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

Ittai Abraham[1], Gilad Asharov[2], Shravani Patil[3], and Arpita Patra[3]

[1] VMWare Research
iabraham@vmware.com
[2] Department of Computer Science, Bar-Ilan University, Israel
Gilad.Asharov@biu.ac.il
[3] Indian Institute of Science, Bangalore, India
{shravanip,arpita}@iisc.ac.in

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 $\Theta(n+D)$ 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 $\mathcal{O}(1)$ 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 $\mathcal{O}(n)$ per secret. The best prior required $\mathcal{O}(n^3)$ communication per secret.

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 [17,5]) and remain secure under universal composition [34].

The seminal protocols of Ben-Or, Goldwasser, and Wigderson [13], and Chaum, Crépeau and Damgård [19] led the foundations of this setting. Since then, there are, in general, two families of protocols:

1. **Efficient but slow:** These protocols [32,12,30] ([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 [15,23], 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 [13,19,29,24,7,2] 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 [15,23] for implementing the broadcast, the number of rounds is increased to $\Omega(n + D)$.

Our Main Result

Our main result is that the best of both families is possible to achieve simultaneously. 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 [12,30], 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 [30], 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 [15,23] 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 [28]. 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 [36] (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 [20] 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 [13,26] 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 [32,12,30] in perfectly-secure MPC is based on the *player elimination framework* (introduced by Hirt, Maurer and Przydatek [32]). 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 some part of 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: (a) *global detection* – wherein a set of parties is excluded from the computation. Unlike [32,12,30], in our case, honest parties are never discarded; (b) *public individual detection* – wherein each party has its own conflict set that is publicly known to all. While a similar mechanism, referred to as 'dispute control' has been used in [11,14,31], these works achieve *statistical* security in the honest majority setting with $\mathcal{O}(n)$ rounds overhead similar to the player-elimination framework; (c) *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. 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 [15,23] 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 [25], and Katz and Koo [33]) 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 [27].

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 it has sent that party. Therefore, an honest dealer never makes the shares of an 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,[4] 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

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

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 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 CONFLICTS to be 0 in all instances. Thus, the dealer uses a broadcast of $\mathcal{O}(n \log n)$ bits, i.e., the set CONFLICTS, and by restarting the protocol it made the shares of parties in 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 CONFLICTS.

Before we proceed, let's highlight what guarantees we have so far: when the dealer is honest, then all the parties in 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 is not in CONFLICTS sends 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 $2t + t/4 + 1$ "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. In this case, P_j broadcasts a complaint $\mathsf{complaint}(j)$, insisting that its shares 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 $j \in$ 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 par-

ticular 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 [22], 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 [22] 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 [22], 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 [22], 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 readers to the full version 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 $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ such that for every i it holds that $c_i = a_i b_i$. Towards this end, the dealer plants $\boldsymbol{a}$ into some bivariate polynomial $A(x, y)$ using our verifiable secret sharing scheme. It plants $\boldsymbol{b}$ into $B(x, y)$ and $\boldsymbol{c}$ 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 $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, 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 each of degree $t + t/4$ in x and t in y, 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 $E(x, y)$ with degree $t + t/4$ in both x and 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.[5] 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 each $j \in [n]$) 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. 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 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.

[5] 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.

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 Section 6 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 multiplication triplets with a dealer (Section 6), and conclude with the MPC protocol in Section 7. Due to lack of space, the proofs and some constructions are deferred to the full version.

3 Preliminaries

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) in the full version. Owing to the results of [18], this guarantees adaptive security with inefficient simulation. We derive universal composability [16] using [34].

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 $\mathcal{P}$.

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 $S(-a, -b) = \mathsf{SECRETS}(a, b)$.

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 3.1 ([21,35])**.** *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, see the full version for details.

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

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 3.3 ([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 ZEROS $\subset [n]$. For an honest dealer, it holds that 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 $i \in$ 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 $\mathsf{CONFLICTS} \cap \mathsf{ZEROS} = \emptyset$.[6] 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 $\mathsf{CONFLICTS}$.
   (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 $\mathsf{CONFLICTS}$ 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 $i \in \mathsf{CONFLICTS}$.

[6] 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: $\mathsf{CONFLICTS} = \mathsf{CONFLICTS} \setminus \mathsf{ZEROS}$.

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* $t < n/3$.

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 CONFLICTS is at least $(n - t/2) + d$. The goal of this step is to reconstruct the g-polynomials for the parties in 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:**[7] 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 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 `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 $(\texttt{proceed}, S(x, j), S(j, y))$.

