEROS}$. Thus, it is guaranteed that the honest parties never output `discard`. Consequently, each honest party P_i either outputs $(\texttt{detect}, \mathsf{CONFLICTS})$ or $(\texttt{proceed}, S(x, i), S(i, y), \mathsf{CONFLICTS})$. The latter holds since no honest party belongs to CONFLICTS in the case of an honest dealer.

In the real execution, since the dealer is honest, it always holds a valid $(3t/2, t + d)$-bivariate polynomial $S(x, y)$ such that $S(x, i) = 0$ and $S(i, y) = 0$ holds for each $i \in \mathsf{ZEROS}$. Moreover an honest dealer is never in conflict with another honest party and hence CONFLICTS may consist of only the corrupted parties. We first show that an honest dealer is never discarded in a real execution. To that end, observe that a dealer is discarded if and only if the following conditions hold:

1. $|\mathsf{ZEROS} \cap \mathsf{CONFLICTS}| \neq \emptyset$.
2. $|\mathsf{CONFLICTS} \cup \mathsf{ZEROS}| > t$.
3. If some P_i broadcasted $\mathsf{complaint}(i, j, u_i, v_i)$ and P_j broadcasted $\mathsf{complaint}(j, i, u_j, v_j)$ with $u_i \neq v_j$ or $v_i \neq u_j$ and $i, j \notin \mathsf{CONFLICTS}$.
4. If some P_i broadcasted $\mathsf{complaint}(i, j, u, v)$ with $j \in \mathsf{ZEROS}$ and $u \neq 0$ or $v \neq 0$ and $i \notin \mathsf{CONFLICTS}$.

It is clear than none of the above conditions hold in the case of an honest dealer, and hence the honest parties do not output `discard`. We thus have the following two cases to consider:

1. **There exists an honest party which outputs** $(\texttt{detect}, \mathsf{CONFLICTS})$ **in the real execution:** In such a case, we claim that all the honest parties output $(\texttt{detect}, \mathsf{CONFLICTS})$. Note that an honest party outputs $(\texttt{detect}, \mathsf{CONFLICTS})$ if and only if $|\mathsf{CONFLICTS}| > t/2 - d$. The set $\mathsf{CONFLICTS}$ is broadcasted by the dealer, hence all the honest parties output $(\texttt{detect}, \mathsf{CONFLICTS})$. Since the simulator emulates the interaction of the honest parties with the adversary as in the real execution of the protocol, all the simulated honest parties hold the same set $\mathsf{CONFLICTS}$ as the honest parties in the real execution. In this case, the simulator invokes the functionality with this set, which in turn sends $(\texttt{detect}, \mathsf{CONFLICTS})$ to all the honest parties in the ideal execution, which is identical to the output of the honest parties in the real world.
2. **No honest party outputs** $(\texttt{detect}, \mathsf{CONFLICTS})$ **in the real execution:** In this case, we show that each honest party outputs $(\texttt{proceed}, S(x, i), S(i, y), \mathsf{CONFLICTS})$. Observe that since the set $\mathsf{CONFLICTS}$ is broadcasted by the dealer and no honest party outputs `detect`, it must hold that $|\mathsf{CONFLICTS}| \leq t/2 - d$. Further, the polynomials $S(x, i), S(i, y)$ held by a party P_i are sent by the dealer and do not change during the protocol execution. Moreover, as mentioned, an honest party never belongs to $\mathsf{CONFLICTS}$. Hence, each honest party outputs the polynomials consistent with the dealer's polynomial $S(x, y)$. That is, each honest P_i outputs $(\texttt{proceed}, S(x, i), S(i, y), \mathsf{CONFLICTS})$. As before, since the simulator emulates the interaction of the honest parties with the adversary as in the real execution of the protocol, all the simulated honest parties hold the same set $\mathsf{CONFLICTS}$ as the honest parties in the real execution. The simulator invokes the functionality with this set, causing all the honest parties P_i to output $(\texttt{proceed}, S(x, i), S(i, y), \mathsf{CONFLICTS})$ in the ideal execution. This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the set ZEROS. Simulate running the protocol on behalf of the honest parties with ZEROS as input.
3. There are three cases to consider:
   (a) Discard: If the output of some simulated honest party P_j is `discard`, then send $S(x, y) = y^{t+d+1}$ to the functionality together with $\mathsf{CONFLICTS} = \emptyset$ (in that case, $S(x, y)$ is being rejected by the functionality and all honest parties output `discard`).
   (b) Otherwise, if some honest party output $(\texttt{detect}, \mathsf{CONFLICTS})$ then it must hold that $|\mathsf{CONFLICTS}| > t/2 - d$. Send $(S(x, y) = y^{t+d}, \mathsf{CONFLICTS})$ to the functionality . In that case, the functionality would output $(\texttt{detect}, \mathsf{CONFLICTS})$ to all parties.

(c) Otherwise, let J be an arbitrary set of $t+d+1$ honest parties that are not in CONFLICTS. Note that since $|\mathsf{CONFLICTS}| \leq t/2 - d$ we have at least $n - t/2 + d \geq 2t + 1 + t/2 + d$ parties that are not in CONFLICTS and therefore at least $3t/2 + d + 1$ honest parties not in CONFLICTS. Find the unique bivariate polynomial $S(x, y)$ in degree $3t/2$ in x and $t + d$ in y such that (a) $S(x, j) = f_j(x)$ for every $j \in J$. Send $S(x, y)$ together with CONFLICTS to the functionality.

Since the simulator emulates the honest parties as in the real execution of the protocol, the view of the adversary in the real and ideal execution is identical. It thus remains to show that the output of the honest parties in the real world and ideal world is the same. There are three cases to consider.

1. **There exists an honest party that outputs `discard` in the real world:** An honest party outputs `discard` only if verification fails at Step 3b. In this case, all the corresponding messages are broadcasted and hence all the honest parties output $\perp$. Since the real and simulated executions are identical, all the simulated honest parties also output `discard`. Thus the simulator invokes the functionality as in Step 3a of the simulation, causing all the honest parties to receive `discard` in the ideal world.
2. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{CONFLICTS})$ **in the real world:** This implies that for the set CONFLICTS broadcasted by the dealer, it holds that $|\mathsf{CONFLICTS}| > t/2 - d$. Thus, each honest party outputs $(\texttt{detect}, \mathsf{CONFLICTS})$ in the real world. Given that the simulated and real executions are identical, the simulated honest parties observe the same set CONFLICTS. The simulator in this case invokes the functionality as in Step 3b of the simulation, causing the honest parties to output $(\texttt{detect}, \mathsf{CONFLICTS})$ in the ideal world.
3. **No honest party outputs `discard` or** $(\texttt{detect}, \mathsf{CONFLICTS})$ **in the real world:** In this case, it means that $|\mathsf{CONFLICTS}| \leq t/2 - d$ and parties output `proceed` in the real execution. We want to show that all the honest parties $j \notin \mathsf{CONFLICTS}$ hold $f_i(x)$ and $g_i(y)$ consistent with a unique $(3t/2, t+d)$-bivariate polynomial $S(x, y)$. Towards that end, observe that since $|\mathsf{CONFLICTS}| \leq t/2 - d$ holds, we have that at least $3t/2 + d + 1$ honest parties are not in CONFLICTS. Let J be an arbitrary set of $t+d+1$ honest parties that are not in CONFLICTS, and reconstruct the $(3t/2, t + d)$-bivariate polynomial $S(x, y)$ from the set of degree-$(3t/2)$ univariate polynomials $(f_j(x))_{j \in J}$. We claim that the polynomials of all honest parties, not belonging to CONFLICTS lie on that polynomial.
   To show that, we first claim that the g polynomial of each honest party P_j where $j \notin \mathsf{CONFLICTS}$ agrees with S. Specifically, we have two cases here:
   - For each honest party $j \in \mathsf{ZEROS}$, it holds that $g_j(y) = S(j, y) = 0$. Specifically, for every $k \in J \cap \mathsf{ZEROS}$ it trivially holds that $g_j(k) = f_k(j) = 0$. And for every $k \in J \setminus \mathsf{ZEROS}$, it must hold that $g_j(k) = f_k(j) = 0$, as otherwise P_k would have raised a complaint and if the dealer does not include it in CONFLICTS then it would have been discarded. Since $g_j(y)$ and $S(j, y)$ are both degree-$(t + d)$ polynomials which agree in $t + d + 1$ points, $g_j(y) = S(j, y) = 0$ holds.
   - For each honest party $j \notin \mathsf{ZEROS}$, it holds that $g_j(y) = S(j, y)$. Specifically, for every $k \in J \cap \mathsf{ZEROS}$ it must hold that $g_j(k) = f_k(j)$, as otherwise P_j would have raised a complaint and thus $j \in \mathsf{CONFLICTS}$, which is a contradiction. Similarly, for every $k \in J \setminus \mathsf{ZEROS}$ it must hold that $g_j(k) = f_k(j)$, as otherwise P_j and P_k would have raised a joint complaint and thus either $j \in \mathsf{CONFLICTS}$ or $k \in \mathsf{CONFLICTS}$, which is a

contradiction. Finally, since $g_j(y)$ and $S(j,y)$ are both degree-$(t+d)$ polynomials which agree in $t+d+1$ points, $g_j(y) = S(j,y)$ holds.

Thus, for each honest $j \notin \mathsf{CONFLICTS}$, it holds that $g_j(y)$ is consistent with $S(x,y)$. Note that since $|\mathsf{CONFLICTS}| \leq t/2 - d$, we have that the g polynomial of at least $3t/2 + d + 1$ honest parties is consistent with S. We now proceed to show that the f polynomial of each honest party P_j where $j \notin \mathsf{CONFLICTS}$ agrees with S. For this, consider an arbitrary set K of $3t/2 + 1$ honest parties that are not in $\mathsf{CONFLICTS}$. Then we have the following,

- For each honest party $j \in \mathsf{ZEROS}$, it holds that $f_j(x) = S(x,j) = 0$. Specifically, for every $k \in K \cap \mathsf{ZEROS}$ it trivially holds that $f_j(k) = g_k(j) = 0$. And for every $k \in K \setminus \mathsf{ZEROS}$, it must hold that $f_j(k) = g_k(j) = 0$, as otherwise P_k would have raised a complaint and thus $k \in \mathsf{CONFLICTS}$, which is a contradiction. Since $f_j(x)$ and $S(x,j)$ are both degree-$(3t/2)$ polynomials which agree in $3t/2 + 1$ points, $f_j(x) = S(x,j) = 0$ holds.
- For each honest party $j \notin \mathsf{ZEROS} \cup \mathsf{CONFLICTS}$, it holds that $f_j(x) = S(x,j)$. Specifically, for every $k \in K \cap \mathsf{ZEROS}$ it must hold that $f_j(k) = g_k(j)$, as otherwise P_j would have raised a complaint and thus $j \in \mathsf{CONFLICTS}$, which is a contradiction. Similarly, for every $k \in J \setminus \mathsf{ZEROS}$ it must hold that $f_j(k) = g_k(j)$, as otherwise P_j and P_k would have raised a joint complaint and thus either $j \in \mathsf{CONFLICTS}$ or $k \in \mathsf{CONFLICTS}$, which is a contradiction. Finally, since $f_j(x)$ and $S(x,j)$ are both degree-$(3t/2)$ polynomials which agree in $3t/2 + 1$ points, it holds that $f_j(x) = S(x,j)$.

We conclude that if the honest parties output `proceed`, then each honest $j \notin \mathsf{CONFLICTS}$ holds $f_j(x)$ and $g_j(y)$ consistent with a unique bivariate polynomial $S(x,y)$. Moreover, since the set $\mathsf{CONFLICTS}$ is broadcasted and it holds that $|\mathsf{CONFLICTS}| \leq t/2 - d$, each honest P_j with $j \in \mathsf{CONFLICTS}$ outputs $(\texttt{proceed}, \perp, \perp, \mathsf{CONFLICTS})$. Since the simulated and real executions are identical, output of the simulated honest parties is the same as the honest parties in the real execution. Thus, in this case, the simulator invokes the functionality as in Step 3c of the simulation, which in turn ensures that the output of the honest parties is identical in the real and ideal executions.

□

4.2 Reconstruction of g-Polynomials in CONFLICTS

When invoking this functionality, we are guaranteed that the shares of the honest parties define a unique bivariate polynomial, and that the number of parties that are not in $\mathsf{CONFLICTS}$ is at least $(n - t/2) + d$. The goal of this step is to reconstruct the g-polynomials for the parties in $\mathsf{CONFLICTS}$, while the possible outcomes are: (i) the dealer is discarded; (ii) the dealer detects additional $t/2$ parties that it will make ZEROS in the next iteration; (iii) the protocol succeeds and all honest parties hold $g_j(y)$ as output.

Functionality 4.4: Reconstruction of g-Polynomials – $\mathcal{F}_{\mathsf{rec\text{-}g}}$

1. **Input:**[4] All honest parties send to the functionality $\mathcal{F}_{\mathsf{rec\text{-}g}}$ the sets $\mathsf{ZEROS} \subset [n]$ and $\mathsf{CONFLICTS} \subset [n]$, each honest $j \notin \mathsf{CONFLICTS}$ sends $(f_i(x), g_i(y))$. Let $S(x,y)$ be the

[4] If not all honest parties send shares that lie on the same bivariate polynomial, or not all send inputs that satisfy the input assumptions as described, then no security is guaranteed. This can be formalized as follows. If the input assumptions do not hold, then the functionality sends to the adversary all the inputs of all honest parties, and lets

unique bivariate polynomial of degree at most $3t/2$ in x and at most $t+d$ in y that satisfies $f_j(x) = S(x,j)$ and $g_j(y) = S(j,y)$ for every $j \notin \mathsf{CONFLICTS}$. Moreover, it holds that $n - |\mathsf{CONFLICTS}| \geq 2t + 1 + t/2 + d$.

2. $\mathcal{F}_{\mathsf{rec\text{-}g}}$ sends $(\mathsf{ZEROS}, \mathsf{CONFLICTS}, (S(x,i), S(i,y))_{i \in I})$ to the adversary. If the dealer is corrupted, then $\mathcal{F}_{\mathsf{rec\text{-}g}}$ sends $S(x,y)$ as well.
3. It receives back from the adversary a message M.
4. **Output:**
   (a) If $M = \texttt{discard}$ and the dealer is corrupted, then $\mathcal{F}_{\mathsf{rec\text{-}g}}$ sends $\texttt{discard}$ to all parties.
   (b) If $M = (\texttt{detect}, \mathsf{Bad})$ with $\mathsf{Bad} \cap (\mathsf{ZEROS} \cup \mathsf{CONFLICTS}) = \emptyset$ and $|\mathsf{Bad}| > t/2$, and with $\mathsf{Bad} \subseteq I$ in the case of an honest dealer, then $\mathcal{F}_{\mathsf{rec\text{-}g}}$ sends $(\texttt{detect}, \mathsf{Bad})$ to all parties.
   (c) If $M = \texttt{proceed}$, then $\mathcal{F}_{\mathsf{rec\text{-}g}}$ sends:
   for each $j \in \mathsf{CONFLICTS}$ the output $(\texttt{proceed}, \perp, S(j,y))$, and
   for each $j \notin \mathsf{CONFLICTS}$ send $(\texttt{proceed}, S(x,j), S(j,y))$.

Protocol 4.5: Reconstruct g-Polynomials in CONFLICTS – $\Pi_{\mathsf{rec\text{-}g}}$

Input: All parties hold the same set CONFLICTS and ZEROS. Each honest party not in CONFLICTS holds a pair of polynomials $(f_i(x), g_i(y))$, and it is guaranteed that all the shares of honest parties lie on the same bivariate polynomial $S(x,y)$ with degree at most $3t/2$ in x and $t+d$ in y.
The protocol:

1. Every party sets $\mathsf{HAVE\text{-}SHARES} = [n] \setminus (\mathsf{ZEROS} \cup \mathsf{CONFLICTS})$.
2. For every $j \in \mathsf{CONFLICTS}$:
   (a) Each party P_i for $i \in \mathsf{HAVE\text{-}SHARES}$ sends $(i, f_i(j))$ to P_j.
   (b) Let (i,u_i) be the value P_j received from P_i. Moreover, for every $i \in \mathsf{ZEROS}$, consider (i,u_i) with $u_i = 0$. Given all $(i,u_i)_{i \notin \mathsf{CONFLICTS}}$, P_j looks for a codeword of a polynomial of degree $t+d$ with a distance of at most $t/2$ from all the values it received (see Corollary 3.3, item 1). If there is such codeword, set $g_j(y)$ to be the unique Reed-Solomon reconstruction. If there is no such a unique codeword, then P_j broadcasts $\mathsf{complaint}(j)$ and every party P_i for $i \in \mathsf{HAVE\text{-}SHARES}$ broadcasts $\mathsf{reveal}(i,j,f_i(j))$.
3. The dealer sets $\mathsf{Bad} = \emptyset$. For each $\mathsf{reveal}(i,j,u)$ message broadcasted, the dealer verifies that $u = f_i(j)$. If not, then it adds i to Bad. The dealer broadcasts Bad.
4. The parties go to Step 6a if one of the following is not true: (i) $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$; (ii) $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. The parties go to Step 6b if $|\mathsf{Bad}| > t/2$.
5. Otherwise, for every $j \in \mathsf{CONFLICTS}$, if $\mathsf{complaint}(j)$ was broadcasted, then the parties consider all the points $R_j = \{(i,u_i)\}$ such that $\mathsf{reveal}(i,j,u_i)$ was broadcasted in Step 2b, and $i \in \mathsf{HAVE\text{-}SHARES} \setminus \mathsf{Bad}$, or $u_i = 0$ if $i \in \mathsf{ZEROS}$. They verify if R_j defines a unique polynomial of degree $t+d$. If not, they go to Step 6a. Otherwise, P_j sets $g_j(y)$ to be that unique polynomial.
6. **Output:**
   (a) Discard the dealer: Output $\texttt{discard}$.

---

the adversary to singlehandedly determine all outputs of all honest parties. This makes the protocol vacuously secure (since anything can be simulated).

(b) Detect: Output $(\texttt{detect}, \mathsf{Bad})$.

(c) Proceed: Each party $j \in \mathsf{CONFLICTS}$ outputs $(\texttt{proceed}, \perp, g_j(y))$. All other parties P_j with $j \notin \mathsf{CONFLICTS}$ output $(\texttt{proceed}, f_j(x), g_j(y))$.

Lemma 4.6. *Protocol 4.5, $\Pi_{\mathsf{rec\text{-}g}}$, perfectly securely computes Functionality 4.4, $\mathcal{F}_{\mathsf{rec\text{-}g}}$, in the presence of a malicious adversary, controlling at most $t < n/3$. The protocol requires the transmission of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels, and each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Proof. We prove the statement separately for the case of an honest dealer and of a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z. Set $\mathsf{HAVE\text{-}SHARES} = [n] \setminus (\mathsf{ZEROS} \cup \mathsf{CONFLICTS})$.
2. Receive from the functionality the sets $\mathsf{ZEROS}, \mathsf{CONFLICTS}$ and the polynomials $S(x,i), S(i,y)$ for every $i \in I$.
3. Set $\mathsf{Bad} = \emptyset$. For every $i \in \mathsf{CONFLICTS}$: (note that since the dealer is honest, then $I \subseteq \mathsf{CONFLICTS}$. Thus, any such element is also in I)
   (a) Simulate party P_j, $j \in \mathsf{HAVE\text{-}SHARES}$ sending $(j, i, f_j(i)) = (j, i, g_i(j))$ to the adversary.
   (b) If P_i broadcasts $\mathsf{complaint}(i)$ then simulate party P_j, for $j \in \mathsf{HAVE\text{-}SHARES}$ broadcasting $\mathsf{reveal}(j, i, g_i(j))$.
   (c) Listen to all broadcasts $\mathsf{reveal}(i', i, u_i)$ that the adversary sends for some $i' \in \mathsf{HAVE\text{-}SHARES} \cap I$. If $u_i \neq g_i(i')$, then add i' to Bad.
4. Simulate the dealer broadcasting Bad. If $|\mathsf{Bad}| > t/2$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
5. Otherwise, send $\texttt{proceed}$ to the functionality , and halt.

When the dealer is honest, we have that $\mathsf{CONFLICTS} \subseteq I$. Moreover, the protocol is deterministic, and so is the simulator. By inspection, the view of the adversary in the real and ideal executions is identical. We now show that the output of honest parties is identical in the real and ideal world.

In the real world, first note that an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| > t$.
2. $\mathsf{Bad} \not\subset \mathsf{HAVE\text{-}SHARES}$.
3. R_j does not define a unique polynomial of degree-$(t+d)$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$ and $\mathsf{CONFLICTS} \subseteq I$. Hence, it is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases:

1. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**:** In this case, it must hold that $|\mathsf{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\texttt{detect}, \mathsf{Bad})$. Since the simulator emulates the interaction of the honest parties with the dealer as in the real execution, all the simulated honest parties hold the same set Bad. In that case, the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , causing all the honest parties in the ideal world to output the same.

2. **There exists an honest party that outputs `proceed`:** This implies that $|\mathsf{Bad}| \leq t/2$. Since this set was broadcasted by the dealer, all the honest parties hold the same set. Moreover, an honest party never belongs to CONFLICTS. Thus, we have that all the honest parties P_j output $(\texttt{proceed}, f_j(x), g_j(y))$, where $f_j(x)$ and $g_j(y)$ are consistent with a bivariate polynomial $S(x,y)$ by our input assumption. Since the simulator emulates the interaction of the honest parties with the dealer as in the real execution, all the simulated honest parties hold the same set Bad. In this case, the simulator sends `proceed` to the functionality , causing all the honest parties in the ideal world to output `proceed`. This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z. Set $\mathsf{HAVE\text{-}SHARES} = [n] \setminus (\mathsf{ZEROS} \cup \mathsf{CONFLICTS})$.
2. Receive from the functionality the sets $\mathsf{ZEROS}, \mathsf{CONFLICTS}$ and the bivariate polynomial $S(x,y)$.
3. Simulate the protocol where each honest party P_j with $j \notin \mathsf{CONFLICTS}$ starts with input $S(x,j), S(j,y)$, and all parties have the same sets $\mathsf{CONFLICTS}, \mathsf{ZEROS}$.
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
   (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
   (b) If the output of some simulated honest party is $(\texttt{detect}, \mathsf{Bad})$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
   (c) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal.

We now turn to show that the outputs of all honest parties is the same in the real and in the ideal, conditioned on the view of the adversary.

1. **There exists an honest party that outputs `discard` in the real world:** An honest party outputs discard in one of the following cases:
   (a) The dealer broadcasted Bad such that (i) $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| > t$; or (ii) $\mathsf{Bad} \not\subset \mathsf{HAVE\text{-}SHARES}$. Since all honest parties hold the same sets CONFLICTS and ZEROS, and since Bad is broadcasted, we get that all honest parties would output `discard`.
   (b) The dealer broadcasted Bad with $|\mathsf{Bad}| \leq t/2$, and for which $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Moreover, some honest party $j \in \mathsf{CONFLICTS}$ broadcasted $\mathsf{reveal}(j)$, and when considering all points $R_j = \{(i, u_i)\}$ such that $\mathsf{reveal}(i, j, u_i)$ was broadcasted in Step 2b, and $i \in \mathsf{HAVE\text{-}SHARES} \setminus \mathsf{Bad}$, or $u_i = 0$ if $i \in \mathsf{ZEROS}$, it holds that R_j does not define a unique polynomial of degree $t + d$. Since the set R_j is public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

In both cases, in the ideal execution the simulator also sends `discard` to the functionality and all honest parties output `discard` in the ideal execution.

2. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**.** An honest party outputs $(\texttt{detect}, \mathsf{Bad})$ if the dealer broadcast Bad such that $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$, and $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Moreover, it holds that $|\mathsf{Bad}| > t/2$. Since the message Bad is broadcast, then all honest parties would output $(\texttt{detect}, \mathsf{Bad})$. In the simulated execution we will have that the simulator sends to the functionality the message $(\texttt{detect}, \mathsf{Bad})$, and all honest parties output $(\texttt{detect}, \mathsf{Bad})$.
3. **There exists an honest party that outputs** `proceed`**.** First, if no honest party outputs `discard` and $(\texttt{detect}, \mathsf{Bad})$, then there must be an honest party in $\mathsf{HAVE\text{-}SHARES}$ that outputs `proceed`. We now claim that all honest parties not in $\mathsf{CONFLICTS}$ output $(\texttt{proceed}, S(x,j), S(j,y))$, and all honest parties in $\mathsf{CONFLICTS}$ outputs $(\texttt{proceed}, \perp, S(j,y))$, where $S(x,y)$ is the bivariate polynomial that is interpolated by the input shares of the honest parties and is guaranteed to exist under our input assumption.
   (a) **All honest parties** P_j **with** $j \notin \mathsf{CONFLICTS}$ **output** $(\texttt{proceed}, S(x,j), S(j,y))$**.** Since an honest party did not output `discard` and $(\texttt{detect}, \mathsf{Bad})$, we have that $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$, $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$, and $|\mathsf{Bad}| \leq t/2$. Moreover, for every $j \in \mathsf{CONFLICTS}$ that broadcast $\mathsf{complaint}(j)$, the points R_j that were broadcasted (excluding the parties in Bad) define a unique polynomial. Since all the messages are broadcast, if one honest party P_k not in $\mathsf{CONFLICTS}$ output $(\texttt{proceed}, S(x,k), S(k,y))$ then all honest parties P_j not in $\mathsf{CONFLICTS}$ have the exact same public view and also output $(\texttt{proceed}, S(x,j), S(j,y))$. Note that the honest parties P_j not in $\mathsf{CONFLICTS}$ hold their respective $S(x,j)$ and $S(j,y)$ polynomials consistent with a bivariate polynomial $S(x,y)$ according to our input assumptions.
   (b) **Each honest party** P_j **with** $j \in \mathsf{CONFLICTS}$ **outputs** $(\texttt{proceed}, \perp, S(j,y))$**.** Here we have two sub-cases to consider. First, if in Step 2b the party P_j has a unique reconstruction, then the reconstruction must be $S(j,y)$. Specifically, let $g_j(y)$ be the unique reconstructed polynomial of P_j. It must hold that $S(j,k) = g_j(k)$ for at least $t+1+t/2+d$ values of k, since each honest party not in $\mathsf{CONFLICTS}$ sent to P_j a point on $S(x,y)$. Since $g_j(y)$ is of degree $t+d$, and since $S(j,y)$ is also of degree $t+d$, we must have that $g_j(y) = S(j,y)$.
   Second, if in Step 2b the party P_j did not have a unique reconstruction, then it broadcast $\mathsf{reveal}(j)$. It will receive its polynomial only in Step 5. There must be a unique reconstruction, as otherwise no honest party would have output `proceed`. We now claim that the unique reconstruction must be $S(j,y)$. As before, all honest parties broadcast values on the polynomial $S(j,y)$. The reconstruct polynomial therefore must agree with $S(j,y)$ on at least $t+1+t/2+d$ points, and since this is a polynomial of degree $t+d$, the two polynomial must be identical. We conclude that P_j outputs $(\texttt{proceed}, \perp, S(j,y))$.

To conclude, in the simulated execution the simulator would submit to the functionality the message `proceed`. Each honest party $j \in \mathsf{CONFLICTS}$ would output $(\texttt{proceed}, \perp, S(j,y))$, whereas each honest party $j \notin \mathsf{CONFLICTS}$ would output $(\texttt{proceed}, S(x,j), S(j,y))$. This is exactly as in the real execution.

□

4.3 Reconstruction of f-Polynomials in CONFLICTS

The goal of this step is to make each party in CONFLICTS to receive its f-share. This is performed in a similar manner to that of reconstruction of g. This time, all honest parties hold shares of g, and thus each party in CONFLICTS receives at least $2t+1$ correct values on each its f polynomial. The f-polynomial is of degree $3t/2$, and therefore we fail to reconstruct if the adversary introduces more than $t/2$ errors. In that case, we will have detection, in a similar manner to the reconstruction of g. The full details of the functionality (denoted by $\mathcal{F}_{\text{rec-f}}$), the protocol (denoted by $\Pi_{\text{rec-f}}$), and the proof are given below.

Functionality 4.7: Reconstruction of f-Polynomials – $\mathcal{F}_{\text{rec-f}}$

1. **Input:** All honest parties send to the functionality the sets ZEROS $\subset [n]$ and CONFLICTS $\subset [n]$, each honest party P_j for $j \notin (\text{CONFLICTS} \cup I)$ sends $(f_j(x), g_j(y))$. Each honest P_j for $j \in \text{CONFLICTS}$ sends $g_j(y)$. Let $S(x,y)$ be the unique bivariate polynomial of degree $3t/2$ in x and $t+d$ in y that satisfies $f_j(x) = S(x,j)$ and $g_j(y) = S(j,y)$ for every $j \notin \text{CONFLICTS}$ and $g_j(y) = S(j,y)$ for every $j \in \text{CONFLICTS}$. Moreover, it holds that $n - |\text{CONFLICTS}| \geq 2t + 1 + t/2 + d$.
2. Send $(\text{ZEROS}, \text{CONFLICTS}, (S(x,i), S(i,y))_{i \in I})$ to the adversary. If the dealer is corrupted, then send also $S(x,y)$.
3. Receive back from the adversary a message M.
4. **Output:**
   (a) If $M = \texttt{discard}$ and the dealer is corrupted, then send `discard` to all parties.
   (b) If $M = (\texttt{detect}, \text{Bad})$ with $\text{Bad} \cap \text{ZEROS} = \emptyset$ and $|\text{Bad}| > t/2$, then send $(\texttt{detect}, \text{Bad})$ to all parties.
   (c) If $M = \texttt{proceed}$ then send for each j the output $(\texttt{proceed}, S(x,j), S(j,y))$.

Protocol 4.8: Reconstruct f-Polynomials in CONFLICTS – $\Pi_{\text{rec-f}}$

- **Input:** All parties hold the same set CONFLICTS and ZEROS. Each honest party not in CONFLICTS holds a pair of polynomials $(f_i(x), g_i(y))$, and each honest party in CONFLICTS holds the polynomial $g_i(y)$. It is guaranteed that all the shares of honest parties lie on the same bivariate polynomial $S(x,y)$ with degree at most $3t/2$ in x and $t+d$ in y.
- **The protocol:**
  1. Set HAVE-SHARES $= [n] \setminus$ ZEROS.
  2. For every $j \in$ CONFLICTS:
     (a) Each party P_i for $i \in$ HAVE-SHARES sends $(i, g_i(j))$ to P_j.
     (b) Let (i, u_i) be the value P_j received from P_i. Moreover, for every $i \in$ ZEROS, consider (i, u_i) with $u_i = 0$. Given all (i, u_i), P_j looks for a codeword of distance at most $t/2$ from all the values it received. If there is such a codeword, set $f_j(x)$ to be the unique Reed-Solomon reconstruction (see Corollary 3.4, item 1). If there is no such a unique codeword, then P_j broadcasts complaint(j) and every party P_i for $i \in$ HAVE-SHARES broadcasts reveal$(i, j, g_i(j))$.
  3. The dealer sets Bad $= \emptyset$. For each reveal(i,j,u) message broadcasted, the dealer verifies that $u = g_i(j)$. If not, then it adds i to Bad. The dealer broadcasts Bad.

4. Verify that (i) $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$; and (ii) $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Otherwise, discard – go to Step 7a.
5. If $|\mathsf{Bad}| > t/2$ then there is a large detection – go to Step 7b.
6. Otherwise, for every $j \in \mathsf{CONFLICTS}$, if $\mathsf{complaint}(j)$ was broadcasted, then consider all the points $R_j = \{(i, u_i)\}$ such that $\mathsf{reveal}(i, j, u_i)$ was broadcasted in Step 2b, and $i \in \mathsf{HAVE\text{-}SHARES} \setminus \mathsf{Bad}$, or $u_i = 0$ if $i \in \mathsf{ZEROS}$. Verify that R_j defines a unique polynomial of degree $3t/2$. If not, go to Step 7a. Otherwise, P_j sets $f_j(x)$ to be that unique polynomial.
7. **Output:**
   (a) Discard the dealer: Output `discard`.
   (b) Detect: Output $(\texttt{detect}, \mathsf{Bad})$.
   (c) Proceed: Each party j outputs $(\texttt{proceed}, f_j(x), g_j(y))$.

Lemma 4.9. *The Protocol $\Pi_{\mathsf{rec\text{-}f}}$ (Protocol 4.8), perfectly securely computes the $\mathcal{F}_{\mathsf{rec\text{-}f}}$ functionality (Functionality 4.7), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z. Set $\mathsf{HAVE\text{-}SHARES} = [n] \setminus \mathsf{ZEROS}$.
2. Receive from the functionality the sets $\mathsf{ZEROS}, \mathsf{CONFLICTS}$ and the polynomials $S(x, i), S(i, y)$ for every $i \in I$.
3. Set $\mathsf{Bad} = \emptyset$. For every $i \in \mathsf{CONFLICTS}$: (note that since the dealer is honest, then $\mathsf{CONFLICTS} \subseteq I$. Thus, any such element is also in I)
   (a) Simulate party P_j, $j \in \mathsf{HAVE\text{-}SHARES}$ sending $(j, i, g_j(i)) = (j, i, f_i(j))$ to the adversary.
   (b) If P_i broadcasts $\mathsf{complaint}(i)$ then simulate party P_j, for $j \in \mathsf{HAVE\text{-}SHARES}$ broadcasting $\mathsf{reveal}(j, i, f_i(j))$.
   (c) Listen to all broadcasts $\mathsf{reveal}(i', i, u_i)$ that the adversary sends for some $i' \in \mathsf{HAVE\text{-}SHARES} \cap I$. If $u_i \neq f_i(i')$, then add i' to Bad.
4. Simulate the dealer broadcasting Bad. If $|\mathsf{Bad}| > t/2$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
5. Otherwise, send `proceed` to the functionality , and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| > t$.
2. $\mathsf{Bad} \not\subset \mathsf{HAVE\text{-}SHARES}$.
3. R_j does not define a unique polynomial of degree-$3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$ and $\mathsf{CONFLICTS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases:

1. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**:** In this case, it must hold that $|\mathsf{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\texttt{detect}, \mathsf{Bad})$. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad. In that case, the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party** `proceed`**:** This implies that $|\mathsf{Bad}| \leq t/2$. Since this set was broadcasted by the dealer, all the honest parties hold the same set. Moreover, an honest party never belongs to $\mathsf{CONFLICTS}$. Thus, we have that all the honest parties P_j output $(\texttt{proceed}, f_j(x), g_j(y))$ where $f_j(x)$ and $g_j(y)$ are consistent with the bivariate polynomial $S(x, y)$ as guaranteed by our input assumption. As mentioned, the simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad. In this case, the simulator sends `proceed` to the functionality , causing all the honest parties P_j in the ideal world to output $(\texttt{proceed}, S(x, j), S(j, y))$. This is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z. Set $\mathsf{HAVE\text{-}SHARES} = [n] \setminus \mathsf{ZEROS}$.
2. Receive from the functionality the sets $\mathsf{ZEROS}, \mathsf{CONFLICTS}$ and the bivariate polynomial $S(x, y)$.
3. Simulate the protocol where each honest party P_j with $j \notin \mathsf{CONFLICTS}$ starts with input $S(x, j), S(j, y)$, each honest P_j with $j \in \mathsf{CONFLICTS}$ starts with input $S(j, y)$ and all parties have the same sets $\mathsf{CONFLICTS}, \mathsf{ZEROS}$.
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
   (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
   (b) If the output of some simulated honest party is $(\texttt{detect}, \mathsf{Bad})$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
   (c) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs** `discard` **in the real world:** Observe that an honest party outputs `discard` if and only if one of the following conditions holds.
   (a) The dealer broadcasted Bad such that either (i) $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| > t$; or (ii) $\mathsf{Bad} \not\subset \mathsf{HAVE\text{-}SHARES}$. Since all honest parties hold the same sets $\mathsf{CONFLICTS}$ and ZEROS, and the set Bad is broadcasted, we get that all honest parties output `discard`.

(b) The dealer broadcasted Bad with $|\mathsf{Bad}| \leq t/2$, and for which $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Moreover, some honest party $j \in \mathsf{CONFLICTS}$ broadcasted $\mathsf{reveal}(j)$, and when considering all points $R_j = \{(i, u_i)\}$ such that $\mathsf{reveal}(i, j, u_i)$ was broadcasted in Step 2b, and $i \in \mathsf{HAVE\text{-}SHARES} \setminus \mathsf{Bad}$, or $u_i = 0$ if $i \in \mathsf{ZEROS}$, it holds that R_j does not define a unique polynomial of degree $3t/2$. Since the set R_j is public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.

2. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$ **in the real world:** In this case, it must hold that $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Moreover, it must hold that $|\mathsf{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output $(\texttt{detect}, \mathsf{Bad})$. The simulated honest parties hold an identical output, and thus the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs** `proceed` **in the real world:** In this case, we show that each honest P_j outputs $(\texttt{proceed}, S(x, j), S(j, y))$ where $S(x, y)$ is the bivariate polynomial that is interpolated from the input shares of the honest parties and is guaranteed to exist under our input assumption. We consider the following two cases.
   (a) **Each honest** P_j **with** $j \notin \mathsf{CONFLICTS}$ **outputs** $(\texttt{proceed}, S(x, j), S(j, y))$**:** Since there exists an honest party that does not output `discard` or $(\texttt{detect}, \mathsf{Bad})$, it must hold that $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$, $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$ and $|\mathsf{Bad}| \leq t/2$. Moreover, for every $j \in \mathsf{CONFLICTS}$ which broadcast $\mathsf{complaint}(j)$, the points R_j which were broadcasted (excluding the points of parties in Bad) define a unique degree-$3t/2$ polynomial. Since all the corresponding messages were broadcast, all the honest parties not in $\mathsf{CONFLICTS}$ have the same view and hence output $(\texttt{proceed}, S(x, j), S(j, y))$, where the polynomials $S(x, j)$ and $S(j, y)$ are consistent with $S(x, y)$ and held by the parties not in $\mathsf{CONFLICTS}$ according to our input assumption.
   (b) **Each honest** P_j **with** $j \in \mathsf{CONFLICTS}$ **outputs** $(\texttt{proceed}, S(x, j), S(j, y))$**:** Here we have two sub-cases to consider. First, note that if P_j has a unique reconstruction in Step 2b, then the reconstruction must be $S(x, j)$. Specifically, let $f_j(x)$ be the unique degree-$3t/2$ reconstructed polynomial of P_j. It must hold that $S(k, j) = f_j(k)$ for at least $2t + 1$ values of k, since each honest party P_k sent to P_j a point on $S(x, j)$. Since both $f_j(x)$ and $S(x, j)$ are of degree $3t/2$, we have that $f_j(x) = S(x, j)$ holds.
   Second, if in Step 2b the party P_j did not have a unique reconstruction, then it must be that P_j broadcast $\mathsf{reveal}(j)$. It will thus receive its polynomial only in Step 6. In this case, it must hold that there is a unique reconstruction of degree-$3t/2$ polynomial from the publicly revealed points in R_j which were broadcasted (excluding the points of parties in Bad), as otherwise no honest party would have output `proceed`. We now claim that the unique reconstruction must be $S(x, j)$. As before, all honest parties broadcast values on the polynomial $S(x, j)$. Moreover, since $|\mathsf{Bad}| \leq t/2$ (otherwise parties would output $(\texttt{detect}, \mathsf{Bad})$), it must hold that the reconstructed polynomial agrees with $S(x, j)$ on at least $3t/2 + 1$ points of the honest parties. Since both the reconstructed polynomial and $S(x, j)$ are of degree $3t/2$, the two polynomial must be identical. We conclude that

P_j outputs $(\texttt{proceed}, S(x,j), S(j,y))$, where it is guaranteed to hold $S(j,y)$ due to our input assumption.

Consequently, in the simulated execution the simulator sends `proceed` to the functionality. Each honest party j thus outputs $(\texttt{proceed}, S(x,j), S(j,y))$ in the ideal execution. This is exactly as in the real execution.

□

4.4 Putting Everything Together: Packed Secret Sharing

We view a list of $(t/2+1)(d+1)$ secrets $\mathsf{SECRETS}$ as a $(t/2+1) \times (d+1)$ matrix.

Functionality 4.10: Packed Secret Sharing – $\mathcal{F}_{\mathsf{PSS}}$

The functionality is parameterized by the set of corrupted parties $I \subseteq [n]$.

- **Input:** All parties input a set $\mathsf{ZEROS} \subset [n]$ such that $|\mathsf{ZEROS}| \leq t$. If the dealer is honest then it is guaranteed that $\mathsf{ZEROS} \subseteq I$.
- **Honest dealer:** The dealer sends $\mathsf{SECRETS}$ to $\mathcal{F}_{\mathsf{PSS}}$. The functionality sends ZEROS to the adversary, which replies with $(f_i(x), g_i(y))_{i \in I}$ under the constraint that $f_i(x) = g_i(y) = 0$ for every $i \in \mathsf{ZEROS}$. The functionality chooses a random bivariate polynomial $S(x,y)$ of degree $3t/2$ in x and $t+d$ in y under the constraints that (i) $\mathsf{SECRETS}$ is *embedded* in S (see Section 3 for the meaning of embedding); (ii) $S(x,i) = f_i(x)$ for every $i \in I$; (iii) $S(i,y) = g_i(y)$.
- **Corrupted dealer:** The functionality sends ZEROS to the adversary, which replies with $S(x,y)$. $\mathcal{F}_{\mathsf{PSS}}$ that verifies that $S(x,y)$ is of degree $3t/2$ in x and degree $t+d$ in y, and that for every $i \in \mathsf{ZEROS}$ it holds that $f_i(x) = g_i(y) = 0$. If not, $\mathcal{F}_{\mathsf{PSS}}$ replaces $S(x,y) = \perp$.
- **Output:** $\mathcal{F}_{\mathsf{PSS}}$ sends to each party P_j the pair of polynomials $S(x,j), S(j,y)$.

We claim that there is always a bivariate polynomial that can be reconstructed. Specifically, consider for simplicity the case where $|I| = t$:

1. A bivariate polynomial of degree $3t/2$ in x and degree $t+d$ in y is determined by $(3t/2+1)(t+d+1)$ values.
2. The adversary sends t pairs of polynomials of degree $3t/2$ and $t+d$. The f polynomials define $t(3t/2+1)$ values. Each g polynomial is already determined in t coordinates, and therefore we have a total of $t(t+d+1-t) = t(d+1)$.
3. $\mathsf{SECRETS}$ determines $(t/2+1) \cdot (d+1)$ values.

Therefore, the number of constraints that we have is $(t/2+1)(d+1) + t(3t/2+1) + t(d+1)$, which is exactly $(3t/2+1)(t+d+1)$, the total number of variables in the bivariate polynomial.

Protocol 4.11: Packed Secret Sharing in the $(\mathcal{F}_{\mathsf{ShareAttempt}}, \mathcal{F}_{\mathsf{rec\text{-}g}}, \mathcal{F}_{\mathsf{rec\text{-}f}})$-hybrid model – Π_{PSS}

Input: The dealer holds $\mathsf{SECRETS}$, and all honest parties hold the same set ZEROS.
The protocol:

1. **Dealing the shares:**
   (a) The dealer chooses a random bivariate polynomial $S(x,y)$ of degree at most $3t/2$ in x and degree $t+d$ in y that embeds $\mathsf{SECRETS}$, under the constraint that for every $i \in \mathsf{ZEROS}$ it holds that $S(x,i) = 0$ and $S(i,y) = 0$.

(b) All parties invoke Functionality 4.1, $\mathcal{F}_{\mathsf{ShareAttempt}}$, where the dealer inputs $S(x,y)$ and all parties input ZEROS:
   i. If the output is `discard`, then proceed to Step 4a.
   ii. If the output is $(\texttt{detect}, \mathsf{CONFLICTS})$ then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{CONFLICTS}$. If $|\mathsf{ZEROS}| > t$ then proceed to Step 4a. Otherwise, go back to Step 1a.
   iii. If the output is $(\texttt{proceed}, f_i(x), g_i(y), \mathsf{CONFLICTS})$, then proceed to the next step. Note that it must hold that (a) for parties $i \in \mathsf{CONFLICTS}$, $f_i(x) = g_i(y) = \perp$ and (b) $n - |\mathsf{CONFLICTS}| \geq n - (t/2 - d)$.

2. **Reconstruct the g-polynomials:** The parties invoke Functionality 4.4, $\mathcal{F}_{\mathsf{rec\text{-}g}}$, where each party P_i inputs $(\mathsf{ZEROS}, \mathsf{CONFLICTS}, f_i(x), g_i(y))$.
   (a) If the output is `discard`, then proceed to Step 4a.
   (b) If the output is $(\texttt{detect}, \mathsf{Bad})$ then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then discard and proceed to Step 4a. Otherwise, go back to Step 1a.
   (c) Otherwise, the output is $(\texttt{proceed}, f_i(x), g_i(y))$ where every party P_i with $i \in \mathsf{CONFLICTS}$ has $g_i(y) \neq \perp$, then proceed to the next step.
3. **Reconstruct the f-polynomials:** The parties invoke Functionality 4.7, $\mathcal{F}_{\mathsf{rec\text{-}f}}$, where each party P_i inputs $(\mathsf{ZEROS}, \mathsf{CONFLICTS}, f_i(x), g_i(y))$. Note that for parties in CONFLICTS it holds that $f_i(x) = \perp$.
   (a) If the output of the functionality is `discard`, then proceed to Step 4a.
   (b) If the output is $(\texttt{detect}, \mathsf{Bad})$ then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then discard and go to Step 4a. Otherwise, go back to Step 1a.
   (c) Otherwise, let $(\texttt{proceed}, f_i(x), g_i(y))$ be the output, where now all parties have $f_i(x) \neq \perp$ and $g_i(y) \neq \perp$. Go to Step 4b.
4. **Output:**
   (a) **Discard:** All parties output $\perp$.
   (b) **Successful:** Output $f_i(x), g_i(y)$.

Lemma 4.12. *Let $t < n/3$ and $d \leq t/4$. Protocol 4.11, Π_{PSS}, perfectly securely computes Functionality 4.10, $\mathcal{F}_{\mathsf{PSS}}$, in the $(\mathcal{F}_{\mathsf{ShareAttempt}}, \mathcal{F}_{\mathsf{rec\text{-}g}}, \mathcal{F}_{\mathsf{rec\text{-}f}})$-hybrid model (Functionality 4.1,4.4,4.7)), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We separate between the case of an honest dealer and a corrupted dealer.

The case of an honest dealer. The simulator is as follows:

1. Invoke $\mathcal{A}$ on an auxiliary input z.
2. Receive from the functionality the set ZEROS.
3. Set SECRETS arbitrarily as input (say, all zeros) and run the protocol where the dealer holds SECRETS and all other parties have ZEROS as input. In particular, simulate all inner functionalities as a functionality would run them.
4. Let $S(x,y)$ be the input of the simulated honest dealer used and sent to the simulated Functionality 4.1 in the last iteration (by iteration, we mean running the protocol from Step 1a until restarting or concluding the protocol). Send $S(x,i), S(i,y)$ to the functionality for every $i \in I$.

We now show that the output of the real and ideal executions are the same. Towards that end, consider the following games:

- Game_1: This is the real execution. We run the protocol where the honest dealer uses $\mathsf{SECRETS}$ as its input. The output of this experiment is the view of the adversary and the output of all honest parties in the protocol.
- Game_2: We run a modified ideal model, in which the simulator receives the same $\mathsf{SECRETS}$ as in Game_1 as an advice, and the dealer uses $\mathsf{SECRETS}$ as its input to the functionality. The simulator uses $\mathsf{SECRETS}$ as its input instead of all zeros as the description of $\mathcal{S}$. The simulator runs the protocol where the input of the honest dealer is $\mathsf{SECRETS}$, exactly as the real execution in Game_1. We claim that the dealer is never discarded. Then, the simulator sends to the functionality the output shares of the corrupted parties in the simulated execution. The functionality chooses some random polynomial $S(x, y)$ that agrees with the output shares of the adversary and $\mathsf{SECRETS}$, and gives the honest parties their shares on that polynomial. The output of this experiment is the view of the adversary as determined by the simulator, and the output of all honest parties (equivalent to $S(x, y)$).
- Game_3: This is the ideal model. In particular, the simulator receives no advice, and runs as in Game_2, but with input $\mathsf{SECRETS} = 0$.

We show that all the outputs of all games are identically distributed.

The outputs of Game_1 and Game_2 are identically distributed. The simulator in Game_2 runs the exact same protocol as the real execution in Game_1, and therefore the view of the adversary is identical in both executions. We now turn to the output of the honest parties. We claim that in that execution, the honest dealer is never discarded. Specifically:

1. The honest dealer chooses a bivariate polynomial that always satisfies the conditions of Functionality 4.1 and therefore that functionality never returns `discard`. Moreover, $\mathsf{CONFLICTS} \subseteq I$, and therefore we never reach $|\mathsf{ZEROS}| > t$ and the dealer is never discarded. Furthermore, we can reboot the protocol only a constant number of times (see Corollary 4.15).
2. When invoking Functionality 4.4 we have that the shares of the honest parties satisfy the input assumption of the functionality. Moreover, $\mathsf{Bad} \subset I$, and so again $\mathsf{Bad} \cup \mathsf{CONFLICTS} \subseteq I$, and so the dealer is not discarded. Again, we can reboot the protocol only a constant number of times (see Corollary 4.15).
3. Finally, when invoking Functionality 4.7 we have that the shares of the honest parties satisfy the input assumption of the functionality. From a similar reasons as above, the output would be shares of the reconstructed polynomial, which is the same polynomial as the dealer used in the beginning of the iteration as its input to $\mathcal{F}_{\mathsf{ShareAttempt}}$.