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

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

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 HAVE-SHARES $= [n] \setminus (\text{ZEROS} \cup \text{CONFLICTS})$.
2. For every $j \in$ CONFLICTS:
   (a) Each party P_i for $i \in$ HAVE-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 $i \in$ ZEROS, consider (i, u_i) with $u_i = 0$. Given all $(i, u_i)_{i \notin \text{CONFLICTS}}$, 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 Theorem 3.1). 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 complaint(j) and every party P_i for $i \in$ HAVE-SHARES broadcasts reveal$(i, j, f_i(j))$.
3. The dealer sets Bad $= \emptyset$. For each 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) $|\text{ZEROS} \cup \text{CONFLICTS} \cup \text{Bad}| \leq t$; (ii) Bad $\subset$ HAVE-SHARES. The parties go to Step 6b if $|\text{Bad}| > t/2$.
5. Otherwise, for every $j \in$ CONFLICTS, if complaint(j) was broadcasted, then the parties consider all the points $R_j = \{(i, u_i)\}$ such that reveal(i, j, u_i) was broadcasted in Step 2b, and $i \in \text{HAVE-SHARES} \setminus \text{Bad}$, or $u_i = 0$ if $i \in$ 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 `discard`.
   (b) Detect: Output $(\texttt{detect}, \text{Bad})$.
   (c) Proceed: Each party $j \in$ CONFLICTS outputs $(\texttt{proceed}, \perp, g_j(y))$. All other parties P_j with $j \notin$ CONFLICTS output $(\texttt{proceed}, f_j(x), g_j(y))$.

Lemma 4.6. *Protocol 4.5, $\Pi_{\text{rec-g}}$, perfectly securely computes Functionality 4.4, $\mathcal{F}_{\text{rec-g}}$, 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.*

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 $\mathcal{F}_{\mathsf{rec\text{-}f}}$), and the protocol (denoted by $\Pi_{\mathsf{rec\text{-}f}}$), as well as the proof of the following lemma are given in the full version.

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

4.4 Putting Everything Together: Packed Secret Sharing

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

Functionality 4.8: 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 ZEROS $\subset [n]$ such that $|\mathsf{ZEROS}| \leq t$. If the dealer is honest then it is guaranteed that ZEROS $\subseteq I$.
- **Honest dealer:** The dealer sends 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 $i \in$ 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) 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 $i \in$ 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. 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 4.9: 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 SECRETS. 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 SECRETS, under the constraint that for every $i \in$ 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 (`detect`, CONFLICTS) then set ZEROS = ZEROS $\cup$ CONFLICTS. If $|\mathsf{ZEROS}| > t$ then proceed to Step 4a. Otherwise, go back to Step 1a.
      iii. If the output is (`proceed`, $f_i(x), g_i(y)$, CONFLICTS), then proceed to the next step. Note that it must hold that (a) for parties $i \in$ 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 (ZEROS, CONFLICTS, $f_i(x), g_i(y)$).
   (a) If the output is `discard`, then proceed to Step 4a.
   (b) If the output is (`detect`, Bad) then set ZEROS = ZEROS $\cup$ Bad. If $|\mathsf{ZEROS}| > t$ then discard and proceed to Step 4a. Otherwise, go back to Step 1a.
   (c) Otherwise, the output is (`proceed`, $f_i(x), g_i(y)$) where every party P_i with $i \in$ CONFLICTS has $g_i(y) \neq \perp$, then proceed to the next step.
3. **Reconstruct the f-polynomials:** The parties invoke Functionality $\mathcal{F}_{\mathsf{rec\text{-}f}}$, where each party P_i inputs (ZEROS, 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 (`detect`, Bad) then set ZEROS = ZEROS $\cup$ Bad. If $|\mathsf{ZEROS}| > t$ then discard and go to Step 4a. Otherwise, go back to Step 1a.
   (c) Otherwise, let (`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 4.10. *Let $t < n/3$ and $d \leq t/4$. Protocol 4.9, Π_{PSS}, perfectly securely computes Functionality 4.8, $\mathcal{F}_{\mathsf{PSS}}$, in the $(\mathcal{F}_{\mathsf{ShareAttempt}}, \mathcal{F}_{\mathsf{rec\text{-}g}}, \mathcal{F}_{\mathsf{rec\text{-}f}})$-hybrid model, in the presence of a malicious adversary, controlling at most $t < n/3$.*

Communication and Efficiency Analysis. We conclude the following lemma, proven in the full version:

Lemma 4.11. *Let $t < n/3$ and $d \leq t/4$. There exists a protocol that implements Functionality 4.8, 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.*

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 $\mathsf{ZEROS} \subset [n]$. 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 $\mathsf{complaint}(i, j, f_i(j), g_i(j), \alpha)$ where $\alpha \in \{1, \ldots, m\}$. Moreover, if a party broadcasts $\mathsf{complaint}(i, j, u_i, v_i)$ for $j \in \mathsf{ZEROS}$, then the dealer must add P_i to $\mathsf{CONFLICTS}$. Thus, there is no need for P_i to broadcast such a complaint in each instance that it sees inconsistency with P_j for $j \in \mathsf{ZEROS}$, 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 $\mathsf{CONFLICTS}$, exactly as we have in single instance.