We conclude that when the dealer is honest, all parties output shares on the same bivariate polynomial, which is a polynomial $S(x, y)$ that the dealer used in that iteration. In Game_1, the output of all honest parties is shares on that polynomial. In Game_2, the simulator sends the shares $(S(x, i), S(i, y))_{i \in I}$ to the functionality, the functionality samples a new polynomial $S'(x, y)$ under the constraints that $S'(x, i) = S(x, i)$ and $S'(i, y) = S(i, y)$ for every $i \in I$, and $\mathsf{SECRETS}$ is embedded in $S'(x, y)$. The output of all honest parties is then equivalent to just outputting $S'(x, y)$. We claim that the output is identical via the following claim:

Claim 4.13. *Let* SECRETS *be any arbitrary sets of* $(t/2+1)(d+1)$ *field elements, and let* $I \subset [n]$ *be a set of cardinality at most* t*. Then, for every* ZEROS $\subseteq I$ *the output of the following two distributions is identical:*

Process I

- *Choose a random* $(3t/2, t+d)$*-bivariate polynomial* $S(x,y)$ *that embeds* SECRETS*, such that for every* $i \in$ ZEROS *it holds that* $S(x,i) = S(i,y) = 0$*.*
- *Output* $S(x,y)$*.*

Process II

- *Choose a random* $(3t/2, t+d)$*-bivariate polynomial* $S(x,y)$ *that embeds* SECRETS*, such that for every* $i \in$ ZEROS *it holds that* $S(x,i) = S(i,y) = 0$*.*
- *Choose a random* $(3t/2, t+d)$*-bivariate polynomial* $S'(x,y)$ *that embeds* SECRETS *such that for every* $i \in I$ *it holds that* $S'(x,i) = S(x,i)$ *and* $S'(i,y) = S(i,y)$*.*
- *Output* $S'(x,y)$*.*

Proof. For the case of $|I| = t$, we show that $S'(x,y) = S(x,y)$. Let $S(x,i) = f_i(x)$ and $S(i,y) = g_i(y)$.

We claim that there is a unique polynomial $S'(x,y)$ that can embed SECRETS and satisfies for every $i \in I$ the conditions $S'(x,i) = f_i(x)$ and $S'(i,y) = g_i(y)$. Specifically, reconstruct the polynomials $g_0(y), \ldots, g_{-t/2}(y)$ of degree $t+d$ each as follows: $g_{-a}(-b) =$ SECRETS(a,b) for $a \in \{0, \ldots, t/2\}$ and $b \in \{0, \ldots, d\}$. Moreover, for every $a \in \{0, \ldots, t/2\}$ and $i \in I$ we set $g_{-a}(i) = f_i(-a)$. This defines $t+d+1$ points on each one of the polynomials $g_0(y), \ldots, g_{-t/2}(y)$, and uniquely define polynomials of degree $t+d$. Now, from Lagrange interpolation there exists a unique bivariate polynomial $S'(x,y)$ of degree $3t/2$ in x and $t+d$ in y that satisfies $S'(a,y) = g_{-a}(y)$ for every $a \in \{0, \ldots, t/2\}$ and $S'(i,y) = g_i(y)$ for every $i \in I$. Note that those are $t/2 + t + 1$ polynomials of degree $t+d$ each, and therefore uniquely define $S'(x,y)$. This polynomial embeds SECRETS, agrees with $g_i(y)$ for every $i \in I$, and also satisfies $S'(x,i) = f_i(x)$ for every $i \in I$, since $f_i(j) = g_j(i) = S(j,i)$ for every $i,j \in I$, and $f_i(-a) = g_{-a}(i) = S(-a,i)$ for every $i \in \{0, \ldots, t/2\}$. Thus the two univariate polynomials, $S(i,y)$ and $f_i(y)$ of degree $3t/2$ must agree.

For the case of $|I| < t$, we can just view process I as first choosing polynomials $f_i(x), g_i(y)$ for every $i \in$ ZEROS such that $f_i(x), g_i(y) = 0$, and then choosing $f_i(x), g_i(y)$ for every $i \in I \setminus$ ZEROS uniformly at random under the constraint that $f_i(j) = g_j(i)$ for every $i,j \in I$. Finally, choose $S(x,y)$ uniformly at random under the constraint that $S(x,i) = f_i(x)$ and $S_i(y) = g_i(y)$ for every $i \in I$. The two process are therefore equivalent. □

Notice that Process I is equivalent to the choice of the polynomial $S(x,y)$ in Game_1. Process II is equivalent to Game_2.

The outputs of Game_2 and Game_3 are identically distributed. The only difference between the two games is that in Game_2 the simulator uses the same secret SECRETS as the honest dealer uses in the ideal execution, whereas in Game_3 the the simulator uses SECRETS $= 0$. The following claim shows that the shares that the corrupted parties receive in the simulated execution is identically distributed. In both execution, given the shares that the simulator sends to the functionality, the outputs of the honest parties are defined in exactly the same process (the functionality uses

$\mathsf{SECRETS}$ and the shares sent by the adversary). Therefore it is enough to show that the view is identically distributed.

Claim 4.14. *Let* $\mathsf{SECRETS}_1$, $\mathsf{SECRETS}_2$ *be two arbitrary sets of* $(t/2+1)(d+1)$ *field elements, and let* $I \subset [n]$ *be a set of cardinality at most* t*. Then, for every* $\mathsf{ZEROS} \subseteq I$ *the output of the following two distributions is identical:*

Process I

- *Choose a random* $(3t/2, t+d)$*-bivariate polynomial* $S_1(x,y)$ *that embeds* $\mathsf{SECRETS}_1$*, such that for every* $i \in \mathsf{ZEROS}$ *it holds that* $S_1(x,i) = S_1(i,y) = 0$*.*
- *Output* $(i, S_1(x,i), S_1(i,y))$ *for every* $i \in I$*.*

Process II

- *Choose a random* $(3t/2, t+d)$*-bivariate polynomial* $S_2(x,y)$ *that embeds* $\mathsf{SECRETS}_2$*, such that for every* $i \in \mathsf{ZEROS}$ *it holds that* $S_2(x,i) = S_2(i,y) = 0$*.*
- *Output* $(i, S_2(x,i), S_2(i,y))$ *for every* $i \in I$*.*

Proof. We show that the probability distributions $\{\{(i, S_1(x,i), S_1(i,y))\}_{i\in I}\}$ corresponding to **Process I** and $\{\{(i, S_2(x,i), S_2(i,y))\}_{i\in I}\}$ corresponding to **Process II** are identical. Towards that end, we begin by defining the probability ensembles $\mathbb{S}_2$ and $\mathbb{S}_1$ as follows:

$$\mathbb{S}_1 = \{\{(i, S_1(x,i), S_1(i,y))\}_{i\in I} | S_1 \text{ embeds } \mathsf{SECRETS}_1 \text{ and } S_1(x,i) = S_1(i,y) = 0 \ \forall i \in \mathsf{ZEROS}\}$$
$$\mathbb{S}_2 = \{\{(i, S_2(x,i), S_2(i,y))\}_{i\in I} | S_2 \text{ embeds } \mathsf{SECRETS}_2 \text{ and } S_2(x,i) = S_2(i,y) = 0 \ \forall i \in \mathsf{ZEROS}\}$$

Given this, we show that $\mathbb{S}_1 \equiv \mathbb{S}_2$. For this, we show that given any set of pairs of degree-$3t/2$ and degree-t polynomials $Z = \{f_i(x), g_i(y)\}_{i\in I}$ that satisfy $f_i(j) = g_j(i)$ for every $i, j \in I$, the number of bivariate polynomials in support of $\mathbb{S}_1$ that are consistent with Z are the same as the number of polynomials in support of $\mathbb{S}_2$.

First, observe that if there exist $i, j \in I$ such that $f_i(j) \neq g_j(i)$, or if there exists some $i \in I \cap \mathsf{ZEROS}$ such that $f_i(x) \neq 0$ or $g_i(y) \neq 0$, then there does not exist any bivariate polynomial in support of $\mathbb{S}_1$ or $\mathbb{S}_2$ that is consistent with Z.

Now consider $Z = \{f_i(x), g_i(y)\}_{i\in I}$ such that for every $i, j \in I$, it holds that $f_i(j) = g_j(i)$. Moreover, for each $i \in I \cap \mathsf{ZEROS}$, it holds that $f_i(x) = g_i(y) = 0$. We begin by counting the number of polynomials in support of $\mathbb{S}_1$ that are consistent with Z. For the case when $|I| = t$, note that Z together with $\mathsf{SECRETS}_1$ defines exactly one polynomial $S(x,y)$, as we saw in the proof of Claim 4.13.

When $|I| < t$, we can first define $g_{-a}(y)$ for each $a \in \{0, \ldots, t/2\}$ by choosing $t - |I|$ points on each of these polynomials uniformly at random (since $g_{-a}(y)$ is degree-$(t+d)$ polynomial, of which $|I|$ points are already defined by $\{f_i(-a)\}_{i\in I}$ and additionally $d+1$ points are defined by $\mathsf{SECRETS}_1(a,b)$ for each $b \in \{0, \ldots, d\}$). Following this, the number of polynomials in the support is $|\mathbb{F}|^{(t/2+1)(t-|I|)}$.

A similar argument shows the same calculation for choosing S_2 according to $\mathbb{S}_2$. Finally, since S_1 and S_2 are chosen randomly from those consistent with Z and $\mathsf{SECRETS}_1$ or $\mathsf{SECRETS}_2$ respectively, the probability that Z is obtained is same in both the cases. □

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ on the auxiliary input z.
2. Receive from the functionality the set ZEROS.
3. Simulate running the protocol with the adversary when all honest parties hold ZEROS as input, while also simulating Functionalities 4.1, 4.4, 4.7 to the adversary.

4. If the output of some simulated honest party is $\perp$, then send $S(x,y) = x^{3t/2+1}$ to the functionality, in which case it sends $\perp$ to all parties.
5. Otherwise, let J be a set of $t+d+1$ honest parties. Reconstruct the unique bivariate polynomial $S(x,y)$ that satisfies $S(x,j) = f_j(x)$ for every $j \in J$, where $f_j(x), g_j(y)$ is the output of the simulated honest party in the simulated execution. Send $S(x,y)$ to the functionality and halt.

Since the code of each party in the protocol which is not the dealer is deterministic, and the functionalities 4.1, 4.4, 4.7 are also deterministic, the view of the adversary is *identical* in the real and ideal. Moreover, since the functionality is deterministic, we can separately consider the view and the outputs of the honest parties. All is left is to show that the output of the honest parties is the same in the real and in the ideal executions. We have two cases to consider:

1. **There exists an honest party that outputs $\perp$.** That is, the shares are rejected. This happens if one of the functionalities returns `discard`, in which case all honest parties receive the same output and all honest parties output $\perp$. Or, it might be the case that the output of one of the invocations of Functionality 4.1 is $(\texttt{detect}, \mathsf{CONFLICTS})$ and it holds that $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{CONFLICTS}$ satisfies $|\mathsf{ZEROS}| > t$. Again, since the output of all parties is the same, and their view of the sets ZEROS and $\mathsf{CONFLICTS}$ is the same, all honest parties output $\perp$. In the ideal execution, the simulated honest parties would also have the exact same output. In that case, the simulator sends $x^{3t/2+1}$, and the functionality delivers $\perp$ to all honest parties. We conclude that in that case the output of the honest parties is $\perp$ both in the real and in the ideal.
2. **One honest party did not output $\perp$.** In that case, we claim that no honest party outputs $\perp$. This is similar to the previous case. Moreover, by the guarantee of functionalities 4.1, 4.4, 4.7 we have that all honest parties have shares of the same bivariate polynomial $S(x,y)$. In the ideal execution, the simulated honest parties would also have shares of that polynomial $S(x,y)$. The simulator chooses an arbitrary set J of $t+d+1$ honest parties, and reconstructs the unique bivariate polynomial that agree with their outputs. It must hold that this polynomial is $S(x,y)$. It sends this polynomial to the functionality , and each honest party receives as output the shares $S(x,i), S(i,y)$, exactly as in the real.

□

Communication and Efficiency Analysis. We conclude the following lemma, proven subsequently:

Lemma 4.15. *Let $t < n/3$ and $d \leq t/4$. There exists a protocol that implements Functionality 4.10, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}((d+1)n)$ values (i.e., $\mathcal{O}(n(d+1)\log n)$ bits) simultaneously in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Proof. By combining Theorems 4.3,4.6, 4.9, and 4.12, we conclude the existence of a protocol that securely computes Functionality 4.10 in the plain model.

Further, Protocol 4.11 requires a constant number of restarts before it terminates successfully. To see this, we analyze the case of an honest dealer and corrupt dealer separately.

In the case of an honest dealer, note that a restart may occur at Steps 1(b)ii, 2b or 3b in Protocol 4.11. In each of the above cases, due to guarantees of functionality 4.1, 4.4 and 4.7, it

is ensured that a new CONFLICTS set of cardinality more than $t/2 - d, t/2$ and $t/2$ respectively is identified. Since $d \leq t/4$, we have that more than $t/4$ conflicts are identified each time that the protocol requires a restart. Recall that in the case of the honest dealer, it is guaranteed that no honest party belongs to CONFLICTS. Thus, each time a CONFLICTS set is identified, a new set of corrupt parties is identified, which are publicly emulated (by setting their shares to 0) in the subsequent run. Consequently, after (at most) 3 restarts, it is guaranteed that more than $3t/4 \geq t/2+d$ corrupt parties are emulated publicly and hence honestly. Since the number of parties in conflict with the honest dealer can be at most t, we have that at most $t-3t/4$, that is $< t/4$ parties can misbehave in the subsequent execution of the protocol. Given that each functionality 4.1, 4.4 and 4.7 succeeds in this case, we have that Protocol 4.11 successfully terminates.

For a corrupt dealer, if the protocol terminates after (at most) 3 restarts, then by the guarantees of the protocol, we have that it computes Functionality 4.10. Otherwise, we are guaranteed that the dealer is corrupt, and hence can be discarded. The protocol requires the transmission of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels, and each party broadcasts at most $\mathcal{O}(n \log n)$ bits. □

5 Batched and Packed Secret Sharing

In this section, we suggest how to keep the broadcast unchanged when running m instances of the packed secret sharing with the same dealer. That is, if one instance requires $\mathcal{O}(n^2 \log n)$ bits communicated over point-to-point channels and each party (including the dealer) broadcasts $\mathcal{O}(n \log n)$ bits, we have a protocol that requires $\mathcal{O}(mn^2 \log n)$ bits communicated over point-to-point channels and each party still has to broadcast at most $\mathcal{O}(n \log n)$ bits (and a total of $\mathcal{O}(n^2 \log n)$). We review the changes necessary for each one of the sub-protocols of packed secret sharing.

Sharing attempt and Batched Complaints. Here the dealer inputs m bivariate polynomials, but there is *one* set ZEROS $\subset [n]$. It is assumed that all bivariate polynomials have 0 shares for the parties in ZEROS.

At Step 2b in Protocol 4.2, every P_i checks consistency in all instances but raises a complaint for only one of them, say, the minimum index of the instance. A complaint now looks like complaint$(i, j, f_i(j), g_i(j), \alpha)$ where $\alpha \in \{1, \ldots, m\}$. Moreover, if a party broadcasts complaint(i, j, u_i, v_i) for $j \in$ ZEROS, then the dealer must add P_i to CONFLICTS. Thus, there is no need for P_i to broadcast such a complaint in each instance that it sees inconsistency with P_j for $j \in$ ZEROS, but it is enough to do it in only one of the instances.

This keeps the broadcast cost $\mathcal{O}(n^2 \log n)$ bits among all m instances combined (as opposed $\mathcal{O}(mn^2 \log n)$ when running them simultaneously in a black-box manner).

Note that when the dealer is honest, honest parties never complain on one another, and this holds in all m invocations. Moreover, if the dealer is corrupted and two honest parties have to file a joint complaint, then both will have the exact same minimal index, and the dealer must have to add one of them into CONFLICTS, exactly as we have in single instance.

Batched reconstruction of g polynomials in CONFLICTS. Here the change in the protocol is more delicate than the previous case, and we provide a full modeling and proof. Specifically, In Step 2b of Protocol 4.5, a party P_j may fail to reconstruct g_j in multiple instances. However, it is enough to pick one instance β (say, the one with minimum index) and complains publicly with β. Now, rest of the public verification happens with respect to βth invocation. If parties publicly reveal values that are different than what they revealed privately, then the party knows

that those parties are corrupted and can try to reconstruct the polynomials without those shares. In particular, the only case when a party cannot uniquely reconstruct is when the the adversary introduces more than $t/2$ errors. However, if the public reconstruction of g in the βth execution is successful, it can recognize $t/2$ misbehaving parties by comparing the polynomial that was publicly reconstruct to the shares sent to it privately. Note that it is possible that a corrupted party sends some share to P_j privately but makes some other value public. P_j knows for sure that such party is corrupt, even though Bad that the dealer broadcasts can even be empty. Once P_j recognizes more than $t/2$ errors, it can eliminate them in all other private reconstructions, remaining with less than $t/2$ errors in *all* the m executions. The functionality (denoted as $\mathcal{F}_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$), the full specification of the protocol (denoted as $\Pi_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$) as well as the proof are given below.

Functionality 5.1: Batched Reconstruction of g-Polynomials – $\mathcal{F}_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$

1. **Input:** All honest parties send to the functionality the sets $\mathsf{ZEROS} \subset [n]$ and $\mathsf{CONFLICTS} \subset [n]$, each honest $j \notin \mathsf{CONFLICTS}$ sends $(f_i^\ell(x), g_i^\ell(y))_{\ell\in[m]}$. Let $S_1(x,y), \ldots, S_m(x,y)$ be the unique bivariate polynomials of degree at most $3t/2$ in x and at most $t+d$ in y that satisfy $f_j^\ell(x) = S^\ell(x,j)$ and $g_j^\ell(y) = S^\ell(j,y)$ for every $j \notin \mathsf{CONFLICTS}$ and $\ell \in [m]$. Moreover, it holds that $n - |\mathsf{CONFLICTS}| \geq 2t + 1 + t/2 + d$.
2. Send $(\mathsf{ZEROS}, \mathsf{CONFLICTS}, (S^\ell(x,i), S^\ell(i,y))_{i\in I,\ell\in[m]})$ to the adversary. If the dealer is corrupted, then send also $(S^\ell(x,y))_{\ell\in[m]}$ to the adversary.
3. Receive back from the adversary a message M.
4. **Output:**
   (a) If $M = \texttt{discard}$ and the dealer is corrupted, then send $\texttt{discard}$ to all parties.
   (b) If $M = (\texttt{detect}, \mathsf{Bad})$ with $\mathsf{Bad} \cap (\mathsf{ZEROS} \cup \mathsf{CONFLICTS}) = \emptyset$ and $|\mathsf{Bad}| > t/2$, and in case of an honest dealer, then $\mathsf{Bad} \subseteq I$, then send $(\texttt{detect}, \mathsf{Bad})$ to all parties.
   (c) If $M = \texttt{proceed}$ then send:
   for each $j \in \mathsf{CONFLICTS}$ the output $(\texttt{proceed}, \perp, (S^\ell(j,y))_{\ell\in[m]})$
   for each $j \notin \mathsf{CONFLICTS}$ send $(\texttt{proceed}, (S^\ell(x,j))_{\ell\in[m]}, (S^\ell(j,y))_{\ell\in[m]})$.

Protocol 5.2: Batched Reconstruction of g-Polynomials in $\mathsf{CONFLICTS}$ – $\Pi_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$

Input: as in the functionality.
The protocol:

1. Set $\mathsf{HAVE\text{-}SHARES} = [n] \setminus (\mathsf{ZEROS} \cup \mathsf{CONFLICTS})$, and $\mathsf{localBad}_i = \emptyset$.
2. For every $j \in \mathsf{CONFLICTS}$:
   (a) Each party P_i for $i \in \mathsf{HAVE\text{-}SHARES}$ sends $(i, j, (f_i^\ell(j))_{\ell\in[m]})$ to P_j.
   (b) Let $(i, (u_i^\ell)_{\ell\in[m]})$ be the values P_j received from P_i. Moreover, for every $i \in \mathsf{ZEROS}$, consider (i, u_i^ℓ) with $u_i^\ell = 0$. For every $\ell \in [m]$, given all $(i, u_i^\ell)_{i\notin\mathsf{CONFLICTS}}$, P_j looks for a codeword of a polynomial of degree $t+d$ with a distance of at most $t/2$ from all the values it received (see Corollary 3.3, item 1). If there is such a codeword, set $g_j^\ell(y)$ to be the unique Reed Solomon reconstruction.
   (c) If there is no such a unique codeword for some ℓ, then P_j broadcasts $\mathsf{complaint}(j, \beta)$, where β is the minimal index in $[m]$ where reconstruction of $g_\ell(y)$ failed.

(d) Every party P_i for $i \in \mathsf{HAVE\text{-}SHARES}$ broadcasts $\mathsf{reveal}(i, j, f_i^\beta(j), \beta)$.

3. The dealer sets $\mathsf{Bad} = \emptyset$. For each $\mathsf{reveal}(i, j, u, \ell)$ message broadcasted, the dealer verifies that $u = f_i^\ell(j) = S^\ell(j, i)$. If not, then it adds i to Bad. The dealer broadcasts Bad.
4. Verify that (i) $|\mathsf{ZEROS} \cup \mathsf{CONFLICTS} \cup \mathsf{Bad}| \leq t$; and (ii) $\mathsf{Bad} \subset \mathsf{HAVE\text{-}SHARES}$. Otherwise, discard – go to Step 8a.
5. If $|\mathsf{Bad}| > t/2$ then there is a large detection – go to Step 8b.
6. Otherwise, for every $j \in \mathsf{CONFLICTS}$, if $\mathsf{complaint}(j, \beta)$ was broadcasted, then consider all the points $R_j = \{(i, u_i^\beta)\}$ such that $\mathsf{reveal}(i, j, u_i^\beta)$ was broadcasted in Step 2c, and $i \in \mathsf{HAVE\text{-}SHARES} \setminus \mathsf{Bad}$, or $u_i^\beta = 0$ if $i \in \mathsf{ZEROS}$. Verify that R_j defines a unique polynomial of degree $t + d$. If not, discard – go to Step 8a.
7. If the dealer is not publicly discarded, then each P_j sets $g_j^\ell(y)$ to be the unique bivariate decoding of the points (i, u_i^β) as specified in the previous step. It defines the set $\mathsf{localBad}_j$ as follows: $\mathsf{localBad}_j = \{i \in [n] \mid g_j^\beta(i) \neq u_i^\beta$ where u_i^β was received in Step 2b$\}$, i.e., it detects all the parties that provided it wrong values. Then, for every $\ell \in [m]$ it reconstructs (see Corollary 3.3, item 2) the unique polynomial $g_j^\ell(y)$ from the points $(u_i^\ell)_{i \notin \mathsf{localBad}_j}$.
8. **Output:**
   (a) Discard the dealer: Output `discard`.
   (b) Detect: Output $(\texttt{detect}, \mathsf{Bad})$.
   (c) Proceed: Each party $j \in \mathsf{CONFLICTS}$ outputs $(\texttt{proceed}, \perp, (g_j^\ell(y))_{\ell \in [m]})$. All other parties $j \notin \mathsf{CONFLICTS}$ output $(\texttt{proceed}, (f_j^\ell(x), g_j^\ell(y))_{\ell \in [m]})$.

Lemma 5.3. *Protocol $\Pi_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$ (Protocol 5.2), perfectly-securely computes $\mathcal{F}_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$ (Functionality 5.1) in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. The analysis is a direct generalization of that of Theorem 4.6 and we omit the description of the simulator, where the only difference is running it multiple polynomials and not just one. We just highlight the non-trivial changes in the proof.

As previously, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$, and the dealer is never discarded. If there exists an honest party that outputs $(\texttt{detect}, \mathsf{Bad})$ then all honest party output $(\texttt{detect}, \mathsf{Bad})$. If there exists an honest party that outputs `proceed` then all honest parties have their g polynomials. Specifically, consider an honest party in $\mathsf{CONFLICTS}$. It receives $t+1+t/2+d$ correct points on each polynomial. Since the output of this iteration is not `detect`, it must hold that the dealer sent a set Bad with $|\mathsf{Bad}| < t/2$. However, we claim that $|\mathsf{localBad}_j| \geq t/2$. If P_j did not succeed to reconstruct the βth instance, then the adversary must have introduced at least $t/2$ errors in the private points sent to P_j. Moreover, if $|\mathsf{Bad}| < t/2$, then it must hold that corrupted parties provided public points reveal that are not the same as they sent privately to P_j. Thus, P_j must identify at least $t/2$ parties in $\mathsf{localBad}_j$, and they must all be corrupted parties. Once P_j identifies $t/2$ corrupted parties, and the number of corrupted parties is bounded by t, the number of possible errors introduced in each one of the polynomials $g_\ell(y)$ for $\ell \in [m]$ is $< t/2$. P_j therefore succeeds to find unique decoding for all those polynomials.

The case of a corrupted dealer. The simulator is similar to that in the proof of Theorem 4.6. Again, if some honest party P_j with $j \notin \mathsf{CONFLICTS}$ outputs $(\texttt{proceed}, (S^\ell(x, j))_{\ell \in [m]}, (S^\ell(j, y))_{\ell \in [m]})$ then all honest parties output `proceed`. Moreover, honest parties not in $\mathsf{CONFLICTS}$ output both

f and g's shares, and parties in $\mathsf{CONFLICTS}$ output the g's shares. This is because all parties not in $\mathsf{CONFLICTS}$ decide what to output according to the public view. As for the parties in $\mathsf{CONFLICTS}$, we have a similar argument to that in the case of an honest dealer: once $|\mathsf{Bad}| < t/2$ it must hold that $|\mathsf{localBad}_j| \geq t/2$, and given that each party have $t + 1 + t/2 + d$ correct points on each polynomial, it can always recover the underlying polynomial. □

Batched reconstruction of f-polynomials in $\mathsf{CONFLICTS}$. This follows the exact same lines as the reconstruction of g polynomials. Specifically, if the local reconstruction is not unique, then it is enough to pick one instance $\gamma \in [m]$ and open it publicly. The public verification happens with respect to the γth instance. P_j will then be able to reconstruct f_j^ℓ for every $\ell \in [m]$.