Batched reconstruction of g polynomials in $\mathsf{CONFLICTS}$. 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 $\mathcal{F}_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$) and the full specification of the protocol (denoted as $\Pi_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$) are given in the full version, as well as the proof of the following lemma:

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

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 5.2: 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 5.3. $\Pi_{\mathsf{PSS}}^{\mathsf{batched}}$ *securely computes* $\mathcal{F}_{\mathsf{PSS}}^{\mathsf{batched}}$ *(Functionality 5.2). 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 5.4: Detectable Reconstruction for Batched and Packed Secret Sharing – $\mathcal{F}_{\mathsf{PSS\text{-}Rec}}^{\mathsf{batched}}$

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 $\Pi_{\mathsf{rec\text{-}g}}^{\mathsf{batched}}$:

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 Theorem 3.1). 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 $(\mathtt{detect}, \mathsf{Bad})$ 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 full version) 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 5.2 and 5.4, 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 5.4 and 5.4, 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 and Batched 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 full version, we show how to implement such packed verifiable triple sharing. We will also present the batched version, denoted as $\Pi^{\mathsf{batched}}_{\mathsf{PVTS}}$, 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, and we refer the reader to the full version for further details.

7 The MPC Protocol

We now describe our complete MPC protocol as a composition of the building blocks, the PSS (Sections 4, 5) and the VTS (Section 6) protocols, as well as other building blocks from the literature (e.g., triplets extractor ($\Pi_{\mathsf{tripleExt}}$) and batch Beaver multiplication (Π_{bBeaver}), see full version). The protocol Π_{MPC} and the corresponding functionality $\mathcal{F}_{\mathsf{MPC}}$ are provided below. 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 7.1: 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 7.2: 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 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 $\Pi_{\mathsf{tripleExt}}$ 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} 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 7.3. *Let $t < n/3$. Protocol 7.2 securely implements $\mathcal{F}_{\mathsf{MPC}}$ (Functionality 7.1) 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.*

Acknowledgements

Gilad Asharov is 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. Shravani Patil would like to acknowledge the support of DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020-2025. Arpita Patra would like to acknowledge the support of DST National Mission on Interdisciplinary Cyber-Physical Systems (NM-ICPS) 2020-2025, Google India Faculty Award, and JPM Faculty Research Award.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed vss. In: TCC (2022). https://doi.org/10.1007/978-3-031-22318-1_14
2. Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. In: Theory of Cryptography (2021). https://doi.org/10.1007/978-3-030-90453-1_3
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). https://doi.org/10.1145/1400751.1400804
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). https://doi.org/10.1007/s00145-015-9214-4
7. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any $t < n/3$. In: Advances in Cryptology - CRYPTO 2011 (2011). https://doi.org/10.1007/978-3-642-22792-9_14
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). https://doi.org/10.1145/3212734.3212735
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). https://doi.org/10.1007/3-540-46766-1_34
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). https://doi.org/10.1007/11681878_16
12. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure mpc with linear communication complexity. In: Theory of Cryptography Conference (2008). https://doi.org/10.1007/978-3-540-78524-8_13
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). https://doi.org/10.1145/62212.62213
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). https://doi.org/10.1007/978-3-642-32009-5_39
15. Berman, P., Garay, J.A., Perry, K.J.: Bit optimal distributed consensus. In: Computer science (1992)
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS (2001). https://doi.org/10.1109/SFCS.2001.959888
17. 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). https://doi.org/10.1007/3-540-44987-6_17
18. Canetti, R., Damgard, I., Dziembowski, S., Ishai, Y., Malkin, T.: Adaptive versus non-adaptive security of multi-party protocols. Journal of Cryptology (2004). https://doi.org/10.1007/s00145-004-0135-x
19. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th Annual ACM Symposium on Theory of Computing (1988). https://doi.org/10.1145/62212.62214
20. 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). https://doi.org/10.1109/SFCS.1985.64

21. Choudhury, A.: Protocols for Reliable and Secure Message Transmission. Ph.D. thesis, Citeseer (2010)
22. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Transactions on Information Theory (2016)
23. Coan, B.A., Welch, J.L.: Modular construction of nearly optimal byzantine agreement protocols. In: ACM Symposium on Principles of distributed computing (1989). https://doi.org/10.1145/72981.73002
24. 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). https://doi.org/10.1007/3-540-45539-6_22
25. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: 20th Annual ACM Symposium on Theory of Computing (1988). https://doi.org/10.1145/62212.62225
26. Feldman, P.N.: Optimal algorithms for Byzantine agreement. Ph.D. thesis, Massachusetts Institute of Technology (1988)
27. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Information Processing Letters (1982)
28. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing (1992). https://doi.org/10.1145/129712.129780
29. 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). https://doi.org/10.1145/277697.277716
30. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional mpc with guaranteed output delivery. In: Annual International Cryptology Conference (2019). https://doi.org/10.1007/978-3-030-26951-7_4
31. 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). https://doi.org/10.1007/978-3-030-56880-1_22
32. 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). https://doi.org/10.1007/3-540-44448-3_12
33. Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: Annual International Cryptology Conference (2006). https://doi.org/10.1007/11818175_27
34. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: 38th Annual ACM Symposium on Theory of Computing (2006). https://doi.org/10.1145/1132516.1132532
35. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes, vol. 16. Elsevier (1977)
36. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: ACM Symposium on Theory of Computing (1989). https://doi.org/10.1145/73007.73014