5.1 Sharing

To conclude, we realize the following functionality putting together the batched version of protocols for the sharing attempt, reconstruction of g and f polynomials. Referring the protocol as $\Pi_{\mathsf{PSS}}^{\mathsf{batched}}$, we culminate at the following theorem.

Functionality 5.4: Batched and Packed Secret Sharing – $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$

The functionality is parameterized by the set of corrupted parties $I \subseteq [n]$.

- **Input:** All parties input a set $\mathsf{ZEROS} \subset [n]$ such that $|\mathsf{ZEROS}| \leq t$. If the dealer is honest then it is guaranteed that $\mathsf{ZEROS} \subseteq I$.
- **Honest dealer:** The dealer sends $(\mathsf{SECRETS}_\ell)_{\ell \in [m]}$ to $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$. The functionality sends ZEROS to the adversary, who sends back $(f_i^\ell(x), g_i^\ell(y))_{i \in I, \ell \in [m]}$ such that $f_i^\ell(k) = g_k^\ell(i)$ for every $i, k \in I$ and $\ell \in [m]$. Moreover, for every $i \in \mathsf{ZEROS}$, $f_i(x) = g_i(y) = 0$. For every $\ell \in [m]$, the functionality chooses a random bivariate polynomial $S^\ell(x, y)$ of degree $3t/2$ in x and $t+d$ in y under the constraints that (i) $\mathsf{SECRETS}_\ell$ is embedded in S_ℓ; (ii) $S^\ell(x, i) = f_i^\ell(x)$ for every $i \in I$; (iii) $S^\ell(i, y) = g_i^\ell(y)$.
- **Corrupted dealer:** For every $\ell \in [m]$, the dealer sends $S^\ell(x, y)$ to $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ that verifies that $S^\ell(x, y)$ is of degree $3t/2$ in x and degree $t + d$ in y, and for every $i \in \mathsf{ZEROS}$ it holds that $f_i(x) = g_i(y) = 0$. If not, $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ replaces $S^\ell(x, y) = \perp$.
- **Output:** $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ sends to each party P_j the polynomials $(S^\ell(x, j), S^\ell(j, y))_{\ell \in [m]}$.

Theorem 5.5. *$\Pi_{\mathsf{PSS}}^{\mathsf{batched}}$ securely computes $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ (Functionality 5.4). It requires a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over-point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}((d+1)mn)$ values (i.e., $\mathcal{O}((d+1)mn \log n)$ bits) simultaneously in $\mathcal{O}(1)$ rounds. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

5.2 Reconstruction

We present the reconstruction protocols for our batched and packed secret sharing. As mentioned in the introduction, for our detectable secret sharing, we get a detectable reconstruction, a weaker form of robust reconstruction. For the case of $d = 0$, we get robust reconstruction, and so verifiable secret sharing. We start with fully specifying the functionality.

Functionality 5.6: Detectable Reconstruction for Batched and Packed Secret Sharing – $\mathcal{F}^{\mathsf{batched}}_{\mathsf{PSS\text{-}Rec}}$

The functionality is parameterized with the set of corrupted parties $I \subset [n]$.

1. **Input:** All honest parties send $\mathsf{ZEROS} \subset [n]$. When the dealer is honest, $\mathsf{ZEROS} \subseteq I$. Each honest party P_j sends $(f_j^k(x), g_j^k(y))$ for each $k \in [m]$ and $j \notin I$. For each k, let $S^k(x,y)$ be the unique bivariate polynomial of degree $3t/2$ in x and $t+d$ in y that satisfies $f_j^k(x) = S^k(x,j)$ and $g_j^k(y) = S^k(j,y)$ for every $j \notin I$.
2. Send ZEROS and $S^1(x,y), \ldots, S^m(x,y)$ to the adversary. If $d = 0$ then go to Step 4c.
3. Receive back from the adversary a message M.
4. **Output:**
   (a) If $M = \texttt{discard}$ and the dealer is corrupted, then send $\texttt{discard}$ to all parties.
   (b) If $M = (\texttt{detect}, \mathsf{Bad})$ with $|\mathsf{Bad}| > t/2$ and $\mathsf{Bad} \cap \mathsf{ZEROS} = \emptyset$, and in case of an honest dealer $\mathsf{Bad} \subseteq I$, then send $(\texttt{detect}, \mathsf{Bad})$ to all parties.
   (c) If $M = \texttt{proceed}$ then send to each j the output $(\texttt{proceed}, S^1(x,y), \ldots, S^m(x,y))$.

Note that if the dealer is honest then $\texttt{discard}$ cannot occur. Moreover, if the dealer is honest and $|\mathsf{ZEROS}| > t/2$, the $(\texttt{detect}, \mathsf{Bad})$ cannot occur, as $|\mathsf{Bad} \cup \mathsf{ZEROS}| \leq t$ and so we cannot have $|\mathsf{Bad}| > t/2$. In that case, we always succeed to reconstruct. On the other hand, if the dealer is honest and $|\mathsf{ZEROS}| \leq t/2$, the adversary might cause to a failure. In that case, we are guaranteed to have a mass detection.

The protocol. To reconstruct shared polynomials $S^1(x,y), \ldots, S^m(x,y)$, the reconstruction protocol follows a similar structure of that of Protocol 5.2:

1. Each party P_i holds $(f_i^\ell, g_i^\ell(y))_{\ell \in [m]}$ and a set $\mathsf{ZEROS} \subset [n]$.
2. Each party now sends all its polynomials $f_i^1(x), \ldots, f_i^m(x)$ over the private channel to all other parties.
3. The parties try to reconstruct polynomials $g_1^\ell(y), \ldots, g_n^\ell(y)$ using the polynomials $f_1^\ell(x), \ldots, f_n^\ell(x)$ (and taking 0 for the parties in ZEROS). E.g., reconstruct $g_j^\ell(y)$ by considering $(k, f_k^\ell(j))_{k \notin \mathsf{ZEROS}}$ and adding $(k, 0)$ for $k \in \mathsf{ZEROS}$. Try to correct at most $t/2$ errors, for every $\ell \in [m]$ (see Corollary 3.5, item 1). If some party fails to decode some polynomial $g_j^\ell(y)$, then it broadcast $\mathsf{complaint}(j, \ell)$. Note that it is enough to broadcast just a single complaint, say the one with the lexicographically smallest j, ℓ.
4. We will have a public reconstruction of $g_j^\ell(y)$: Each party broadcasts its point on that polynomial, and the dealer broadcasts a set Bad if there are any wrong values broadcasted. The parties output $(\texttt{detect}, \mathsf{Bad})$ if $|\mathsf{Bad}| > t/2$. The parties check that when excluding all points in Bad then all points lie on a single polynomial $g_j^\ell(y)$.
5. Using the public reconstruction, the party P_j can now locate $t/2$ corruptions and reconstruct (see Corollary 3.5, item 2) all polynomials $g_1^\ell(y), \ldots, g_n^\ell(y)$ for every $\ell \in [m]$. All parties can now find unique bivariate polynomials $S^\ell(x,y)$ satisfying $S^\ell(i,y) = g_i^\ell(y)$ for every $i \in [n]$. The parties output those polynomials.

There are few properties that we would like to highlight with respect to the above protocol:

1. Note that when $d = 0$, then we can simply run Reed-Solomon decoding in Step 3 and always succeed to reconstruct as Reed Solomon decoding returns unique decoding when there are at most t errors. Thus, there is no need for public resolution.

2. There are at most n complaints, which lead to each party broadcasting at most $\mathcal{O}(n \log n)$ bits to resolve all complaints.

Conclusion: Detectable Secret Sharing. While we provide functionality-based modeling and proofs, the verifiable secret sharing literature is also full of property based definitions, and some readers might find such modeling helpful. We provide here such properties for completeness. From combining Functionalities 5.4 and 5.6, when using $d > 0$ we obtain a two-phase protocol for parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ where a distinguished dealer $P^* \in \mathcal{P}$ holds initial $\mathsf{SECRETS}$, and all honest parties hold the same set $\mathsf{ZEROS}_{P^*} \subseteq [n]$ (where no honest party is in ZEROS_{P^*} if P^* is honest) such that the following properties hold:

- **Secrecy**: If the dealer is honest during the first phase (the sharing phase), then at the end of this phase, the joint view of the malicious parties is independent of the dealer's input $\mathsf{SECRETS}$.
- **Reconstruction or detection – corrupted dealer:** At the end of the sharing phase, the joint view of the honest parties define values $\mathsf{SECRETS}'$ such that at the end of the reconstruction phase – all honest parties will output either $\mathsf{SECRETS}'$, or discard the dealer, or $t/2$ new values will be added to ZEROS_{P^*}.
- **Reconstruction or detection – honest dealer:** At the end of the sharing phase, the joint view of the honest parties define values $\mathsf{SECRETS}' = \mathsf{SECRETS}$ that the dealer used as input for the sharing phase. At the end of the reconstruction phase, all honest parties will output $\mathsf{SECRETS}$, or $t/2$ new indices, all of corrupted parties, will be added to ZEROS_{P^*}. If ZEROS_{P^*} initially contained more than $t/2$ values during the sharing phase, then the output of the second phase is always $\mathsf{SECRETS}$.

When $|\mathsf{SECRETS}| \in \Omega(n^2)$, the protocol uses $\mathcal{O}(n^4 \log n + |\mathsf{SECRETS}| \log n)$ communication complexity for both sharing and reconstruction.

Conclusion: Verifiable Secret Sharing. From combining Functionalities 5.6 and 5.6, when using $d = 0$ we obtain a verifiable secret sharing: A two-phase protocol for parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ where a distinguished dealer $P^* \in \mathcal{P}$ holds initial secrets $s_1, \ldots, s_t$ is a Verifiable Secret Sharing Protocol tolerating t malicious parties and the following conditions hold for any adversary controlling at most t parties:

- **Validity:** Each honest party P_i outputs the values $s_{i,1}, \ldots, s_{i,t}$ at the end of the second phase (the reconstruction phase). Furthermore, if the dealer is honest then $(s_{i,1}, \ldots, s_{i,t}) = (s_1, \ldots, s_t)$.
- **Secrecy:** If the dealer is honest during the first phase (the sharing phase) then at the end of this phase, the joint view of the malicious parties is independent of the dealer's input $s_1, \ldots, s_t$.
- **Reconstruction:** At the end of the sharing phase, the joint view of the honest parties defines values $s'_1, \ldots, s'_t$ such that all honest parties will output $s'_1, \ldots, s'_t$ at the end of the reconstruction phase.

When $|\mathsf{SECRETS}| \in \Omega(n)$, the protocol uses $\mathcal{O}(n^4 \log n + |\mathsf{SECRETS}| \cdot n \log n)$ communication complexity for both sharing and reconstruction.

6 Packed Verifiable Triple Sharing

Packed verifiable triple sharing (VTS) allows a dealer to verifiably share $t/2+1$ multiplication triples at the cost of incurring $\mathcal{O}(n^2)$ elements of communication over point-to-point channels as well as broadcast. Precisely, VTS outputs each element of the triples to be Shamir-shared via a degree-t polynomial. In the next section, we will present the batched version, where $\mathcal{O}(mn)$ shared triplets are prepared with $\mathcal{O}(mn^2)$ elements of communication over point-to-point channels and the same broadcast as needed for one instance (i.e. $\mathcal{O}(n^2)$). This is an important contribution of this work that utilizes our both verifiable secret sharing and detectable secret sharing constructions.

Ideally, in a VTS, for an honest dealer, the protocol guarantees privacy of the triples, while for a corrupt dealer, it ensures correctness of the triples (i.e. they satisfy product relation) if the dealer is not discarded. However, here we construct only a weak version where it is possible for an adversary to learn the triples even when the dealer is honest. Interestingly, we show this is sufficient for our purpose, for there is a way to overcome this shortcoming in the batched version, where we show how to bound the total number of compromised instances to at most n. Furthermore, the identities of the compromised instances will be publicly known. Discarding these instances, we will still have enough 'safe' instances that satisfy the desired qualities, when m is sufficiently large.

6.1 The High-Level Idea

In order to explain VTS, we require to break open the bivariate polynomial shared through the packed secret sharing and interpret the secrets shared through univariate polynomials. Recall the way the secret-matrix $\mathsf{SECRETS}$ is planted in the bivariate polynomial $S(x,y)$: $S(-a,-b) = \mathsf{SECRETS}(a,b)$ for every $a \in \{0,\ldots,t/2\}$ and $b \in \{0,\ldots,d\}$. At the end of PSS, every party P_i holds $f_i(x), g_i(y)$.

1. *Sharing via degree-t polynomial aka. Shamir-sharing:* First assume that $d=0$ which is the case for the verifiable PSS: here $\mathsf{SECRETS}$ is a $(t/2+1)$-length vector and we can treat that the ℓth secret in $\mathsf{SECRETS}$ is Shamir-shared via the degree-t polynomial $g_{-\ell}(y) = S(-\ell, y)$, for $\ell = \{0,\ldots,t/2\}$. Note that every party P_i holds a share on $g_{-\ell}(y)$ which is $f_i(-\ell)$.
2. *Sharing via degree-3t/2 polynomial:* Now assume that $d = t/4$ which is the case for the detectable PSS: here $\mathsf{SECRETS}$ is a $(t/2+1) \times (t/4+1)$ matrix and we can treat the degree-$3t/2$ polynomial $f_{-\ell}(x) = S(x,-\ell)$ as the packed-sharing of the ℓth column of $\mathsf{SECRETS}$, for $\ell = \{0,\ldots,t/4\}$. Note that every party P_i holds a share on $f_{-\ell}(x)$ which is $g_i(-\ell)$.

We are now ready to explain the high-level idea of the packed VTS. The goal here is to generate Shamir-sharing of three vectors of secrets $\mathsf{SECRETS}_a = \{a_0,\ldots,a_{t/2}\}, \mathsf{SECRETS}_b = \{b_0,\ldots,b_{t/2}\}, \mathsf{SECRETS}_c = \{c_0,\ldots,c_{t/2}\}$, each of size $t/2+1$ such that $c_i = a_i b_i$ holds for each $i \in \{0,\ldots,t/2\}$. Our packed VTS consists of two phases, of which the latter is the core contribution here.

- Share degree-$(3t/2, t)$ bivariate polynomials A, B, C which embed $t/2+1$ multiplication triples via verifiable PSS (Functionality 4.10) with $d=0$. Following this, every party P_i holds $A(x,i), A(i,y), B(x,i), B(i,y), C(x,i), C(i,y)$. The secrets for the triplets appear at $A(-\ell,0), B(-\ell,0)$ and $C(-\ell,0)$ for every $\ell \in \{0,\ldots,t/2\}$ and are therefore t-shared via $A(-\ell,y), B(-\ell,y)$ and $C(-\ell,y)$ (see Item 1 above).

- A proof of product relation for the embedded secrets. The dealer needs to prove that *the product of the secrets Shamir shared via $A(-\ell, y)$ and $B(-\ell, y)$ is the secret Shamir shared via $C(-\ell, y)$*. The constant term of the product polynomial $A(-\ell, y)B(-\ell, y)$ must match with the constant term of $C(-\ell, y)$. That is, the degree $2t$ polynomial $A(-\ell, y)B(-\ell, y) - C(-\ell, y)$ must have zero in its free term. This is the check we want to conduct.
  Note that it is possible that this proof leaks an honest dealer's secrets, in which case the parties will kill (discard) this instance. But if it does not, then it is perfect proof in zero knowledge.

$$
\begin{array}{lccccccc}
E_0(y) = & e_{(0,0)} & e_{(0,1)}y & \cdots & e_{(0,i)}y^i & \cdots & e_{(0,2t)}y^{2t} \\
E_{-1}(y) = & e_{(-1,0)} & e_{(-1,1)}y & \cdots & e_{(-1,i)}y^i & \cdots & e_{(-1,2t)}y^{2t} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
E_{-t/2}(y) = & e_{(-t/2,0)} & e_{(-t/2,1)}y & \cdots & e_{(-t/2,i)}y^i & \cdots & e_{(-t/2,2t)}y^{2t}
\end{array}
$$

(a) The product polynomials $E_{-\ell}(y)$ for $\ell \in \{0, \ldots, t/2\}$

$$
\begin{array}{cccccc}
e_{(0,0)} & \cdots & e_{(0,t/4)} & \cdots & e_{(0,i)} & \cdots & e_{(0,2t)} \\
e_{(-1,0)} & \cdots & e_{(-1,t/4)} & \cdots & e_{(-1,i)} & \cdots & e_{(-1,2t)} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
e_{(-t/2,0)} & \cdots & e_{(-t/2,t/4)} & \cdots & e_{(-t/2,i)} & \cdots & e_{(-t/2,2t)}
\end{array}
$$

$\mathsf{SECRETS}_1$ $\quad$ e_i

(b) Pictorial depiction of notation and packing of coefficients of $E_{-\ell}(y)$

$$
\begin{array}{ccccccc}
e_0 & \cdots & e_{t/4} & e_{t/4+1} & \cdots & e_{2t/4+1} & \cdots \quad e_{7t/4+7} \quad \cdots \quad e_{2t} \\
\mathsf{SECRETS}_1 & & & \mathsf{SECRETS}_2 & & & \mathsf{SECRETS}_8
\end{array}
$$

(c) Packing of e_is in $\mathsf{SECRETS}_1, \ldots, \mathsf{SECRETS}_8$

$$
\underbrace{\begin{bmatrix} E_0(i) \\ E_{-1}(i) \\ \vdots \\ E_{-t/2}(i) \end{bmatrix}}_{E(i)} = \underbrace{\begin{bmatrix} e_{(0,0)} & \cdots & e_{(0,t/4)} \\ e_{(-1,0)} & \cdots & e_{(-1,t/4)} \\ \vdots & \vdots & \vdots \\ e_{(-t/2,0)} & \cdots & e_{(-t/2,t/4)} \end{bmatrix}}_{e_0 \qquad\qquad e_{t/4}} \begin{bmatrix} 1 \\ i \\ \vdots \\ i^{t/4} \end{bmatrix} + \ldots + \underbrace{\begin{bmatrix} e_{(0,7t/4+7)} & \cdots & e_{(0,2t)} \\ e_{(-1,7t/4+7)} & \cdots & e_{(-1,2t)} \\ \vdots & \vdots & \vdots \\ e_{(-t/2,7t/4+7)} & \cdots & e_{(-t/2,2t)} \end{bmatrix}}_{e_{7t/4+7} \qquad\qquad e_{2t}} \begin{bmatrix} i^{7t/4+7} \\ i^{7t/4+8} \\ \vdots \\ i^{2t} \end{bmatrix}
$$

$\mathsf{SECRETS}_1$ $\qquad\qquad$ $\mathsf{SECRETS}_8$

(d) Pictorial depiction of computation of $E(i)$

Figure 1: Pictorial depiction of Packed Verifiable Triple Sharing

To conduct the proof, similar to [2], the dealer finds the unique bivariate polynomial $E_{-\ell}(y)$ of

degree (at most) $2t$ defined by the polynomials

$$A(-\ell, y)B(-\ell, y) - C(-\ell, y)$$

for each $\ell \in \{0, \ldots, t/2\}$ (see Figure 1a). Following this, the dealer distributes these polynomials by sharing its coefficients using the PSS (Functionality 4.10), with $d = t/4$. Specifically, let e_i denote the vector of the ith coefficients of all the $t/2 + 1$ $E_{-\ell}(y)$ polynomials (see Figure 1b). The dealer views batches of $d = t/4 + 1$ e_is as the secrets, and invokes Functionality 4.10 at most eight times with $d = t/4$ (see Figure 1b-1c). This is because there are $2t + 1$ e_is and each batch can contain $t/4 + 1$ e_is. After these invocations of PSSs, through the f-polynomials, the parties hold a degree-$3t/2$ packed sharing of e_i (see Item 2 above). That is, parties hold $2t + 1$ packed-sharings each of degree $3t/2$ corresponding to $2t + 1$ coefficient vectors $(e_i)_{i\in\{0,\ldots,2t\}}$. Denote these polynomials by $f^{e_i}(x)$. Using linearity, these packed-sharings allow for *local* computation of degree $3t/2$ packed-sharing of evaluation points on the $t/2 + 1$ polynomials $E_{-\ell}(y)$. Let $f^{E(i)}$ denote the degree $3t/2$ polynomial which shares the evaluation of these polynomials at $y = i$ (see Figure 1d). That is, $f^{E(i)}$ shares the secrets $E(i) = (E_{-t/2}(i), \ldots, E_0(i))$. Next assume that each P_i has access to $f^{E(i)}, f^{E(0)}$ or equivalently $E(i)$ and $E(0)$. Then, to verify the product relation, each party P_i requires to check if the relation $E_{-\ell}(y) = A(-\ell, y)B(-\ell, y) - C(-\ell, y)$ holds at $y = i$ and $y = 0$ for each $\ell \in \{0, \ldots, t/2\}$. Specifically, it checks if (a) $E(0)$ is $t/2$-length zero-vector and (b) $E(i)$ is the same as the vector $(A(-\ell, i)B(-\ell, i) - C(-\ell, i))_{i\in\{0,\ldots,t/2\}}$, the latter received in the first phase. If not, it complains. If no party complains, we have that $2t + 1$ honest parties verified the product relation at distinct evaluation points on polynomials of at most degree $2t$, thus ensuring correctness of the multiplication triples. Otherwise, we must find a way to make the shares of the complaining party on A, B, C and E polynomials public and verify the product relation publicly.

The remaining tasks are (a) complaint resolution and (b) make sure P_i gets access to $E(0), E(i)$. We note that both these require reconstruction of $3t/2$-degree polynomials: $A(x, i), B(x, i), C(x, i)$, $f^{E(i)}$s for the former and $f^{E(i)}$s for the latter. The former is a public reconstruction, while the latter is private to every P_i. We therefore design a private and a public reconstruction protocol. Due to the high degree of $3t/2$, robust reconstruction via Reed-Solomon decoding is not an option. Instead, we design protocols which ensure one of the following outcomes:

- discard the dealer when it is corrupt
- identify a large of conflicts with the dealer (and restart)
- proceed with the successful reconstruction.

For the case of private reconstruction, we get a weaker guarantee wherein apart from the above three outcomes, there is a fourth possibility that the polynomials to be reconstructed privately to the honest parties are compromised. It is due to this breach, we have the weaker privacy guarantee for our VTS, namely an honest dealer's triplets may be leaked to the adversary.

Lastly, similar to PSS, here too we may have a constant number of restarts of the VTS (precisely 3) before a successful run, where the restarts can happen inside the PSS instances or as a part of the reconstructions.

Below we discuss our reconstruction protocols, and then move on to the packed VTS protocol.

6.2 Reconstructions

We present two variants of reconstructions– private and public.

6.2.1 Weak Private Reconstruction

In a private reconstruction functionality, the parties give their respective shares of $n+1$ degree-$3t/2$ polynomials. The functionality reconstructs these polynomials and sends the 0th and the ith polynomial privately to P_i. As mentioned earlier, we have four possible outcomes here: discard the dealer, detect large number of conflicts, kill the instance (in which case the honest parties' polynomials are compromised) and proceed to successful completion.

Functionality 6.1: Weak Private Reconstruction of $3t/2$-Shared Polynomials – $\mathcal{F}_{\mathsf{privRec}}$

- **Input:** All honest parties send $\mathsf{ZEROS} \subset [n]$. When the dealer is honest, $\mathsf{ZEROS} \subseteq I$. Each honest party P_j inputs shares $(h_0(j), \ldots, h_n(j))$ on polynomials, each of degree $3t/2$ to $\mathcal{F}_{\mathsf{privRec}}$. The functionality reconstructs the polynomials as $h_0(x), \ldots, h_n(x)$.
- **The functionality:**
  1. If the dealer is honest, then send ZEROS, $(h_0(i), \ldots, h_n(i))_{i \in I}$ and $(h_0(x), h_i(x))_{i \in I}$ to the adversary. If the dealer is corrupted, then send ZEROS and $h_0(x), \ldots, h_n(x)$ to the adversary.
  2. Receive a bit $\texttt{leak}$ from the adversary. If $\texttt{leak} = 1$ and the dealer is honest then send $h_0(x), \ldots, h_n(x)$ to the adversary.
  3. Receive a message M from the adversary.
  4. **Output:**
     (a) If $M = \texttt{discard}$ and the dealer is corrupted, then send $\texttt{discard}$ to all parties.
     (b) If $M = (\texttt{detect}, \mathsf{Bad})$ with $\mathsf{Bad} \cap \mathsf{ZEROS} = \emptyset$ and $|\mathsf{Bad}| > t/2$, and in case of an honest dealer then $\mathsf{Bad} \subseteq I$, then send $(\texttt{detect}, \mathsf{Bad})$ to all parties.
     (c) If $M = \texttt{kill}$ and $\texttt{leak} = 1$ then send $\texttt{kill}$ to the parties.
     (d) If $M = \texttt{proceed}$ and $\texttt{leak} = 0$ then send $(\texttt{proceed}, h_0(x), h_\ell(x))$ to each party P_ℓ.

Protocol 6.2: Weak Private Reconstruction of $3t/2$-Shared Polynomials – Π_{privRec}

- **Input:** All parties hold the same set ZEROS. Each honest party P_j holds shares $(h_0(j), \ldots, h_n(j))$ on n polynomials $h_0(x), \ldots, h_n(x)$ each of degree $3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials.
- **The protocol:**
  1. Each P_j sends $(j, h_0(j), h_\ell(j))$ to every P_ℓ.
  2. Let (j, u_j, v_j) be the value P_ℓ received from P_j. For every $j \in \mathsf{ZEROS}$, (j, u_j, v_j) is such that $u_j = v_j = 0$. Given all (j, u_j) and (j, v_j), P_ℓ looks for a codeword of distance at most $t/2$ from all the values it received (see Corollary 3.4, item 1). If there is such a codeword, set $h_0(x)$ and $h_\ell(x)$ to be the unique Reed-Solomon reconstruction respectively. If there is no such a unique codeword, then P_ℓ broadcasts $\mathsf{complaint}(\ell)$ and every party P_j broadcasts $\mathsf{reveal}(\ell, j, h_0(j), h_\ell(j))$.
  3. If no party broadcasts $\mathsf{complaint}(\ell)$ then go to Step 8d.
  4. The dealer sets $\mathsf{Bad} = \emptyset$. For each $\mathsf{reveal}(\ell, j, u, v)$ message broadcasted, the dealer verifies that $u = h_0(j)$ and $v = h_\ell(j)$. If not, then it adds j to Bad. The dealer broadcasts Bad.

5. Verify that (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$; and (ii) $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Otherwise, discard the dealer and go to Step 8a.
6. If $|\mathsf{Bad}| > t/2$ then there is a large detection and so go to Step 8b.
7. Otherwise, consider all the points $R_\ell = \{(j, u_j)\}$ and $T_\ell = \{(j, v_j)\}$ such that $\mathsf{reveal}(\ell, j, u_j, v_j)$ was broadcasted in Step 2 for $j \notin \mathsf{Bad}$ where $u_j = v_j = 0$ if $j \in \mathsf{ZEROS}$. Verify that each R_ℓ and T_ℓ define a unique polynomial of degree $3t/2$. If not, go to Step 8a. Otherwise, go to Step 8c.
8. **Output:**
   (a) Discard the dealer: Output `discard`.
   (b) Detect: Output $(\texttt{detect}, \mathsf{Bad})$.
   (c) Kill: Output `kill`.
   (d) Proceed: Each P_ℓ outputs $(\texttt{proceed}, h_0(x), h_\ell(x))$.

Lemma 6.3. *Protocol 6.2, Π_{privRec}, perfectly securely computes Functionality 6.1, $\mathcal{F}_{\mathsf{privRec}}$, in the presence of a malicious adversary controlling at most $t < n/3$.*

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the sets ZEROS, $\{(h_0(i), \ldots, h_n(i))\}_{i \in I}$ and $(h_0(x), h_i(x))_{i \in I}$.
3. For each $i \in I$, simulate every honest P_j sending $(j, h_0(j), h_i(j))$ to P_i. Receive $(i, h_0(i), h_j(i))$ from the adversary for each $i \in I$ and every honest P_j.
4. If for some honest P_j, (i, u_i, v_i) sent by the adversary is such that $u_i \neq h_0(i)$ or $v_i \neq h_j(i)$ for more than $t/2$ such $i \in I$ then simulate P_j broadcasting $\mathsf{complaint}(j)$. Also listen to all $\mathsf{complaint}(i)$ messages broadcasted by the adversary. If some party broadcasts $\mathsf{complaint}$, then send $\texttt{leak} = 1$ to the functionality. Receive $h_0(x), \ldots, h_n(x)$ from the functionality. Set $\mathsf{Bad} = \emptyset$.
5. For each $\mathsf{complaint}(\ell)$ broadcasted by some P_ℓ, simulate broadcasting $(\ell, j, h_0(j), h_\ell(j))$ for every honest P_j and listen to all the adversary's broadcasts.
6. For each $i \in I$ where (ℓ, i, u_i, v_i) broadcasted is such that $u_i \neq h_0(i)$ or $v_i \neq h_\ell(i)$, add i to Bad.
7. Simulate the dealer broadcasting Bad. If $|\mathsf{Bad}| > t/2$ then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
8. If $\texttt{leak} = 1$ then send `kill` to the functionality and halt.
9. Otherwise, send `proceed` to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$.

2. $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$.
3. R_ℓ or T_ℓ do not define a unique polynomial of degree-$3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following cases to consider:

1. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**:** In this case, it must hold that $|\mathsf{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\texttt{detect}, \mathsf{Bad})$. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad. In that case, the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party that outputs** `kill`**:** In this case, it must hold that $|\mathsf{Bad}| \leq t/2$. Moreover, there exists some party P_ℓ which complained and each R_ℓ and T_ℓ define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, all the honest parties would output `kill`. The simulator emulates the honest parties as in the real execution, hence all the simulated honest parties hold the same set Bad and see the same complaints. In this case, the simulator sends $\texttt{leak} = 1$ to the functionality, followed by `kill`, causing all the honest parties in the ideal execution to output `kill`.
3. **There exists an honest party** `proceed`**:** This implies that no party broadcast $\mathsf{complaint}$ in the real execution. Thus, it must hold that each honest party P_ℓ obtained a unique reconstruction in Step 2, which agrees with the shares of all the honest parties. Since the reconstructed polynomials and $h_0(x), h_\ell(x)$ defined by the honest parties' input shares are each of degree $3t/2$ and agree in at least $2t+1$ points, it must hold that the unique reconstructed polynomials are $h_0(x), h_\ell(x)$. Hence it must hold that all honest parties output $(\texttt{proceed}, h_0(x), h_\ell(x))$ in the real execution. Since the simulated execution is identical to the real execution, all the simulated honest parties also see that no $\mathsf{complaint}$ was broadcast. In this case, the simulator sends `proceed` to the functionality, causing all the honest parties in the ideal world to have an output that is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the set ZEROS, and the polynomials $h_0(x), \ldots, h_n(x)$.
3. Simulate the protocol where each honest party P_j holds $h_0(j), \ldots, h_n(j)$ and all parties have the same set ZEROS.
4. If some party broadcasts $\mathsf{complaint}$ in Step 2 then the simulator sends $\texttt{leak} = 1$ to the functionality.
5. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
   (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
   (b) If the output of some simulated honest party is $(\texttt{detect}, \mathsf{Bad})$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
   (c) If the output of some simulated honest party is `kill` then send `kill` to the functionality and halt.

(d) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs `discard` in the real world:** Observe that an honest party outputs `discard` if and only if one of the following conditions holds.
   (a) The dealer broadcasted Bad such that either (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$; or (ii) $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$. Since all honest parties hold the same set ZEROS, and the set Bad is broadcasted, we get that all honest parties output `discard`.
   (b) The dealer broadcasted Bad with $|\mathsf{Bad}| \leq t/2$, and for which $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. However, for $R_\ell = \{(j, u_j)\}$ and $T_\ell = \{(j, v_j)\}$ such that $\mathsf{reveal}(\ell, j, u_j, v_j)$ was broadcasted in Step 2 and $j \notin \mathsf{Bad}$, it holds that R_ℓ or T_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ and T_ℓ are public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

   Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.
2. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$ **in the real world:** In this case, it must hold that $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Moreover, it must hold that $|\mathsf{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output $(\texttt{detect}, \mathsf{Bad})$. The simulated honest parties hold an identical output, and thus the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs `kill` in the real world:** In this case, it must hold that some party P_ℓ broadcast $\mathsf{complaint}(\ell)$. Moreover, $|\mathsf{Bad}| \leq t/2$ and R_ℓ and T_ℓ each define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, it must hold that all the honest parties output `kill`. In this case, the simulated honest parties also hold an identical output. Since the simulated honest parties also observe the $\mathsf{complaint}(\ell)$ broadcast by P_ℓ, the simulator sends $\texttt{leak} = 1$ to the functionality, followed by $M = \texttt{kill}$ causing all the honest parties in the ideal world to output `kill`.
4. **There exists an honest party that outputs `proceed` in the real world:** In this case, we show that all honest parties P_ℓ output $(\texttt{proceed}, h_0(x), h_\ell(x))$ where $h_0(x), h_\ell(x)$ are the degree $3t/2$ polynomials that are interpolated from the respective input shares of the honest parties and are guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output `discard`, $(\texttt{detect}, \mathsf{Bad})$ or `kill` it must hold that no party broadcasted $\mathsf{complaint}$ in Step 2. Hence, it must hold that each honest P_ℓ has a unique reconstruction and consequently lesser than $t/2$ errors occurred in the reconstruction for P_ℓ. The reconstructed polynomials agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, hence they agree with the shares of at least $3t/2 + 1$ honest parties. Since the reconstructed polynomials and each $h_0(x), h_\ell(x)$ are of degree $3t/2$, it must hold

that the reconstructed polynomials for each honest P_ℓ are $h_0(x), h_\ell(x)$ consistent with the input shares of the honest parties. In this case, the simulated honest parties also observe that no party complains and hence the simulator sends `proceed` to the functionality, causing all the honest parties in the ideal world to obtain an output identical to the output in the real execution.

□

Lemma 6.4. *Let $t < n/3$. There exists a protocol that implements, Functionality 6.1, $\mathcal{F}_{\mathsf{privRec}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

6.2.2 Public Reconstruction

Similar to private reconstruction, shares of n degree-$3t/2$ polynomials are taken as the input by the functionality. It also receives n flags where ith one indicates if the ith polynomial needs to be made public.

Functionality 6.5: Public Reconstruction of $3t/2$-Shared Polynomials – $\mathcal{F}_{\mathsf{pubRec}}$

- **Input:** All honest parties send $\mathsf{ZEROS} \subset [n]$ and a binary vector $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$. When the dealer is honest, $\mathsf{ZEROS} \subseteq I$. Each honest party P_j inputs shares $(h_1(j), \ldots, h_n(j))$ on n polynomials, each of degree $3t/2$ to $\mathcal{F}_{\mathsf{pubRec}}$. The functionality reconstructs the polynomials as $h_1(x), \ldots, h_n(x)$.
- **The functionality:**
  1. If the dealer is honest, then send ZEROS, $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$, $h_\ell(i)$ for each $i \in I$ and every $\ell \in [n]$, $h_i(x)$ for every $i \in I$ and $h_\ell(x)$ for every $\ell \in [n]$ such that $\mathsf{pub}_\ell = 1$ to the adversary. If the dealer is corrupted, then send ZEROS, $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$ and $h_1(x), \ldots, h_n(x)$ to the adversary.
  2. Receive a message M from the adversary.
  3. **Output:**
     (a) If $M =$ `discard` and the dealer is corrupted, then send `discard` to all parties.
     (b) If $M = (\texttt{detect}, \mathsf{Bad})$ with $\mathsf{Bad} \cap \mathsf{ZEROS} = \emptyset$ and $|\mathsf{Bad}| > t/2$, and in case of an honest dealer then $\mathsf{Bad} \subseteq I$, then send $(\texttt{detect}, \mathsf{Bad})$ to all parties.
     (c) If $M =$ `proceed` then send $(\texttt{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t. } \mathsf{pub}_\ell = 1})$ to all parties.

Protocol 6.6: Public Reconstruction of $3t/2$-Shared Polynomials – Π_{pubRec}

- **Input:** All parties hold the same set ZEROS. Each honest party P_j holds shares $(h_1(j), \ldots, h_n(j))$ on n polynomials $h_1(x), \ldots, h_n(x)$ each of degree $3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials. All parties hold the same binary vector $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$.
- **The protocol:**
  1. Each P_j broadcasts $(\ell, j, h_\ell(j))$ for every ℓ such that $\mathsf{pub}_\ell = 1$.
  2. Let $(\ell, j, u_{\ell,j})$ be the value P_j broadcasted. For every $j \in \mathsf{ZEROS}$, $(\ell, j, u_{\ell,j})$ is such that $u_{\ell,j} = 0$.

3. The dealer sets $\mathsf{Bad} = \emptyset$. For each $(\ell, j, u_{\ell,j})$ message broadcasted, the dealer verifies that $u_{\ell,j} = h_\ell(j)$. If not, then it adds j to Bad. The dealer broadcasts Bad.
4. Verify that (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$; and (ii) $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Otherwise, discard the dealer and go to Step 7a.
5. If $|\mathsf{Bad}| > t/2$ then there is a large detection and so go to Step 7b.
6. Otherwise, consider all the points $R_\ell = \{(j, u_{\ell,j})\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted in Step 1 for $j \notin \mathsf{Bad}$ where $u_{\ell,j} = 0$ if $j \in \mathsf{ZEROS}$. Verify that each R_ℓ defines a unique polynomial of degree $3t/2$. If not, go to Step 7a. Otherwise, set $h_\ell(x)$ to be this unique polynomial and go to Step 7c.
7. **Output:**
   (a) Discard the dealer: Output `discard`.
   (b) Detect: Output $(\texttt{detect}, \mathsf{Bad})$.
   (c) Proceed: All parties output $(\texttt{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t. } \mathsf{pub}_\ell = 1})$.

Lemma 6.7. *Protocol 6.6, Π_{pubRec}, perfectly securely computes Functionality 6.5, $\mathcal{F}_{\mathsf{pubRec}}$, in the presence of a malicious adversary controlling at most $t < n/3$.*

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the sets ZEROS, $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$ $\{h_\ell(i)\}_{i \in I}$ and $h_\ell(x)$ for each ℓ such that $\ell \in I$ or $\mathsf{pub}_\ell = 1$.
3. For each ℓ such that $\mathsf{pub}_\ell = 1$, simulate every honest P_j broadcasting $(\ell, j, h_\ell(j))$. Listen to the broadcasts $(\ell, i, h_\ell(i))$ from the adversary for each $i \in I$. Set $\mathsf{Bad} = \phi$.
4. For each $(\ell, i, u_{\ell,i})$ broadcasted by the adversary, if $u_{\ell,i} \neq h_\ell(i)$ then add i to Bad.
5. Simulate the dealer broadcasting Bad. If $|\mathsf{Bad}| > t/2$ then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
6. Otherwise, send `proceed` to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. Thus, it remains to be shown that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$.
2. $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$.
3. R_ℓ does not define a unique polynomial of degree-$3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following two cases to consider:

1. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**:** In this case, it must hold that $|\mathsf{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\texttt{detect}, \mathsf{Bad})$. The simulator emulates the the honest parties as in the real execution, hence all the simulated honest parties hold the same set Bad and the same output. In this case, the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party** `proceed`**:** In this case, we show that all the honest parties output $(\texttt{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t } \mathsf{pub}_\ell=1})$. For this, note that since there exists an honest party that does not output `discard` or $(\texttt{detect}, \mathsf{Bad})$, it must hold that $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$, $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$ and $|\mathsf{Bad}| \leq t/2$. Moreover, for every ℓ with $\mathsf{pub}_\ell = 1$, the points R_ℓ which were broadcasted (excluding the points of parties in Bad) define a unique degree-$3t/2$ polynomial $h_\ell(x)$. Since all the corresponding messages were broadcast, all the honest parties have the same view and hence output $(\texttt{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t } \mathsf{pub}_\ell=1})$. Since the real and simulated executions are identical, the simulated honest parties have the same output. In this case, the simulator sends `proceed` to the functionality, causing the honest parties in the ideal world to receive an identical output.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the set ZEROS, $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$ and the polynomials $h_1(x), \ldots, h_n(x)$.
3. Simulate the protocol where each honest party P_j holds $h_1(j), \ldots, h_n(j)$ and all parties have the same set ZEROS and $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$.
4. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
   (a) If the output of some simulated honest party is `discard`, then send `discard` to the functionality and halt.
   (b) If the output of some simulated honest party is $(\texttt{detect}, \mathsf{Bad})$, then send $(\texttt{detect}, \mathsf{Bad})$ to the functionality and halt.
   (c) If the output of some simulated honest party is `proceed`, then send `proceed` to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs** `discard` **in the real world:** Observe that an honest party outputs `discard` if and only if one of the following conditions holds.
   (a) The dealer broadcasted Bad such that either (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$; or (ii) $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$. Since all honest parties hold the same set ZEROS, and the set Bad is broadcasted, we get that all honest parties output `discard`.
   (b) The dealer broadcasted Bad with $|\mathsf{Bad}| \leq t/2$, and for which $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. However, for $R_\ell = \{(j, u_j)\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted

in Step 1 and $j \notin \mathsf{Bad}$, it holds that R_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ is public, all honest parties will identify that there is no unique reconstruction, and all would output `discard`.

Since the simulated honest parties have the same view as the honest parties in the real execution, the simulated honest parties also output `discard`. In this case, the simulator sends `discard` to the functionality causing all the honest parties to output `discard` in the ideal execution.

2. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$ **in the real world:** In this case, it must hold that $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Moreover, it must hold that $|\mathsf{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output $(\texttt{detect}, \mathsf{Bad})$. The simulated honest parties hold an identical output, and thus the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality , which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs** `proceed` **in the real world:** In this case, we show that all honest parties P_ℓ output $(\texttt{proceed}, \{h_\ell(x)\}_{\ell \text{ s.t } \mathsf{pub}_\ell=1})$ where each $h_\ell(x)$ is the degree $3t/2$ polynomials that is interpolated from the respective input shares of the honest parties and is guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output `discard`, $(\texttt{detect}, \mathsf{Bad})$ it must hold that $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$, $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$ and $|\mathsf{Bad}| \leq t/2$. Moreover, for each $R_\ell = \{(j, u_j)\}$ such that $(\ell, j, u_{\ell,j})$ was broadcasted in Step 1 and $j \notin \mathsf{Bad}$, it holds that R_ℓ defines a unique polynomial of degree $3t/2$. Since $|\mathsf{Bad}| \leq t/2$, it must thus hold that the reconstructed polynomial for each ℓ agrees with shares of at least $n - t/2 \geq 5t/2 + 1$ parties. Consequently, the reconstructed polynomial agrees with shares of at least $3t/2 + 1$ honest parties. Since both the reconstructed polynomial and $h_\ell(x)$ are of degree $3t/2$, it holds that polynomial obtained from R_ℓ is indeed $h_\ell(x)$ that is defined by the input shares of the honest parties. In this case, the simulated honest parties also hold the same output and hence the simulator sends `proceed` to the functionality. Thus the output of honest parties in the ideal execution is identical to their output in the real world.

□

Lemma 6.8. *Let $t < n/3$. There exists a protocol that implements Functionality 6.5, $\mathcal{F}_{\mathsf{pubRec}}$, and has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

6.3 Putting Everything Together: Packed VTS

We now present the functionality and the protocol for a packed VTS.

Functionality 6.9: Packed Verifiable Triple Sharing – $\mathcal{F}_{\mathsf{PVTS}}$

The functionality is parameterized by a set of corrupted parties $I \subset [n]$.

1. **Honest dealer:**
   (a) The dealer sends $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ to $\mathcal{F}_{\mathsf{PVTS}}$.
   (b) The adversary sends $(f_i^a(x), g_i^a(y))_{i \in I}$, $(f_i^b(x), g_i^b(y))_{i \in I}$, $(f_i^c(x), g_i^c(y))_{i \in I}$ to $\mathcal{F}_{\mathsf{PVTS}}$ such that $f_i^a(k) = g_k^a(i)$, $f_i^b(k) = g_k^b(i)$ and $f_i^c(k) = g_k^c(i)$ for every $i, k \in I$.

(c) The functionality chooses random bivariate polynomials $A(x,y)$, $B(x,y)$ and $C(x,y)$ of degree $3t/2$ in x and t in y under the constraints that (i) $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ are embedded in A, B, C respectively; (ii) $A(x,i) = f_i^a(x)$, $B(x,i) = f_i^b(x)$ and $C(x,i) = f_i^c(x)$ for every $i \in I$; (iii) $A(i,y) = g_i^a(y)$, $B(i,y) = g_i^b(y)$ and $C(i,y) = g_i^c(y)$ for every $i \in I$.

2. **Corrupted dealer:** The dealer sends $A(x,y)$, $B(x,y)$ and $C(x,y)$ to $\mathcal{F}_{\mathsf{PVTS}}$ that verifies that (i) $A(x,y)$, $B(x,y)$ and $C(x,y)$ are of degree $3t/2$ in x and degree t in y; and (ii) $A(-\ell,0) \cdot B(-\ell,0) = C(-\ell,0)$ holds for each $\ell \in \{0,\ldots,t/2\}$. If not, $\mathcal{F}_{\mathsf{PVTS}}$ replaces each $A(x,y)$, $B(x,y)$ and $C(x,y)$ with $\perp$.
3. The functionality receives from the adversary a message M.
4. **Output:**
   (a) If $M =$ `kill`, then send `kill` to all parties and if the dealer is honest, then send $A(x,y), B(x,y), C(x,y)$ to the adversary.
   (b) Otherwise, send to each party P_j the pairs of polynomials $A(x,j), A(j,y)$, $B(x,j), B(j,y)$ and $C(x,j), C(j,y)$.

Protocol 6.10: Packed VTS in the $(\mathcal{F}_{\mathsf{PSS}}, \mathcal{F}_{\mathsf{privRec}}, \mathcal{F}_{\mathsf{pubRec}})$-Hybrid model – Π_{PVTS}

Input: The dealer holds three lists $\mathsf{SECRETS}_a = \{a_0,\ldots,a_{t/2}\}, \mathsf{SECRETS}_b = \{b_0,\ldots,b_{t/2}\}, \mathsf{SECRETS}_c = \{c_0,\ldots,c_{t/2}\}$, each of size $t/2+1$ such that $c_i = a_i b_i$ holds for each $i \in \{0,\ldots,t/2\}$.
The protocol:

1. All parties set $\mathsf{ZEROS} = \emptyset$.
2. **Dealing the shares of triples:** Parties invoke $\mathcal{F}_{\mathsf{PSS}}$ (Functionality 4.10) three times with $d = 0$, where the dealer inputs $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ respectively and each party inputs ZEROS. If the output of any instance is $\perp$, then proceed to Step 5a. Otherwise, each P_i holds $f_i^a(x) = A(x,i), g_i^a(y) = A(i,y)$, $f_i^b(x) = B(x,i), g_i^b(y) = B(i,y)$ and $f_i^c(x) = C(x,i)$, $g_i^c(y) = C(i,y)$.
3. **Dealing the shares of product polynomials:**
   (a) For each $\ell \in \{0,\ldots,t/2\}$, the dealer defines the polynomials $E_{-\ell}(y)$ of degree at most $2t$ such that $E_{-\ell}(y) = A(-\ell,y) \cdot B(-\ell,y) - C(-\ell,y) = e_{(-\ell,0)} + e_{(-\ell,1)}y + \ldots + e_{(-\ell,2t)}y^{2t}$. Define $e_i = (e_{(-t/2,i)},\ldots,e_{(0,i)})$ for $i \in \{0,\ldots,2t\}$, as the vector of ith coefficients of all the $t/2+1$ polynomials.
   (b) The dealer views the coefficients of these $t/2+1$ polynomials as (at most) eight matrices $\mathsf{SECRETS}_1,\ldots,\mathsf{SECRETS}_8$, each of size $(t/2+1)(t/4+1)$. Specifically, $\mathsf{SECRETS}_u(.,b) = e_{(t/4+1)\cdot(u-1)+b}$ where $b \in \{0,\ldots,t/4\}$, $u \in [8]$.
   (c) All the parties invoke $\mathcal{F}_{\mathsf{PSS}}$ (Functionality 4.10) (at most) eight times with $d = t/4$, where the dealer inputs $\mathsf{SECRETS}_u$ for each $u \in [8]$ respectively and each party inputs ZEROS. If the output of any instance is $\perp$, then proceed to Step 5a. Otherwise, each P_i holds the degree-$(t+t/2)$ polynomials $f_i^u(x) = E_u(x,i)$ and degree-$(t+t/4)$ polynomials $g_i^u(y) = E_u(i,y)$ for all $u \in [8]$.
   (d) Let $f^{e_{(t/4+1)\cdot(u-1)+b}}(x) = E_u(x,-b)$ for $b \in \{0,\ldots,t/4\}$, $u \in [8]$, denote the $3t/2$-degree polynomial that packed-shares the coefficient vector $e_{(t/4+1)\cdot(u-1)+b}$.
4. **Proof of product relation:** To check the product relation, each P_i requires to verify that $E_{-\ell}(i) = A(-\ell,i) \cdot B(-\ell,i) - C(-\ell,i)$ holds for each $\ell \in \{0,\ldots,t/2\}$. Let $E(i) =$

$(E_{-t/2}(i), \ldots, E_0(i))$ and $f^{E(i)}(x) = \sum_{k=0}^{2t} i^k f^{e_k}(x)$ be the $3t/2$-degree polynomial that packed-shares $E(i)$. Note that each $f^{E(i)}(x)$ is a linear combination of the polynomials $(f^{e_j}(x))_{j\in\{0,\ldots,2t\}}$.

(a) **Reconstructing $E(i)$ and $E(0)$ towards each P_i:** For each party P_i, every party P_j computes $f^{E(i)}(j) = \sum_{k=0}^{2t} i^k \cdot f^{e_k}(j)$ and $f^{E(0)}(j) = f^{e_0}(j)$. Parties invoke $\mathcal{F}_{\mathsf{privRec}}$ (Functionality 6.1) where P_j inputs $(f^{E(0)}(j), f^{E(1)}(j), \ldots, f^{E(n)}(j))$ and ZEROS.

i. If the output is `discard`, then proceed to Step 5a.

ii. If the output is $(\texttt{detect}, \mathsf{Bad})$, then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.

iii. If the output is `kill` then go to Step 5b.

iv. Otherwise, P_i sets $f^{E(0)}(x) = h_0(x)$ and $f^{E(i)}(x) = h_i(x)$ where $(\texttt{proceed}, h_0(x), h_i(x))$ is the output of $\mathcal{F}_{\mathsf{privRec}}$.

(b) **Verifying the product relation of each P_i and that $E(0) = e_0 = (0, \ldots, 0)$ holds:**

i. Each P_i verifies that $E(i)$ obtained from reconstructed polynomial $f^{E(i)}(x)$ matches with $(f_i^a(-\ell) \cdot f_i^b(-\ell) - f_i^c(-\ell))_{\ell\in\{0,\ldots,t/2\}}$. P_i also verifies that $f^{E(0)}(-\ell) = 0$ holds for each $\ell \in \{0, \ldots, t/2\}$. If not, then P_i broadcasts $\mathsf{complaint}(i)$.

ii. Parties construct a binary vector $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$ where $\mathsf{pub}_i = 1$ if P_i broadcasted $\mathsf{complaint}(i)$.

iii. Parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ (Functionality 6.5) five times where each P_j inputs $(\mathsf{pub}_1, \ldots, \mathsf{pub}_n)$, ZEROS in each instance and the following respectively in five instances: (i) $(f^{E(1)}(j), \ldots, f^{E(n)}(j))$, (ii) $(f^{E(0)}(j), \ldots, f^{E(0)}(j))$, (iii) $(g_j^a(1), \ldots, g_j^a(n))$, (iv) $(g_j^b(1), \ldots, g_j^b(n))$ and (v) $(g_j^c(1), \ldots, g_j^c(n))$.
For any of the above instances:

A. If the output is `discard`, then proceed to Step 5a.

B. If the output is $(\texttt{detect}, \mathsf{Bad})$, then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.

C. Otherwise, when the output is `proceed`, all parties set $f^{E(i)}(x), f^{E(0)}(x)$ and $f_i^a(x), f_i^b(x), f_i^c(x)$ as the respective output obtained in the corresponding five instances of $\mathcal{F}_{\mathsf{pubRec}}$ for each P_i which broadcasted $\mathsf{complaint}(i)$.

iv. For each P_i which complained, parties verify the checks as in Step 4(b)i using the polynomials returned by the instances of $\mathcal{F}_{\mathsf{pubRec}}$. If it does not hold for some complaining party, then go to Step 5a. Otherwise, go to Step 5c.

5. **Output:**

(a) **Discard the dealer:** All parties output $\perp$.

(b) **Kill this instance:** All parties output `kill`.

(c) **Successful:** Each P_i outputs $(f_i^a(x), g_i^a(y), f_i^b(x), g_i^b(y), f_i^c(x), g_i^c(y))$, where $\{(f_i^a(-\ell), f_i^b(-\ell), f_i^c(-\ell))\}_{\ell\in\{0,\ldots,t/2\}}$ defines P_i's shares of the $t/2+1$ multiplication triples.

Lemma 6.11. *Let $t < n/3$. Protocol 6.10, Π_{PVTS}, perfectly securely computes Functionality 6.9, $\mathcal{F}_{\mathsf{PVTS}}$, in the $(\mathcal{F}_{\mathsf{PSS}}, \mathcal{F}_{\mathsf{privRec}}, \mathcal{F}_{\mathsf{pubRec}})$-hybrid model (Functionality 4.10,6.1,6.5), in the presence of a malicious adversary, controlling at most $t < n/3$.*

Proof. We consider the case of an honest dealer and a corrupt dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke $\mathcal{A}$ on an auxiliary input z.
2. Set $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ arbitrarily as input (say, all zeros) and run the protocol where the dealer holds $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ and all other parties have no inputs. In particular, simulate the inner functionalities $\mathcal{F}_{\mathsf{PSS}}$, $\mathcal{F}_{\mathsf{privRec}}$ and $\mathcal{F}_{\mathsf{pubRec}}$ as a functionality would run it.
3. Let $A(x,y), B(x,y), C(x,y)$ be the polynomials used in the simulated functionality $\mathcal{F}_{\mathsf{PSS}}$ in the last iteration (by iteration, we mean running the protocol from Step 2 until restarting or concluding the protocol). Send $A(x,i), A(i,y)$, $B(x,i), B(i,y)$ and $C(x,i), C(i,y)$ to the functionality for every $i \in I$.
4. If the output of simulated functionality $\mathcal{F}_{\mathsf{privRec}}$ is `kill`, then send `kill` to the functionality.

We will now show that the output of the real and ideal executions are the same. For this, similar to the proof of Π_{PSS} (Lemma 4.12) consider the following games:

- Game_1: This is the real execution. We run the protocol where the honest dealer uses $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ as its input. The output of this game is the view of the adversary and the output of all honest parties in the protocol.
- Game_2: Here, we run a modified ideal model, in which the simulator receives the same $\mathsf{input} = (\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c)$ as in Game_1 as an advice, and the dealer uses input as its input to the functionality. The simulator uses input as its input instead of all zeros. The simulator runs the protocol where the input of the honest dealer is input, exactly as the real execution in Game_1. We claim that the dealer is never discarded. Then, the simulator sends to the functionality the output shares of the corrupted parties in the simulated execution and a message `kill` if some party complains in the simulated execution. If the latter holds, then the functionality sends `kill` to all. Otherwise, the functionality chooses some random polynomials $A(x,y), B(x,y), C(x,y)$ that agree with the output shares of the adversary, and gives the honest parties their shares on these polynomials. The output of this experiment is the view of the adversary as determined by the simulator, and the output of all honest parties.
- Game_3: This is the ideal execution. In particular, the simulator receives no advice, and runs as in Game_2, but with input $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ set to 0.

We will now show that the output of all the games are identically distributed.

The outputs of Game_1 and Game_2 are identically distributed. The simulator in Game_2 runs the exact same protocol as the real execution in Game_1, and therefore the view of the adversary is identical in both executions. We now turn to the output of the honest parties. We claim that in that execution, the honest dealer is never discarded. In particular:

1. An honest dealer always chooses bivariate polynomials that satisfy the conditions of Functionality 4.10 and therefore is not discarded.
2. Similarly, by the guarantees of Functionality 6.1, we have that an honest dealer is never discarded. Moreover, $\mathsf{Bad} \subseteq I$ and $\mathsf{Bad} \cup \mathsf{ZEROS} \subseteq I$ and hence $|\mathsf{ZEROS}| \leq t$ always holds and the dealer is not discarded.
3. Finally, for the same reasons as above, during the invocation of Functionality 6.5, we have that an honest dealer is never discarded.

Further, if the output of honest parties in Game_1 is `kill`, then it must hold that the output of simulated honest parties in Game_2 is also `kill` and hence the simulator sends `kill` to the functionality. All the honest parties in Game_2 also output `kill`. Finally, when the dealer is honest and parties do not output `kill`, all parties output shares on the same bivariate polynomials, which are $A(x,y), B(x,y), C(x,y)$ that the dealer used in that iteration. In Game_1, the output of all honest parties is shares on these polynomials. In Game_2, the simulator sends the shares $(A(x,i), A(i,y))_{i\in I}$, $(B(x,i), B(i,y))_{i\in I}$ and $(C(x,i), C(i,y))_{i\in I}$ to the functionality, the functionality samples new polynomials $A'(x,y), B'(x,y), C'(x,y)$ under the constraints that $A'(x,i) = A(x,i), A'(i,y) = A(i,y)$, $B'(x,i) = B(x,i), B'(i,y) = B(i,y)$ and $C'(x,i) = C(x,i), C'(i,y) = C(i,y)$ for every $i \in I$, and $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ are embedded in $A'(x,y), B'(x,y), C'(x,y)$ respectively. The output of all honest parties is then equivalent to just outputting shares on $A'(x,y), B'(x,y), C'(x,y)$. Using a similar argument as Claim 4.13, it can be seen that the output of honest parties is identically distributed.

The outputs of Game_2 and Game_3 are identically distributed. The only difference between the two games is that in Game_2 the simulator uses the same secrets input as the honest dealer uses in the ideal execution, whereas in Game_3 the honest dealer uses input as all 0. The following claim shows that the shares that the corrupted parties receives in the simulated execution is identically distributed. In both execution, given the shares and the message `kill` that the simulator sends to the functionality, the outputs of the honest parties are defined in exactly the same process (that is, the functionality either sends `kill` to all or uses $\mathsf{SECRETS}_a, \mathsf{SECRETS}_b, \mathsf{SECRETS}_c$ and the shares sent by the adversary to define the shares of honest parties). Therefore it is enough to show that the view of the adversary is identically distributed. This can be easily shown using similar argument as Claim 4.14.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ on the auxiliary input z.
2. Observe that all honest parties have no input to the protocol. Simulate running the protocol with the adversary, while also simulating Functionalities 4.10, 6.1 to the adversary as the functionality would run it.
3. If the output of some simulated honest party is $\perp$, then send $A(x,y) = B(x,y) = C(x,y) = x^{3t/2+1}$ to the functionality, in which case it sends $\perp$ to all parties.
4. If the output of some simulated honest party is `kill`, then send $M =$ `kill` to the functionality.
5. Otherwise, let J be a set of $t+1$ honest parties. Reconstruct the unique bivariate polynomials $A(x,y), B(x,y), C(x,y)$ that satisfy $A(x,j) = f_j^a(x), B(x,j) = f_j^b(x)$ and $C(x,j) = f_j^c(x)$ for every $j \in J$, where $f_j^a(x), g_j^a(y)$, $f_j^b(x), g_j^b(y)$ and $f_j^c(x), g_j^c(y)$ is the output of the simulated honest party in the simulated execution. Send $A(x,y), B(x,y), C(x,y)$ to the functionality and halt.

Since the code of each party in the protocol which is not the dealer is deterministic, and the functionalities 4.10, 6.1 are deterministic, the view of the adversary is *identical* in the real and ideal executions. Moreover, since the functionality is deterministic, we can separately consider the view and the outputs of the honest parties. Thus, all that is left to be shown is that the output of the honest parties is the same in the real and in the ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs $\perp$ in the real world.** An honest party outputs $\perp$ if and only if one of the following conditions holds: (i) $\mathcal{F}_{\mathsf{PSS}}$ (Functionality 4.10),

$\mathcal{F}_{\mathsf{privRec}}$ or $\mathcal{F}_{\mathsf{pubRec}}$ returns $\perp$, or (ii) $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$. In the first case, if $\mathcal{F}_{\mathsf{PSS}}$, $\mathcal{F}_{\mathsf{privRec}}$ or $\mathcal{F}_{\mathsf{pubRec}}$ returned $\perp$ to one honest party, it implies that all the honest parties received $\perp$, and thus all would output $\perp$ in the protocol. For the latter case, due to the guarantees of the functionalities $\mathcal{F}_{\mathsf{PSS}}$, $\mathcal{F}_{\mathsf{privRec}}$ and $\mathcal{F}_{\mathsf{pubRec}}$, all the honest parties must hold identical sets Bad and ZEROS. Hence, all would output $\perp$ in the protocol. Since the real and simulated executions are identical, the simulated honest parties must also output $\perp$. In this case, the simulator invokes the functionality with $A(x,y) = B(x,y) = C(x,y) = x^{3t/2+1}$, causing all the honest parties to output $\perp$ in the ideal execution.

2. **There exists an honest party that outputs `kill` in the real world.** An honest party outputs `kill` if and only if $\mathcal{F}_{\mathsf{privRec}}$ returns `kill`. By the guarantees of $\mathcal{F}_{\mathsf{privRec}}$, we have that all the honest parties receive `kill` from $\mathcal{F}_{\mathsf{privRec}}$ and hence all would output `kill` in the protocol. Due to identical views of the honest parties in the real and simulated execution, the same must hold true for each simulated honest party. In this case, the simulator sends `kill` to the functionality causing all the honest parties to receive output `kill` in the ideal execution. This is exactly the output of the honest parties in the real world.
3. **No honest party outputs $\perp$ or `kill` in the real world.** In this case, first note that by the guarantees of $\mathcal{F}_{\mathsf{PSS}}$, we have that all the honest parties P_i output shares $(f_i^a(x), g_i^a(y), f_i^b(x), g_i^b(y), f_i^c(x), g_i^c(y))$ consistent with some bivariate polynomials $A(x,y), B(x,y)$ and $C(x,y)$ respectively. Thus, it remains to show that $A(-\ell,0) \cdot B(-\ell,0) = C(-\ell,0)$ indeed holds for every $\ell \in \{0,\ldots,t/2\}$. For this, first note that since no honest party outputs $\perp$ or `kill`, in the final iteration of the protocol, it must hold that each P_i successfully reconstructs its $f^{E(0)}(x)$ and $f^{E(i)}(x)$ polynomials in Step 4a which are required to verify the product relation. We now claim that since the honest parties do not output $\perp$ in the final iteration of the protocol, it must hold that no honest party P_i broadcast $\mathsf{complaint}(i)$ in Step 4(b)i of the final iteration. Suppose some honest party P_i had indeed broadcasted $\mathsf{complaint}(i)$ at this step, all the parties must have invoked $\mathcal{F}_{\mathsf{pubRec}}$ in Step 4(b)iii. Since this is the final iteration, we have that $\mathcal{F}_{\mathsf{pubRec}}$ did not output $(\texttt{detect}, \mathsf{Bad})$ such that $|\mathsf{Bad}| > t/2$, as otherwise the parties would have rebooted to start a new iteration. Moreover, no party output $\perp$ hence $\mathcal{F}_{\mathsf{pubRec}}$ did not output `discard`. Consequently, each party would successfully reconstructed the f polynomials of the complaining party P_i. Given that the reconstructed polynomials agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, it must agree with with the shares of at least $3t/2 + 1$ honest parties (since at most $t/2$ parties may have been identified as Bad in $\mathcal{F}_{\mathsf{pubRec}}$), thus forcing the reconstruction of the dealer's polynomial committed during the sharing phase. Thus, all the parties would publicly verify an honest P_i's complain and as a result discard the dealer, which is a contradiction. Hence, it must hold that no honest party P_i broadcast $\mathsf{complaint}(i)$ in Step 4(b)i of the final iteration. Since no honest party complains, it must hold that $A(-\ell,i) \cdot B(-\ell,i) - C(-\ell,i) = E_{-\ell}(i)$ holds for each honest P_i and every $\ell \in \{0,\ldots,t/2\}$. Since each of $A(-\ell,y) \cdot B(-\ell,y) - C(-\ell,y)$ and $E_{-\ell}(y)$ are at most degree $2t$ polynomials, which agree in at least $2t+1$ points corresponding to the honest parties, it must hold that $A(-\ell,y) \cdot B(-\ell,y) - C(-\ell,y) = E_{-\ell}(y)$. Using a similar argument, it can be clearly seen that $E_{-\ell}(0)$ holds when the dealer is not discarded. Thus, it can be concluded that $A(-\ell,0) \cdot B(-\ell,0) = C(-\ell,0)$ holds. In the simulated execution, the simulator chooses an arbitrary set of $t+1$ honest parties and reconstructs the bivariate polynomials that agree with their output. Since the real and simulated executions are identical, it must hold that these polynomials are $A(x,y), B(x,y)$ and $C(x,y)$ respectively. The simulator then invokes

the functionality with these polynomials causing the honest parties in the ideal execution to output shares that are identical to those in the real execution.

□

Lemma 6.12. *Let $t < n/3$ and $d \leq t/4$. There exists a protocol that implements Functionality 6.9, $\mathcal{F}_{\mathsf{PVTS}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast for sharing $\mathcal{O}(n)$ multiplication triples simultaneously in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

7 Batched Verifiable Triple Sharing

In this section, we discuss how to run m instances of our packed VTS while keeping the broadcast cost unchanged. Specifically, while one instance of the packed VTS requires $\mathcal{O}(n^2 \log n)$ bits communication over point-to-point channels and each party broadcasts $\mathcal{O}(n \log n)$ bits, we show how to run m instances together at a cost of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and retain the same broadcast of $\mathcal{O}(n \log n)$ bits per party. An important property to note here is that although an adversary can compromise an instance of packed VTS even for an honest dealer, in the batched version we ensure that at most n out of the m instances can be compromised. When $m > n$, our protocols achieves the properties of a verifiable triple sharing. That is, when the dealer is honest, our protocol guarantees privacy of triples obtained from the $m - n$ uncompromised instances of packed VTS. While for a corrupt dealer, as in packed VTS, our zero knowledge proof ensures the correctness of the triples. We now elaborate on the changes necessary to the packed VTS protocol and its sub-protocols.

Dealing the shares of triples and product polynomials. The dealer holds m lists of triples, and one set $\mathsf{ZEROS} \subset [n]$. Further, for the parties in ZEROS, the shares of on all the bivariate polynomials are assumed to be 0. For sharing the triples as well as the coefficients of the product polynomials of m instances, batched variant of the packed secret sharing $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ (Functionality 5.4) is invoked. This ensures a cost of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and a broadcast of $\mathcal{O}(n \log n)$ bits per party.

Batched private reconstruction. The change in the protocol here follows closely to the batched reconstruction of g polynomials defined in Section 5. Specifically, in Step 2, a party P_ℓ may fail to reconstruct h_0 or h_ℓ in multiple instances. However, it suffices for P_ℓ to choose one instance (say the instance with a minimum index β) and complain with respect to β. The complaint for a party now looks like $\mathsf{complaint}(\ell, \beta)$. Following this, the public verification happens only for the βth instance for P_ℓ. In the case that the dealer is not discarded and the publicly identified set Bad has at most $t/2$ parties, each party is guaranteed to reconstruct its polynomials in all the instances successfully. This follows similar to the argument described in Section 5. Specifically, using the publicly identified Bad set, and in addition the locally identified conflicts (by comparing the values it received over point-to-point channels and those broadcast by parties during public reveal), a party can recognize more than $t/2$ errors and correct the remaining (less than $t/2$) errors across the m instances. We provide the complete modelling via $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$ (Functionality 7.1). The protocol $\Pi_{\mathsf{privRec}}^{\mathsf{batched}}$ (Protocol 7.2) and its security proof follow.

Functionality 7.1: Batched Private Reconstruction of $3t/2$-Shared Polynomials – $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$

- **Input:** All honest parties send $\mathsf{ZEROS} \subset [n]$. When the dealer is honest, $\mathsf{ZEROS} \subseteq I$. Each honest party inputs shares on m sets of polynomials, each of degree $3t/2$ to $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$. It reconstructs the polynomials as $\{h_0^k(x), \ldots, h_n^k(x)\}_{k\in[m]}$.
- **The functionality:**
  1. If the dealer is honest, then send (a) the polynomials $h_0^k(x), \ldots, h_n^k(x)$ for every $k \in [m]$ to the dealer; (b) ZEROS, $(h_0^k(i), \ldots, h_n^k(i))_{i\in I}$ and $(h_0^k(x), h_i^k(x))_{i\in I}$ for each $k \in [m]$ to the adversary. If the dealer is corrupted, then send ZEROS and $h_0^k(x), \ldots, h_n^k(x)$ to the adversary for each $k \in [m]$.
  2. Receive a binary vector $\mathtt{leak}$ of length m with at most n 1's from the adversary. For every k such that $\mathtt{leak}_k = 1$, send $\{h_0^k(x), \ldots, h_n^k(x)\}$ to the adversary.
  3. Receive a message M from the adversary.
  4. **Output:**
     (a) If $M = \mathtt{discard}$ and the dealer is corrupted, then send $\mathtt{discard}$ to all parties.
     (b) If $M = (\mathtt{detect}, \mathsf{Bad})$ with $\mathsf{Bad} \cap \mathsf{ZEROS} = \emptyset$ and $|\mathsf{Bad}| > t/2$, and in case of an honest dealer then $\mathsf{Bad} \subseteq I$, then send $(\mathtt{detect}, \mathsf{Bad})$ to all parties.
     (c) If $M = \mathtt{kill}$ and $\mathtt{leak}$ is not a 0-vector then then send $(\mathtt{proceed}, h_0^k(x), h_\ell^k(x))$ to each party P_ℓ for every k where such that $\mathtt{leak}_k = 0$. For every other k, send $\mathtt{kill}$ to all the parties.
     (d) If $M = \mathtt{proceed}$ and $\mathtt{leak}$ is a 0-vector then send $(\mathtt{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$ to each party P_ℓ.

Protocol 7.2: Batched Private Reconstruction of $3t/2$-Shared Polynomials – $\Pi_{\mathsf{privRec}}^{\mathsf{batched}}$

- **Input:** All parties hold the same set ZEROS. Each honest party P_j holds m sets of shares $(h_0^k(j), \ldots, h_n^k(j))$ for each $k \in [m]$ on polynomials $h_0^k(x), \ldots, h_n^k(x)$ each of degree $3t/2$. It is guaranteed that all the shares of honest parties lie on the same $3t/2$ degree polynomials.
- **The protocol:**
  1. Each P_j sends $(j, h_0^k(j), h_\ell^k(j))$ for every $k \in [m]$ to every P_ℓ.
  2. Let (j, u_j^k, v_j^k) be the value P_ℓ received from P_j. For every $j \in \mathsf{ZEROS}$, (j, u_j^k, v_j^k) is such that $u_j = v_j = 0$. Given all (j, u_j^k) and (j, v_j^k), P_ℓ looks for a codeword of distance at most $t/2$ from all the values it received (see Corollary 3.4, item 1). If there is such a codeword, set $h_0^k(j)$ and $h_\ell^k(x)$ to be the unique Reed-Solomon reconstruction respectively for each $k \in [m]$. If there is no such a unique codeword for some k, then P_ℓ broadcasts $\mathsf{complaint}(\ell, \beta)$ where β is the minimal index from $\{1, \ldots, m\}$ and every party P_j broadcasts $\mathsf{reveal}(\ell, \beta, j, h_0^\beta(j), h_\ell^\beta(j))$. If P_ℓ sees some $\mathsf{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ with a value different than the one sent by P_j then add j to $\mathsf{localBad}_\ell$.
  3. If no party broadcasts $\mathsf{complaint}(\ell, \beta)$ then go to Step 9d.
  4. The dealer sets $\mathsf{Bad} = \emptyset$. For each $\mathsf{reveal}(\ell, \beta, j, u, v)$ message broadcasted, the dealer verifies that $u = h_0^\beta(j)$ and $v = h_\ell^\beta(j)$. If not, then it adds j to Bad. The dealer broadcasts Bad.
  5. Verify that (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$; and (ii) $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Otherwise, discard – go to Step 9a.
  6. If $|\mathsf{Bad}| > t/2$ then there is a large detection – go to Step 9b.

7. Otherwise, consider all the points $R_\ell = \{(j, u_j^\beta)\}$ and $T_\ell = \{(j, v_j^\beta)\}$ such that $\mathsf{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ was broadcasted in Step 2 for $j \notin \mathsf{Bad}$, where $u_j^\beta = v_j^\beta = 0$ if $j \in \mathsf{ZEROS}$. Verify that R_ℓ and T_ℓ each define a unique polynomial of degree $3t/2$. If not, go to Step 9a.
8. If the dealer is not publicly discarded, then each P_ℓ sets $h_0^k(x)$ and $h_\ell^k(x)$ for each $k \in [m]$ as the unique decoding of the points (j, u_j^k) and (j, v_j^k) respectively (see Corollary 3.4, item 2) for every $j \notin \mathsf{Bad} \cup \mathsf{localBad}_\ell$, received in Step 1 and proceed to Step 9c.
9. **Output:**
   (a) Discard the dealer: Output $\mathtt{discard}$.
   (b) Detect: Output $(\mathtt{detect}, \mathsf{Bad})$.
   (c) Leak: Output $(\mathtt{proceed}, h_0^k(x), h_\ell^k(x))$ for each $k \in [m]$ such that $\mathsf{complaint}(\cdot, k)$ was not broadcasted in Step 2. For every other k, output $\mathtt{kill}$.
   (d) Proceed: Each P_ℓ outputs $(\mathtt{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$.

Lemma 7.3. *Protocol 7.2,* $\Pi^{\mathsf{batched}}_{\mathsf{privRec}}$*, perfectly securely computes Functionality 7.1,* $\mathcal{F}^{\mathsf{batched}}_{\mathsf{privRec}}$*, in the presence of a malicious adversary controlling at most* $t < n/3$*.*

Proof. We show the case of an honest dealer and a corrupted dealer separately.

The case of an honest dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the sets ZEROS, $\{(h_0^k(i), \ldots, h_n^k(i))\}_{i \in I}$ and $(h_0^k(x), h_i^k(x))_{i \in I}$ for every $k \in [m]$.
3. For each $i \in I$, simulate every honest P_j sending $(j, h_0^k(j), h_i^k(j))$ to P_i. Receive $(i, h_0^k(i), h_j^k(i))$ from the adversary for each $i \in I$ and every honest P_j, for every $k \in [m]$.
4. If for some honest P_j, (i, u_i^k, v_i^k) sent by the adversary is such that $u_i^k \neq h_0^k(i)$ or $v_i^k \neq h_j^k(i)$ for more than $t/2$ such $i \in I$ then simulate P_j broadcasting $\mathsf{complaint}(j, \beta)$ where β is the minimal index from $[m]$. Also listen to all $\mathsf{complaint}(i, \beta)$ messages broadcasted by the adversary. Construct a vector $\mathtt{leak}$ of length m such that $\mathtt{leak}_\beta = 1$ for each β for which $\mathsf{complaint}(\cdot, \beta)$ was broadcasted by some party, and send $\mathtt{leak}$ to the functionality. Receive $\{h_0^\beta(x), \ldots, h_n^\beta(x)\}$ from the functionality for each β where $\mathtt{leak}_\beta = 1$. Set $\mathsf{Bad} = \emptyset$.
5. For each $\mathsf{complaint}(\ell, \beta)$ broadcasted by some P_ℓ, simulate broadcasting $(\ell, \beta, j, h_0^\beta(j), h_\ell^\beta(j))$ for every honest P_j and listen to all the adversary's broadcasts.
6. For each $i \in I$ where $(\ell, \beta, i, u_i^\beta, v_i^\beta)$ broadcasted is such that $u_i^\beta \neq h_0^\beta(i)$ or $v_i^\beta \neq h_\ell^\beta(i)$, add i to Bad.
7. Simulate the dealer broadcasting Bad. If $|\mathsf{Bad}| > t/2$ then send $(\mathtt{detect}, \mathsf{Bad})$ to the functionality and halt.
8. If $\mathtt{leak}$ is not a 0-vector then send $\mathtt{kill}$ to the functionality and halt.
9. Otherwise, send $\mathtt{proceed}$ to the functionality and halt.

The protocol as well as the simulator is deterministic. Hence, it can be easily observed that the view of the adversary in the real and ideal executions is identical. It remains to show that the output of honest parties is identical in the real and ideal world.

Towards that end, note that in the real world, an honest dealer is discarded if and only if one of the following conditions holds:

1. $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$.
2. $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$.
3. R_ℓ or T_ℓ do not define a unique polynomial of degree-$3t/2$.

Note that for an honest dealer, an honest party never belongs to Bad and we have that $\mathsf{ZEROS} \subseteq I$. It is clear that none of the above conditions hold and the dealer is not discarded. We thus have the following cases to consider:

1. **There exists an honest party that outputs** $(\texttt{detect}, \mathsf{Bad})$**:** In this case, it must hold that $|\mathsf{Bad}| > t/2$. Since the corresponding message was broadcasted by the dealer, it must hold that all the honest parties output $(\texttt{detect}, \mathsf{Bad})$. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties hold the same set Bad. In that case, the simulator sends $(\texttt{detect}, \mathsf{Bad})$ to the functionality, causing all the honest parties in the ideal world to output the same.
2. **There exists an honest party that outputs** `kill` **for some** $k \in [m]$**:** In this case, it must hold that $|\mathsf{Bad}| \leq t/2$. Moreover, there exists some party P_ℓ which broadcasted $\mathsf{complaint}(\ell, k)$ and each R_ℓ and T_ℓ define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, all the honest parties would output `kill` for k. The same holds true for each k for which $\mathsf{complaint}(\cdot, k)$ was broadcast. For every other k, it must hold that each honest party P_ℓ is able to reconstruct $h_0^k(x), h_\ell^k(x)$ either in Step 2 or in Step 8 after discarding the shares of (at least $t/2$ corrupt) parties in $\mathsf{Bad} \cup \mathsf{localBad}_\ell$ (similar to batched reconstruction of g polynomials in Section 5). Since these polynomials are degree $3t/2$ and agree with the shares of at least $2t+1$ honest parties, they are guaranteed to be the same polynomials defined by the input shares of the honest parties. The simulator emulates the interaction of the honest parties with the dealer as in the real execution, hence all the simulated honest parties P_ℓ hold the same set Bad, see the same complaints. In this case, the simulator constructs `leak` with $\texttt{leak}_k = 1$ for each k such that $\mathsf{complaint}(\cdot, k)$ was broadcast and sends `leak` to the functionality, followed by `kill`, causing all the honest parties in the ideal execution to have the same output as the honest parties in the real execution.
3. **There exists an honest party that outputs** `proceed` **for all** $k \in [m]$**:** This implies that no party broadcast $\mathsf{complaint}$ in the real execution. Thus, it must hold that each honest party P_ℓ obtained a unique reconstruction in Step 2, which agrees with the shares of all the honest parties. Since the reconstructed polynomials and $h_0^k(x), h_\ell^k(x)$ defined by the honest parties' input shares are each of degree $3t/2$ and agree in at least $2t+1$ points, it must hold that the unique reconstructed polynomials are $h_0^k(x), h_\ell^k(x)$. Hence it must hold that all honest parties output $(\texttt{proceed}, h_0^k(x), h_\ell^k(x))$ for every $k \in [m]$ in the real execution. Since the simulated execution is identical to the real execution, all the simulated honest parties also see that no $\mathsf{complaint}$ was broadcast. In this case, the simulator sends `proceed` to the functionality, causing all the honest parties in the ideal world to have an output that is identical to the output of the honest parties in the real world.

The case of a corrupted dealer. The simulator is as follows:

1. Invoke the adversary $\mathcal{A}$ with an auxiliary input z.
2. Receive from the functionality the set ZEROS, and the polynomials $h_0^k(x), \ldots, h_n^k(x)$ for each $k \in [m]$.

3. Simulate the protocol where each honest party P_j holds $h_0^k(j), \ldots, h_n^k(j)$ for each k and all parties have the same set ZEROS.
4. If some party P_ℓ broadcasts $\mathsf{complaint}(\ell, \beta)$ in Step 2 then the simulator sets $\mathtt{leak}_\beta = 1$. It sends $\mathtt{leak} = (\mathtt{leak}_1, \ldots, \mathtt{leak}_m)$ to the functionality.
5. Send the message M to the functionality according to the following cases (the proof will show that the cases are mutually-exclusive):
   (a) If the output of some simulated honest party is $\mathtt{discard}$, then send $\mathtt{discard}$ to the functionality and halt.
   (b) If the output of some simulated honest party is $(\mathtt{detect}, \mathsf{Bad})$, then send $(\mathtt{detect}, \mathsf{Bad})$ to the functionality and halt.
   (c) If the output of some simulated honest party is $\mathtt{kill}$ for some k then send $\mathtt{kill}$ to the functionality and halt.
   (d) If the output of some simulated honest party is $\mathtt{proceed}$ for all k, then send $\mathtt{proceed}$ to the functionality and halt.

Since the simulator uses the exact same inputs of the simulated honest parties as the real honest parties in the real execution, and since the protocol is deterministic, we get that the view of the adversary is exactly the same in the real and in the ideal executions. Thus it remains to be shown that the output of the honest parties is identical in the real and ideal executions. We have the following cases to consider:

1. **There exists an honest party that outputs $\mathtt{discard}$ in the real world:** Observe that an honest party outputs $\mathtt{discard}$ if and only if one of the following conditions holds.
   (a) The dealer broadcasted Bad such that either (i) $|\mathsf{ZEROS} \cup \mathsf{Bad}| > t$; or (ii) $\mathsf{Bad} \not\subset [n] \setminus \mathsf{ZEROS}$. Since all honest parties hold the same set ZEROS, and the set Bad is broadcasted, we get that all honest parties output $\mathtt{discard}$.
   (b) The dealer broadcasted Bad with $|\mathsf{Bad}| \leq t/2$, and for which $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. However, for $R_\ell = \{(j, u_j^\beta)\}$ and $T_\ell = \{(j, v_j^\beta)\}$ such that $\mathsf{reveal}(\ell, \beta, j, u_j^\beta, v_j^\beta)$ was broadcasted in Step 2 and $j \notin \mathsf{Bad}$, it holds that R_ℓ or T_ℓ does not define a unique polynomial of degree $3t/2$. Since the set R_ℓ and T_ℓ are public, all honest parties will identify that there is no unique reconstruction, and all would output $\mathtt{discard}$.

   Since the simulated honest parties have the same view as the honest parties, the simulated honest parties also output $\mathtt{discard}$. In this case, the simulator sends $\mathtt{discard}$ to the functionality causing all the honest parties to output $\mathtt{discard}$ in the ideal execution.
2. **There exists an honest party that outputs $(\mathtt{detect}, \mathsf{Bad})$ in the real world:** In this case, it must hold that $|\mathsf{ZEROS} \cup \mathsf{Bad}| \leq t$ and $\mathsf{Bad} \subset [n] \setminus \mathsf{ZEROS}$. Moreover, it must hold that $|\mathsf{Bad}| \geq t/2$. Since the corresponding set Bad is broadcast, all the honest parties hold the same set and hence output $(\mathtt{detect}, \mathsf{Bad})$. The simulated honest parties hold an identical output, and thus the simulator sends $(\mathtt{detect}, \mathsf{Bad})$ to the functionality , which in turn sends the same to all the honest parties in the ideal execution.
3. **There exists an honest party that outputs $\mathtt{kill}$ for some $k \in [m]$ in the real world:** In this case, it must hold that some party P_ℓ broadcast $\mathsf{complaint}(\ell, k)$. Moreover, $|\mathsf{Bad}| \leq t/2$ and R_ℓ and T_ℓ each define a unique degree $3t/2$ polynomials. Since all the corresponding messages are broadcast, it must hold that all the honest parties output $\mathtt{kill}$

for k. This must hold for each k for which $\mathsf{complaint}(\cdot, k)$ was broadcast by some party. Further, for every other k, it must hold that each honest party P_ℓ is able to reconstruct $h_0^k(x), h_\ell^k(x)$ either in Step 2 or in Step 8 after discarding the shares of (at least $t/2$) parties in $\mathsf{Bad} \cup \mathsf{localBad}_\ell$ (similar to batched reconstruction of g polynomials in Section 5). Note than an honest party never belongs to $\mathsf{localBad}_\ell$ set of another honest party P_ℓ. Hence, during the reconstruction of its polynomials, a party may discard shares of at most $t/2$ honest parties from Bad. Since these reconstructed polynomials are degree $3t/2$ and agree with the shares of at least $2t + 1 - t/2 \geq 3t/2 + 1$ honest parties, they are guaranteed to be the same polynomials defined by the input shares of the honest parties. The simulator emulates the honest parties as in the real execution, hence all the simulated honest parties P_ℓ see the same complaints. In this case, the simulator constructs $\texttt{leak}$ with $\texttt{leak}_k = 1$ for each k such that $\mathsf{complaint}(\cdot, k)$ was broadcast and sends $\texttt{leak}$ to the functionality, followed by $\texttt{kill}$, causing all the honest parties in the ideal execution to have the same output as the honest parties in the real execution.

4. **There exists an honest party that outputs $\texttt{proceed}$ for all $k \in [m]$ in the real world:** In this case, we show that all honest parties P_ℓ output $(\texttt{proceed}, h_0^k(x), h_\ell^k(x))$ where $h_0^k(x), h_\ell^k(x)$ are the degree $3t/2$ polynomials that are interpolated from the respective input shares of the honest parties and are guaranteed to exist under our input assumption. Towards that, observe that since there exists an honest party that does not output $\texttt{discard}$, $(\texttt{detect}, \mathsf{Bad})$ or $\texttt{kill}$ for any k it must hold that no party broadcasted $\mathsf{complaint}$ in Step 2. Hence, it must hold that each honest P_ℓ has a unique reconstruction and consequently lesser than $t/2$ errors occurred in the reconstruction for P_ℓ. The reconstructed polynomials agree with the shares of at least $n - t/2 \geq 5t/2 + 1$ parties, hence they agree with the shares of at least $3t/2 + 1$ honest parties. Since the reconstructed polynomials and each $h_0^k(x), h_\ell^k(x)$ are of degree $3t/2$, it must hold that the reconstructed polynomials for each honest P_ℓ are $h_0^k(x), h_\ell^k(x)$ consistent with the input shares of the honest parties. In this case, the simulated honest parties also observe that no party complains and hence the simulator sends $\texttt{proceed}$ to the functionality, causing all the honest parties in the ideal world to obtain an output identical to the output in the real execution.

□

Lemma 7.4. *Let $t < n/3$. There exists a protocol that implements Functionality 7.1 and has a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Every party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Note that in the batched private reconstruction, the public verification happens for one instance corresponding to each party's complaint. Thus, in total, we have that the public reveal of shares is performed for at most n instances out of the total m that are batched together. Consequently, for an honest dealer, we have that the adversary does not learn any additional information for the $m - n$ instances where the shares are not publicly revealed. Since private reconstruction is used for reconstructing distinct points on the product polynomials to each party in the triple sharing protocol, we have the same privacy guarantee of $m - n$ uncompromised instances carry over to the batched packed VTS protocol.

Public reconstruction for complaining parties. Similar to private reconstruction, if the product relation fails to hold during the verification in Step 4b, a party P_i complains only for one instance, say β_i. The public verification proceeds as before via $\mathcal{F}_{\mathsf{pubRec}}$ (Functionality 6.5),

with the only difference being that the ith input share of every party corresponds to the shares of polynomials in β_ith instance, for which P_i broadcasts a complaint. At the conclusion of this verification, if the dealer is not discarded then parties either detect more than $t/2$ conflicts with the dealer, or successfully reconstruct the polynomials of P_i for the β_ith instance. In the latter case, the verification of product polynomial is performed on the reconstructed polynomial and the dealer is discarded in case of failure. Note that when the dealer is corrupt, it is sufficient for an honest party to complain in one instance. The "binding" property of $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ guarantees that for any individual instance, if a polynomial is reconstructed then it is the same polynomial committed to by the dealer in the sharing phase. Thus if the dealer shares incorrect multiplication triples, causing an honest party to broadcast complaint in some β_ith instance such that its polynomials are reconstructed successfully during Step 4b, then the dealer is guaranteed to be discarded. The functionality for batched verifiable secret sharing appears below. We provide the protocol for completeness.

Functionality 7.5: Batched and Packed Verifiable Triple Sharing Functionality – $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$

The functionality is parameterized by a set of corrupted parties $I \subset [n]$.

1. **Honest dealer:**
   (a) The dealer sends $\{\mathsf{SECRETS}_a^\alpha, \mathsf{SECRETS}_b^\alpha, \mathsf{SECRETS}_c^\alpha\}_{\alpha\in[m]}$ to $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$.
   (b) The adversary sends $(f_i^{\alpha,a}(x), g_i^{\alpha,a}(y))_{i\in I}$, $(f_i^{\alpha,b}(x), g_i^{\alpha,b}(y))_{i\in I}$, $(f_i^{\alpha,c}(x), g_i^{\alpha,c}(y))_{i\in I}$ to $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$ such that $f_i^{\alpha,a}(k) = g_k^{\alpha,a}(i)$, $f_i^{\alpha,b}(k) = g_k^{\alpha,b}(i)$ and $f_i^{\alpha,c}(k) = g_k^{\alpha,c}(i)$ for every $i, k \in I$ and every $\alpha \in [m]$.
   (c) The functionality chooses random bivariate polynomials $A^\alpha(x,y)$, $B^\alpha(x,y)$ and $C^\alpha(x,y)$ of degree $3t/2$ in x and t in y under the constraints that (i) $\mathsf{SECRETS}_a^\alpha, \mathsf{SECRETS}_b^\alpha, \mathsf{SECRETS}_c^\alpha$ are embedded in $A^\alpha, B^\alpha, C^\alpha$ respectively; (ii) $A^\alpha(x,i) = f_i^{\alpha,a}(x)$, $B^\alpha(x,i) = f_i^{\alpha,b}(x)$ and $C^\alpha(x,i) = f_i^{\alpha,c}(x)$ for every $i \in I$; (iii) $A^\alpha(i,y) = g_i^{\alpha,a}(y)$, $B^\alpha(i,y) = g_i^{\alpha,b}(y)$ and $C^\alpha(i,y) = g_i^{\alpha,c}(y)$ for every $i \in I$.
2. **Corrupted dealer:** The dealer sends $A^\alpha(x,y)$, $B^\alpha(x,y)$ and $C^\alpha(x,y)$ for each $\alpha \in [m]$ to $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$ that verifies that (i) $A^\alpha(x,y)$, $B^\alpha(x,y)$ and $C^\alpha(x,y)$ are of degree $3t/2$ in x and degree t in y for each $\alpha \in [m]$; and (ii) $A(-i,0)\cdot B(-i,0) = C(-i,0)$ holds for each $i \in \{0,\ldots,t/2\}$. If not, $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$ replaces each $A^\alpha(x,y)$, $B^\alpha(x,y)$ and $C^\alpha(x,y)$ with $\perp$.
3. **Output:** $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$ sends to each party P_j the pairs of polynomials $A^\alpha(x,j), A^\alpha(j,y)$, $B^\alpha(x,j)$, $B^\alpha(j,y)$ and $C^\alpha(x,j), C^\alpha(j,y)$ for every $\alpha \in [m-n]$.

Protocol 7.6: Batched and Packed VTS in the $(\mathcal{F}_{\mathsf{PSS}}, \mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}, \mathcal{F}_{\mathsf{pubRec}})$-hybrid model – $\Pi_{\mathsf{PVTS}}^{\mathsf{batched}}$

Input: The dealer holds m sets of three lists $\mathsf{SECRETS}_a^\alpha = \{a_0^\alpha, \ldots, a_{t/2}^\alpha\}, \mathsf{SECRETS}_b^\alpha = \{b_0^\alpha, \ldots, b_{t/2}^\alpha\}$, $\mathsf{SECRETS}_c^\alpha = \{c_0^\alpha, \ldots, c_{t/2}^\alpha\}$, each of size $t/2+1$ such that $c_i^\alpha = a_i^\alpha b_i^\alpha$ holds for each $i \in \{0,\ldots,t/2\}$ and each $\alpha \in [m]$.
The protocol:

1. All parties set $\mathsf{ZEROS} = \emptyset$.
2. **Dealing the shares of triples:** Parties invoke $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ (Functionality 5.4) three times with $d = 0$. The dealer inputs $(\mathsf{SECRETS}_a^\alpha)_{\alpha\in[m]}, (\mathsf{SECRETS}_b^\alpha)_{\alpha\in[m]}, (\mathsf{SECRETS}_c^\alpha)_{\alpha\in[m]}$ respectively and each party inputs ZEROS. If the output of any instance is $\perp$, then proceed to Step 5a.

Otherwise, each P_i holds $f_i^{\alpha,a}(x) = A^\alpha(x,i), g_i^{\alpha,a}(y) = A^\alpha(i,y)$, $f_i^{\alpha,b}(x) = B^\alpha(x,i), g_i^{\alpha,b}(y) = B^\alpha(i,y)$ and $f_i^{\alpha,c}(x) = C^\alpha(x,i), g_i^{\alpha,c}(y) = C^\alpha(i,y)$.

3. **Dealing the shares of product polynomials:**
   (a) For each $\ell \in \{0,\ldots,t/2\}$ and every $\alpha \in [m]$, the dealer defines the polynomials $E^\alpha_{-\ell}(y)$ of degree at most $2t$ such that $E^\alpha_{-\ell}(y) = A^\alpha(-\ell,y)\cdot B^\alpha(-\ell,y) - C^\alpha(-\ell,y) = e^\alpha_{(-\ell,0)} + e^\alpha_{(-\ell,1)}y + \ldots + e^\alpha_{(-\ell,2t)}y^{2t}$. Define $e_i^\alpha = (e^\alpha_{(-t/2,i)},\ldots,e^\alpha_{(0,i)})$ for $i \in \{0,\ldots,2t\}$, as the vector of ith coefficients of all the $t/2+1$ polynomials for the αth set of polynomials.
   (b) The dealer views the coefficients of these $t/2+1$ polynomials for each of the m sets of polynomials as (at most) eight matrices $\mathsf{SECRETS}_1^\alpha,\ldots,\mathsf{SECRETS}_8^\alpha$, each of size $(t/2+1)(t/4+1)$. Specifically, $\mathsf{SECRETS}_u^\alpha(\cdot,b) = e^\alpha_{(t/4+1)\cdot(u-1)+b}$ where $b \in \{0,\ldots,t/4\}$ and $u \in [8]$.
   (c) All the parties invoke $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ (Functionality 5.4) (at most) eight times with $d = t/4$, where the dealer inputs $\mathsf{SECRETS}_u^\alpha$ for each $u \in [8]$ and every $\alpha \in [m]$ respectively and each party inputs ZEROS. If the output of any instance is $\perp$, then proceed to Step 5a. Otherwise, each P_i holds the degree-$(t+t/2)$ polynomials $f_i^{\alpha,u}(x) = E_u^\alpha(x,i)$ and degree-$(t+t/4)$ polynomials $g_i^{\alpha,u}(y) = E_u^\alpha(i,y)$ for all $u \in [8]$ and every $\alpha \in [m]$.
   (d) Let $f^{e^\alpha_{(t/4+1)\cdot(u-1)+b}}(x) = E_u^\alpha(x,-b)$ for $b \in \{0,\ldots,t/4\}$, $u \in [8]$ and $\alpha \in [m]$, denote the $3t/2$-degree polynomial that packed-shares the coefficient vector $e^\alpha_{(t/4+1)\cdot(u-1)+b}$.
4. **Proof of product relation:** To check the product relation, each P_i requires to verify that $E^\alpha_{-\ell}(i) = A^\alpha(-\ell,i)\cdot B^\alpha(-\ell,i) - C^\alpha(-\ell,i)$ holds for each $\ell \in \{0,\ldots,t/2\}$. Let $E^\alpha(i) = (E^\alpha_{-t/2}(i),\ldots,E^\alpha_0(i))$ and $f^{E^\alpha(i)}(x)$ be the $3t/2$-degree polynomial that packed-shares of $E^\alpha(i)$. Note that each $f^{E^\alpha(i)}(x)$ is a linear combination of the polynomials $(f^{e_j^\alpha}(x))_{j\in\{0,\ldots,2t\}}$ for each $\alpha \in [m]$.
   (a) **Reconstructing $E^\alpha(i)$ and $E^\alpha(0)$ towards each P_i:** For each party P_i, every party P_j computes $f^{E^\alpha(i)}(j) = \sum_{k=0}^{2t} i^k \cdot f^{e_k^\alpha}(j)$ and $f^{E^\alpha(0)}(j) = f^{e_0^\alpha}(j)$ for each $\alpha \in [m]$. Parties invoke $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$ (Functionality 7.1) where P_j inputs $(f^{E^\alpha(0)}(j), f^{E^\alpha(1)}(j),\ldots, f^{E^\alpha(n)}(j))_{\alpha\in[m]}$.
      i. If the output is $\mathtt{discard}$, then proceed to Step 5a.
      ii. If the output is $(\mathtt{detect},\mathsf{Bad})$, then set $\mathsf{ZEROS} = \mathsf{ZEROS}\cup\mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.
      iii. Otherwise, P_i sets $f^{E^\alpha(0)}(x) = h_0^\alpha(x)$ and $f^{E^\alpha(i)}(x) = h_i^\alpha(x)$ for each $\alpha \in [m]$ such that $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$ did not output $\mathtt{kill}$.
   (b) **Verifying the product relation of each P_i and that $E^\alpha(0) = e_0^\alpha = (0,\ldots,0)$ holds:**
      i. Each P_i verifies that $E^\alpha(i)$ obtained from reconstructed polynomial $f^{E^\alpha(i)}(x)$ matches with $(f_i^{\alpha,a}(-\ell)\cdot f_i^{\alpha,b}(-\ell) - f_i^{\alpha,c}(-\ell))_{\ell\in\{0,\ldots,t/2\}}$. P_i also verifies that $f^{E^\alpha(0)}(-\ell) = 0$ holds for each $\ell \in \{0,\ldots,t/2\}$ and each $\alpha \in [m]$ for which $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$ did not output $\mathtt{kill}$. If not, then P_i broadcasts $\mathsf{complaint}(i,\beta_i)$ where β_i is the smallest α for which the verification fails.
      ii. Parties construct a binary vector $(\mathsf{pub}_1,\ldots,\mathsf{pub}_n)$ where $\mathsf{pub}_i = 1$ if P_i broadcasted $\mathsf{complaint}(i,\beta_i)$.
      iii. Parties invoke $\mathcal{F}_{\mathsf{pubRec}}$ (Functionality 6.5) five times where each P_j inputs $(\mathsf{pub}_1,\ldots,\mathsf{pub}_n)$, ZEROS in every instance and the following shares respectively in the five instances: (i) $(f^{E^{\beta_1}(1)}(j),\ldots,f^{E^{\beta_n}(n)}(j))$, (ii) $(f^{E^{\beta_1}(0)}(j),\ldots,f^{E^{\beta_n}(0)}(j))$, (iii) $(g_j^{\beta_1,a}(1),$

$\ldots, g_j^{\beta_n,a}(n))$, (iv) $(g_j^{\beta_1,b}(1), \ldots, g_j^{\beta_n,b}(n))$ and (v) $(g_j^{\beta_1,c}(1), \ldots, g_j^{\beta_n,c}(n))$. For P_i which did not broadcast $\mathsf{complaint}(i, \beta_i)$, each P_j sets the ith share to be 0 in the five instances above. For any of the above instances:

A. If the output is `discard`, then proceed to Step 5a.

B. If the output is $(\texttt{detect}, \mathsf{Bad})$, then set $\mathsf{ZEROS} = \mathsf{ZEROS} \cup \mathsf{Bad}$. If $|\mathsf{ZEROS}| > t$ then go to Step 5a. Otherwise, go to Step 2.

C. Otherwise, when the output is `proceed`, all parties set $f^{E^{\beta_i}(i)}(x), f^{E^{\beta_i}(0)}(x)$ and $f_i^{\beta_i,a}(x), f_i^{\beta_i,b}(x), f_i^{\beta_i,c}(x)$ as the respective output obtained in the corresponding five instances of $\mathcal{F}_{\mathsf{pubRec}}$ for each P_i which broadcasted $\mathsf{complaint}(i, \beta_i)$.

iv. For each P_i which complained, parties verify the checks as in Step 4(b)i using the polynomials returned by the instances of $\mathcal{F}_{\mathsf{pubRec}}$. If it does not hold for some complaining party, then go to Step 5a. Otherwise, go to Step 5b.

5. **Output:**

(a) **Discard:** All parties output $\perp$.

(b) **Successful:** Each P_i outputs $(f_i^{\alpha,a}(x), g_i^{\alpha,a}(y), f_i^{\alpha,b}(x), g_i^{\alpha,b}(y), f_i^{\alpha,c}(x), g_i^{\alpha,c}(y))$ for each $\alpha \in [m]$ for which $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$ did not output `kill` in Step 4a. Here, $\{(f_i^{\alpha,a}(-\ell), f_i^{\alpha,b}(-\ell), f_i^{\alpha,c}(-\ell))\}_{\ell \in \{0,\ldots,t/2\}}$ defines P_i's shares of the multiplication triples.

Theorem 7.7. *Protocol 7.6, $\Pi_{\mathsf{PVTS}}^{\mathsf{batched}}$, perfectly securely computes Functionality 7.5, $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$, in the presence of a malicious adversary controlling at most $t < n/3$. It requires a communication complexity of $\mathcal{O}(mn^2 \log n)$ bits over-point-to-point channels and $\mathcal{O}(n^2 \log n)$ bits broadcast, and $\mathcal{O}(1)$ rounds. Each party broadcasts at most $\mathcal{O}(n \log n)$ bits.*

Note that since (at most) n instances can be compromised (due to `kill` as output from $\mathcal{F}_{\mathsf{privRec}}^{\mathsf{batched}}$), the dealer shares $\mathcal{O}((m-n)n)$ multiplication triples. Consequently, we have that when $m - n = \mathcal{O}(n)$, the dealer generates $\mathcal{O}(n^2)$ triples at an amortized cost of $\mathcal{O}(n \log n)$ bits over point-to-point channels and $\mathcal{O}(1)$ bits of broadcast per triple.

8 Linear Perfectly Secure MPC

In this section, we first give details of the additional building blocks necessary for MPC such as reconstruction of degree-t polynomials, and Beaver triple generation. We conclude with our complete MPC protocol relying on these building blocks, the packed secret sharing (Sections 4, 5) and the verifiable triple sharing (Sections 6, 7). In the following, we use $\langle v \rangle$ to denote the degree-t Shamir-sharing of a value v among parties.

8.1 Secret Reconstruction

Since the outcome of our VSS is secrets in Shamir-shared format, we discuss how such sharing can be reconstructed efficiently. We use two standard ways of reconstruction:

Private reconstruction. Here, the secret is reconstructed privately to a specified party. This can be achieved by simply letting all the parties disclose the shares to the party who applies RS error

correction for recovering the secret. We denote this protocol as Π_{Rec}. This requires $\mathcal{O}(n \log n)$ bits communication.

Batched public reconstruction. Naïvely, reconstructing $t+1$ secrets that are Shamir-shared requires $(t+1)n$ private reconstructions (via Π_{Rec}), resulting in $O(n^3 \log n)$ communication[5]. On the other hand the batch reconstruction protocol, first presented in [26], allows parties to robustly reconstruct $t+1$ Shamir-shared values at a cost of communicating $\mathcal{O}(n^2 \log n)$ bits, ensuring an amortized cost of $\mathcal{O}(n \log n)$ bits per reconstruction.

In particular, given $\langle v_0 \rangle, \ldots, \langle v_t \rangle$, parties translate them to n sharings *non-interactively*, say $\langle v'_1 \rangle, \ldots, \langle v'_n \rangle$, using a linear error correcting code, such as Reed-Solomon code which tolerates up to t errors. To be specific, $(v'_1, \ldots, v'_n)$ can be thought of as n points on a t-degree polynomial $p(x) = \sum_{i=0}^{t} v_i x^i$. Following this, of the n sharings, one sharing $\langle v'_i \rangle$ is reconstructed towards each party P_i via private reconstruction protocol Π_{Rec} who obtains v'_i. At this stage, the parties essentially hold $\langle v_0 \rangle$. Therefore, n instances of private reconstruction enables every party to recover $p(x)$, the polynomial used to share v_0, whose coefficients are the desired output. This requires a total communication of $\mathcal{O}(n^2 \log n)$ bits. The protocol Π_{bPubRec} appears below for completeness.

Protocol 8.1: Batched Public Reconstruction Protocol – Π_{bPubRec}

Common input: The description of a field $\mathbb{F}$, n non-zero distinct elements $1, \ldots, n$.
Input: Parties hold the univariate degree-t sharings $\langle v_0 \rangle, \ldots, \langle v_t \rangle$.

1. Let $p(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_t x^t$.
2. For each P_i, parties locally compute $\langle v'_i \rangle = \langle p(i) \rangle = \langle v_0 \rangle + \langle v_1 \rangle \cdot i + \langle v_2 \rangle \cdot i^2 + \ldots + \langle v_t \rangle \cdot i^t$.
3. For each party P_i, parties invoke Π_{Rec} with $\langle v'_i \rangle$ as input to enable P_i to privately reconstruct $v'_i = p(i)$. Note that parties now hold $\langle p(0) \rangle$.
4. For each party P_i, parties invoke Π_{Rec} with $\langle p(0) \rangle$ as input to enable P_i to privately reconstruct the polynomial $p(x)$. Upon reconstructing, each P_i outputs the $t+1$ coefficients $v_0, v_1, \ldots, v_t$ of $p(x)$.

Lemma 8.2. *Protocol 8.1, Π_{bPubRec}, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and no broadcast for publicly reconstructing $\mathcal{O}(n)$ values (i.e., $\mathcal{O}(n \log n)$ bits) simultaneously in 2 rounds.*

8.2 From the PSS and VTS to MPC

We now give the road-map for our MPC. Excluding the PSS and the VTS, the existing tools are taken from [23]. Our protocol has two phases: (a) preparation of Beaver triples $(\langle a_l \rangle, \langle b_l \rangle, \langle c_l \rangle)_{l \in C}$, where C is the number of multiplication gates in circuit to be evaluated; (b) batched evaluation of the multiplication gates consuming the Beaver triples. Let us start with the latter.

Batched Beaver Multiplication. This protocol relies on the well known technique of Beaver's circuit randomization [10], which, given a pre-computed t-shared random and private multiplication triple $(\langle a \rangle, \langle b \rangle, \langle c \rangle)$, reduces the computation of $\langle xy \rangle$ from $\langle x \rangle$ and $\langle y \rangle$ to two public reconstructions. Towards this, parties first locally compute $\langle d \rangle = \langle x \rangle - \langle a \rangle$ and $\langle e \rangle = \langle y \rangle - \langle b \rangle$, followed by public

[5] Alternatively, $(t+1)n$ elements of broadcasts. Broadcasts are expensive and would require a minimum of $\mathcal{O}(n^3 \log n)$ communication and a minimum of constant expected inflation in the round complexity.

reconstruction of d and e. Since $z = xy = ((x-a)+a)((y-b)+b) = (d+a)(e+b) = de+db+ea+ab$, parties can locally compute $\langle z \rangle = \langle xy \rangle$ using the shared multiplication triple and the publicly reconstructed values d and e. Specifically, parties locally compute $\langle xy \rangle = de + d\langle b \rangle + e\langle a \rangle + \langle c \rangle$.

To leverage the efficiency benefits offered by the batch public reconstruction protocol, the protocol handles a batch of l multiplications together, each requiring 2 reconstructions. The $2l$ public reconstructions are thus batched together in groups of $t+1$ to invoke Π_{bPubRec} and ensure an amortized communication complexity of $\mathcal{O}(n \log n)$ bits per reconstruction. The resultant communication complexity of Π_{bBeaver} for handling l multiplications is $\mathcal{O}((n^2 + nl) \log n)$. The formal description appears in Protocol 8.3.

Protocol 8.3: Batched Beaver Multiplication – Π_{bBeaver}

Input: Parties hold l degree-t shared triples $(\langle a_i \rangle, \langle b_i \rangle, \langle c_i \rangle)$ for every $i \in [l]$ and l degree-t shared pairs of values $(\langle x_i \rangle, \langle y_i \rangle)$ to be multiplied.

1. For each $i \in [l]$, parties locally compute $\langle d_i \rangle = \langle x_i \rangle - \langle a_i \rangle$ and $\langle e_i \rangle = \langle y_i \rangle - \langle b_i \rangle$.
2. Let $2l = k(t+1)$. Parties execute k parallel instances of Π_{bPubRec} and publicly reconstruct $\{d_i, e_i\}$ for every $i \in [l]$.
3. For each $i \in [l]$, parties locally compute $\langle z_i \rangle = \langle x_i y_i \rangle = d_i e_i + d_i \langle b_i \rangle + e_i \langle a_i \rangle + \langle c_i \rangle$.

Lemma 8.4. *Protocol 8.3, Π_{bBeaver}, has a communication complexity of $\mathcal{O}((ln + n^2) \log n)$ bits over point-to-point channels and no broadcast for the multiplication of l pairs of shared values in 2 rounds.*

Preparing Beaver triples. This task is further phrased in two tasks. First, a verifiable triple sharing (VTS) is used to make a dealer Shamir-share three values (a, b, c) such that $c = ab$. Second, a triple extraction protocol that takes n verified triples, ith one contributed by P_i and extracts $t/2$ triples that are unknown to the adversary.

Verifiable Triple Sharing. During this phase, each party shares verified multiplication triples which are subsequently consumed to extract random triples (unknown to any party) required for circuit evaluation via Beaver's trick [10]. Towards that, each party invokes $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$ (Functionality 7.5) in parallel to generate the desired number of triples in a batched manner. The exact number of triples that a party has to share depends on the number of multiplication gates in the circuit, and we provide a cost analysis in Lemma 8.10. We thus have that there are n *parallel* instances of $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$, where in each instance a party broadcasts $n \log n$ bits. Considering all the n instances, each party broadcasts $n^2 \log n$ in parallel. For this, we use the parallel broadcast primitive $\mathcal{F}_{\mathsf{BC}}^{\mathsf{parallel}}$ (Functionality 3.6). Thus, for the parallel batched verifiable triple sharing, we have the following.

Lemma 8.5. *Parallel batched verifiable triple sharing requires a communication of $\mathcal{O}(mn^3 \log n)$ bits over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast in $\mathcal{O}(1)$ rounds. Each party broadcasts $\mathcal{O}(n^2 \log n)$ bits in parallel.*

Using the broadcast realisation of [1], we have that the parallel batched verifiable triple sharing requires $\mathcal{O}(mn^3 \log n + n^4 \log n)$ bits over point-to-point channels.

At the termination of this phase, we have n sets of triples, one shared by each party. Note that for each corrupt party which was discarded during $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$, all parties assume default sharing of

some publicly known triples. Given this, the next phase "merges" the triples shared by, and known to each party respectively to extract random triples.

Triple Extraction. Our last component is a triple extraction protocol that consumes one (verified) multiplication triple, say $(\langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle)$, shared by each party P_i in the prior stage and extracts $\mathcal{O}(n)$ random triples not known to any party at the cost of $\mathcal{O}(n^2)$ point to point communication. In particular, the protocol extracts $h+1-t$ multiplication triples, where $h = \lfloor \frac{n-1}{2} \rfloor$ using n triples, one shared by each party. At a high level, the protocol proceeds as follows. First, the parties "transform" the n random shared triples $(\langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle)$ for each $i \in [n]$ into n correlated triples $(\langle x_i\rangle, \langle y_i\rangle, \langle z_i\rangle)$ for every $i \in [n]$ such that the values $\{x_i, y_i, z_i\}_{i\in[n]}$ lie on the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively where $X(\cdot)\cdot Y(\cdot) = Z(\cdot)$. Specifically, for each $i \in [n]$, it holds that $X(i) = x_i, Y(i) = y_i$ and $Z(i) = z_i$ where $1, \ldots, n$ are publicly known distinct elements from $\mathbb{F}$. Furthermore, the transformation ensures that the adversary knows $\{x_i, y_i, z_i\}$ only if P_i is corrupt. This implies that the adversary may know t points on each of the polynomials $X(\cdot), Y(\cdot)$ and $Z(\cdot)$ of degree h, h and $2h$ respectively, thus guaranteeing a degree of freedom of $h+1-t = t/2$ in $X(\cdot), Y(\cdot)$ (and thus $Z(\cdot)$). Parties thus output the shared evaluation of these polynomials at $h+1-t$ publicly known points $\beta_1, \ldots, \beta_{h+1-t}$ as the extracted shared multiplication triples.

The transformation itself works as follows. The parties simply set $x_i = a_i, y_i = b_i, z_i = c_i$ for $i \in \{1, \ldots, h+1\}$. Next, $\langle x_i\rangle$ and $\langle y_i\rangle$ for every $i \in \{h+2, \ldots, n\}$ can be computed *non-interactively* by taking linear combination of $\{x_i, y_i\}_{i\in[h+1]}$. Following this, $\langle z_i\rangle$ for every $i \in \{h+2, \ldots, n\}$ is computed using Beaver's trick where the inputs are $\langle x_i\rangle$ and $\langle y_i\rangle$ and the triple $(\langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle)$. Clearly, if P_i is corrupt then x_i, y_i, z_i is known to the adversary as claimed. To conclude, we note that triple extraction reduces to running a batch of $\mathcal{O}(n)$ Beaver multiplications which requires $O(n^2 \log n)$ bits communication using Π_{bPubRec}. The formal description appears in Protocol 8.6.

Protocol 8.6: Triple Extraction – $\Pi_{\mathsf{tripleExt}}$

Common input: The description of a field $\mathbb{F}$, $n = 2h+1$ non-zero distinct elements $1, \ldots, n$ and $h+1-t$ non-zero distinct elements $\beta_1, \ldots, \beta_{h+1-t}$.
Input: Parties hold the degree-t shared triples $(\langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle)$ for every $i \in [n]$ such that (a_i, b_i, c_i) is known to party P_i.

1. For each $i \in [h+1]$, parties locally set $\langle x_i\rangle = \langle a_i\rangle$, $\langle y_i\rangle = \langle b_i\rangle$ and $\langle z_i\rangle = \langle c_i\rangle$.
2. Let $X(\cdot)$ and $Y(\cdot)$ be the degree-h polynomials defined by the points $\{x_i\}_{i\in[h+1]}$ and $\{y_i\}_{i\in[h+1]}$ respectively such that $X(i) = x_i$ and $Y(i) = y_i$ for all $i \in [h+1]$.
3. For each $i \in \{h+2, \ldots, n\}$, parties locally compute $\langle x_i\rangle = \langle X(i)\rangle$ and $\langle y_i\rangle = \langle Y(i)\rangle$.
4. Parties invoke Π_{bBeaver} with $\{\langle x_i\rangle, \langle y_i\rangle, \langle a_i\rangle, \langle b_i\rangle, \langle c_i\rangle\}_{i\in\{h+2,\ldots,n\}}$ and obtain $\{\langle z_i\rangle\}_{i\in\{h+2,\ldots,n\}}$ where $z_i = x_i y_i$ for every $i \in \{h+2, \ldots, n\}$.
5. Let $Z(\cdot)$ be the degree-$2h$ polynomial defined by the points $\{z_i\}_{i\in[n]}$ such that $Z(i) = z_i$ for all $i \in [n]$.
6. Parties locally compute $\langle \mathbf{a}_i\rangle = \langle X(\beta_i)\rangle$, $\langle \mathbf{b}_i\rangle = \langle Y(\beta_i)\rangle$ and $\langle \mathbf{c}_i\rangle = \langle Z(\beta_i)\rangle$ for every $i \in [h+1-t]$.

Lemma 8.7. *Protocol 8.6, $\Pi_{\mathsf{tripleExt}}$, has a communication complexity of $\mathcal{O}(n^2 \log n)$ bits over point-to-point channels and no broadcast for sharing $\mathcal{O}(n)$ random multiplication triples in 2 rounds.*

8.3 The MPC Protocol

The protocol Π_{MPC} and the corresponding functionality $\mathcal{F}_{\mathsf{MPC}}$ are provided below. As described, at a high level, the protocol is divided into the following two phases:

1. *Beaver triple generation:* In this phase, parties generate C number of degree-t Shamir-shared multiplication triples where, C denotes the number of multiplication gates in the circuit. Towards that, each party first generates triples using our VTS protocol. Subsequently, a triple extraction protocol "merges" the triples generated by all parties and "extracts" random triples (not known to any party) which will be consumed in the second phase. For sufficiently large circuits, specifically for circuits of size $\Omega(n^3)$, this phase incurs an amortized cost of $\mathcal{O}(n \log n)$ bits point-to-point communication per triple.
2. *Circuit computation:* Upon sharing of inputs by the input holding parties, in this phase the computation of the circuit proceeds by parties performing shared evaluation of the circuit. Since our sharing is linear, the linear operations of addition and multiplication by a constant are local. For multiplication of shared values, parties consume the Beaver triples generated in the prior phase. This is followed by the reconstruction of the outputs to the designated parties to complete the circuit evaluation.

Functionality 8.8: MPC – $\mathcal{F}_{\mathsf{MPC}}$

Input: Each P_i holds input $x_i \in \mathbb{F} \cup \{\perp\}$.
Common Input: An n-party function $f(x_1, \ldots, x_n)$.

1. Each P_i sends x_i to the functionality. For any P_i, if x_i is outside the domain or P_i did not send any input, set x_i to a predetermined default value.
2. Compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ and send y_i to P_i for every $i \in [n]$.

Protocol 8.9: MPC – Π_{MPC}

Common input: The description of a circuit, the field $\mathbb{F}$, n non-zero distinct elements $1, \ldots, n$ and a parameter h where $n = 2h + 1$. Let $m = \lceil \frac{C}{h+1-t} \rceil$.
Input: Parties hold their inputs (belonging to $\mathbb{F} \cup \{\perp\}$) to the circuit.
(Beaver triple generation:)

1. Each P_i chooses $m+n(t/2+1)$ random multiplication triples and executes $\Pi_{\mathsf{PVTS}}^{\mathsf{batched}}$ (Section 7, Protocol 7.6) batching $\lceil \frac{m}{(t/2+1)} \rceil + n$ instances each with $t/2+1$ triples. Let $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for $j \in [m]$ denote the triples shared by P_i.
2. Parties execute m instances of $\Pi_{\mathsf{tripleExt}}$ (Protocol 8.6) with $(\langle a_i^j \rangle, \langle b_i^j \rangle, \langle c_i^j \rangle)$ for every $i \in [n]$ as the input for the j^{th} instance. Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ for $i \in [C]$ denote the random multiplication triples generated.

(Circuit computation:)

1. **(Input)** Each party P_i holding k_i inputs to the circuit executes $\Pi_{\mathsf{PSS}}^{\mathsf{batched}}$ (Section 5) batching $\lceil \frac{k_i}{t/2+1} \rceil$ instances to share its inputs.

2. **(Linear Gates)** Parties locally apply the linear operation on their respective shares of the inputs.
3. **(Multiplication Gates)** Let $(\langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle)$ be the multiplication triple associated with the i^{th} multiplication gate with shared inputs $(\langle x_i \rangle, \langle y_i \rangle)$. Parties invoke Π_{bBeaver} (Protocol 8.3) with $\{\langle x_i \rangle, \langle y_i \rangle, \langle \mathbf{a}_i \rangle, \langle \mathbf{b}_i \rangle, \langle \mathbf{c}_i \rangle\}$ for all gates i at the same layer of the circuit and obtain the corresponding $\langle z_i \rangle$ as the output sharing for every gate i.
4. **(Output)** For each output gate j with the associated sharing $\langle v_j \rangle$, parties execute Π_{Rec} towards every party P_i who is supposed to receive the output v_j.

Theorem 8.10. *Let $t < n/3$. Protocol 8.9 securely implements $\mathcal{F}_{\mathsf{MPC}}$ (Functionality 8.8) and has a communication complexity of $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ bits over point to point channels and $\mathcal{O}(n^3 \log n)$ bits broadcast for evaluating a circuit with C gates and depth D in expected $\mathcal{O}(D)$ rounds. Every party broadcasts $\mathcal{O}(n^2 \log n)$ bits.*

Proof. The circuit evaluation requires C random multiplication triples. We analyse the cost of the two phases separately.

Beaver triple generation. Note that the triple extraction protocol ($\Pi_{\mathsf{tripleExt}}$) generates $\mathcal{O}(n)$ (specifically, $h+1-t$ where $h = \lfloor \frac{n-1}{2} \rfloor$) such random triples by consuming one verified multiplication triple per party. Thus, we need $\mathcal{O}(C/n)$ instances of the triple extraction protocol, which incurs a cost of $\mathcal{O}(Cn \log n + n^2 \log n)$ bits over point-to-point channels. Each instance of triple extraction consumes one verified multiplication triple per party. This requires each party to ensure the sharing of a set of $\mathcal{O}(C/n)$ verified multiplication triples. Since our verified triple sharing packs $\mathcal{O}(n)$ (specifically, $t/2 + 1$) triples in one instance, this corresponds to each party parallelly running $\mathcal{O}(C/n^2) + n$ instances of packed verifiable triple sharing in a batched manner, where the additional n accounts for the (at most) n instances that the adversary can compromise in $\mathcal{F}_{\mathsf{PVTS}}^{\mathsf{batched}}$. This phase incurs a cost of $\mathcal{O}(Cn \log n + n^4 \log n)$ bits of communication over point-to-point channels and $\mathcal{O}(n^3 \log n)$ bits of broadcast, where every party broadcasts $\mathcal{O}(n^2 \log n)$ in parallel.

Circuit computation. In this phase, parties batched the multiplication gates at the same level in the circuit and invoke the batched Beaver multiplication protocol (Π_{bBeaver}) for evaluating them. Given C_i is the number of gates per level of the circuit, this stage incurs a cost of $\mathcal{O}(C_i \cdot n \log n + n^2 \log n)$ bits communication over point-to-point channels. Consequently, we have that the circuit computation requires $\sum_{i=1}^{D} \mathcal{O}(C_i \cdot n \log n + n^2 \log n) = \mathcal{O}(Cn \log n + Dn^2 \log n)$ bits communication over point-to-point channels. □

References

[1] Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed vss. In: TCC (2022)

[2] Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. In: Theory of Cryptography (2021)

[3] Abraham, I., Dolev, D., Halpern, J.Y.: An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In: PODC '08 (2008)

[4] Anirudh, C., Choudhury, A., Patra, A.: A survey on perfectly-secure verifiable secret-sharing. Cryptology ePrint Archive (2021)

[5] Asharov, G., Cohen, R., Shochat, O.: Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In: 3rd Conference on Information-Theoretic Cryptography, ITC 2022 (2022)

[6] Asharov, G., Lindell, Y.: A full proof of the bgw protocol for perfectly secure multiparty computation. Journal of Cryptology (2017)

[7] Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any $t < n/3$. In: Advances in Cryptology - CRYPTO 2011 (2011)

[8] Bangalore, L., Choudhury, A., Patra, A.: Almost-surely terminating asynchronous byzantine agreement revisited. In: 2018 ACM Symposium on Principles of Distributed Computing, PODC. ACM (2018)

[9] Bangalore, L., Choudhury, A., Patra, A.: The power of shunning: Efficient asynchronous byzantine agreement revisited*. J. ACM (2020)

[10] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Annual International Cryptology Conference (1991)

[11] Beerliova-Trubiniova, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3. pp. 305–328 (2006)

[12] Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure mpc with linear communication complexity. In: Theory of Cryptography Conference (2008)

[13] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Annual ACM Symposium on Theory of Computing (1988)

[14] Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings. pp. 663–680 (2012)

[15] Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. In: Computer science (1992)

[16] Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. (2000)

[17] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001)

[18] Canetti, R., Damgård, I., Dziembowski, S., Ishai, Y., Malkin, T.: On adaptive vs. non-adaptive security of multiparty protocols. In: Advances in Cryptology - EUROCRYPT 2001 (2001)

[19] Canetti, R., Damgard, I., Dziembowski, S., Ishai, Y., Malkin, T.: Adaptive versus non-adaptive security of multi-party protocols. Journal of Cryptology (2004)

[20] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing (1988)

[21] Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In: 26th Annual Symposium on Foundations of Computer Science (1985)

[22] Choudhury, A.: Protocols for Reliable and Secure Message Transmission. Ph.D. thesis, Citeseer (2010)

[23] Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Transactions on Information Theory (2016)

[24] Coan, B.A., Welch, J.L.: Modular construction of nearly optimal byzantine agreement protocols. In: ACM Symposium on Principles of distributed computing (1989)

[25] Cramer, R., Damgård, I., Maurer, U.: General secure multi-party computation from any linear secret-sharing scheme. In: International Conference on the Theory and Applications of Cryptographic Techniques (2000)

[26] Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference (2007)

[27] Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: 20th Annual ACM Symposium on Theory of Computing (1988)

[28] Feldman, P.N.: Optimal algorithms for Byzantine agreement. Ph.D. thesis, Massachusetts Institute of Technology (1988)

[29] Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Information Processing Letters (1982)

[30] Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing (1992)

[31] Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: ACM symposium on Principles of distributed computing (1998)

[32] Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional mpc with guaranteed output delivery. In: Annual International Cryptology Conference (2019)

[33] Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority mpc. In: Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II. pp. 618–646 (2020)

[34] Hirt, M., Maurer, U., Przydatek, B.: Efficient secure multi-party computation. In: International conference on the theory and application of cryptology and information security (2000)

[35] Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: Annual International Cryptology Conference (2006)

[36] Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: 38th Annual ACM Symposium on Theory of Computing (2006)

[37] MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes, vol. 16. Elsevier (1977)

[38] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: ACM Symposium on Theory of Computing (1989)