Perfectly-Secure Synchronous MPC with Asynchronous Fallback Guarantees*

Ananya Appan[†] Anirudh Chandramouli[‡] Ashish Choudhury[§]

Abstract

Secure *multi-party computation* (MPC) is a fundamental problem in secure distributed computing. An MPC protocol allows a set of n mutually distrusting parties to carry out any joint computation of their private inputs, without disclosing any additional information about their inputs. MPC with *information-theoretic* security (also called *unconditional security*) provides the strongest security guarantees and remains secure even against *computationally unbounded* adversaries. *Perfectly-secure* MPC protocols is a class of information-theoretically secure MPC protocols, which provides all the security guarantees in an *error-free* fashion. The focus of this work is perfectly-secure MPC. Known protocols are designed *assuming* either a *synchronous* or *asynchronous* communication network. It is well known that perfectly-secure *synchronous* MPC protocol is possible as long as adversary can corrupt any $t_s < n/3$ parties. On the other hand, perfectly-secure *asynchronous* MPC protocol can tolerate up to $t_a < n/4$ corrupt parties. A natural question is does there exist a *single* MPC protocol for the setting where the parties are *not aware* of the exact network type and which can tolerate up to $t_s < n/3$ corruptions in a synchronous network and up to $t_a < n/4$ corruptions in an *asynchronous* network. We design such a *best-of-both-worlds* perfectly-secure MPC protocol, provided $3t_s + t_a < n$ holds.

For designing our protocol, we design two important building blocks, which are of independent interest. The first building block is a best-of-both-worlds *Byzantine agreement* (BA) protocol tolerating $t < n/3$ corruptions and which remains secure, *both* in a synchronous as well as asynchronous network. The second building block is a polynomial-based best-of-both-worlds *verifiable secret-sharing* (VSS) protocol, which can tolerate up to t_s and t_a corruptions in a *synchronous* and in an *asynchronous* network respectively.

Keywords: Perfect security, MPC, Verifiable Secret Sharing, Byzantine Agreement, Synchronous Network, Asynchronous Network.

1 Introduction

Consider a set of n mutually distrusting parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, where each P_i has some private input. The distrust among the parties is modeled as a centralized adversary, who can control

*A preliminary version of this article was published as an extended abstract in PODC 2022 [5]. This is the full and elaborate version, with complete proofs.

[†]The work was done when the author was a student at the International Institute of Information Technology, Bangalore India.

[‡]The work was done when the author was a student at the International Institute of Information Technology, Bangalore India. The author would like to thank Google Research for travel support to attend and present the preliminary version of the paper at PODC 2022.

[§]International Institute of Information Technology, Bangalore India. Email: ashish.choudhury@iiitb.ac.in. This research is an outcome of the R & D work undertaken in the project under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation (formerly Media Lab Asia). The author is also thankful to the Electronics, IT & BT Government of Karnataka for supporting this work under the CIET project.

any t out of the n parties in a *Byzantine* (malicious) fashion and force them to behave arbitrarily during the execution of any protocol. An MPC protocol [49, 39, 14, 24, 47] allows the parties to securely compute any known function of their private inputs, such that the *honest* parties (who are not under adversary's control) obtain the correct output, irrespective of the behaviour of the adversary. Moreover, adversary does not learn any additional information about the inputs of the honest parties, beyond what can be revealed by the function output and the inputs of the corrupt parties. If the adversary is *computationally bounded* then the notion of security achieved is called *conditional security* (also known as *cryptographic security*) [49, 39, 36]. On the other hand, *unconditionally secure* protocols (also known as *information-theoretically* secure protocols) provide security against *computationally unbounded* adversaries [14, 24]. Unconditionally secure protocols provide *ever-lasting* security, as their security is *not* based on any computational-hardness assumptions. Moreover, compared to conditionally secure protocols, the protocols are simpler and faster by several order of magnitude, as they are based on very simple operations, such as polynomial interpolation and polynomial evaluation over finite fields. Unconditionally secure protocols can be further categorized as *perfectly-secure* MPC protocols [14, 36, 29, 11, 40, 2], where all security properties are achieved in an *error-free* fashion. On the other hand, *statistically-secure* MPC protocols [47, 28, 9, 15, 40] allow for a negligible error in the achieved security properties.

Traditionally, MPC protocols are designed *assuming* either a *synchronous* or *asynchronous* communication model. In *synchronous* MPC (SMPC) protocols, parties are assumed to be synchronized with respect to a global clock and there is a *publicly-known* upper bound on message delays. Any SMPC protocol operates as a sequence of communication *rounds*, where in each round, every party performs some computation, sends messages to other parties and receives messages sent by the other parties, in that order. Consequently, if during a round a receiving party *does not* receive an expected message from a designated sender party by the end of that round, then the receiving party has the assurance that the sender party is definitely *corrupt*. Though synchronous communication model is highly appealing in terms of its simplicity, in practice, it might be very difficult to guarantee such strict time-outs over the channels in real-world networks like the Internet. Such networks are better modeled through the *asynchronous* communication model [21].

An *asynchronous* MPC (AMPC) protocol operates over an *asynchronous* network, where the messages can be arbitrarily, yet finitely delayed. The only guarantee in the model is that every sent message is *eventually* delivered. Moreover, the messages *need not* be delivered in the same order in which they were sent. Furthermore, to model the worst case scenario, the sequence of message delivery is assumed to be under the control of the adversary. Unlike SMPC protocols, the protocol execution in an AMPC protocol occurs as a sequence of *events*, which depend upon the order in which the parties receive messages. Comparatively, AMPC protocols are more challenging to design than SMPC protocols. This is because inherently, in any AMPC protocol, a receiving party *cannot* distinguish between a *slow* sender party (whose messages are arbitrarily delayed in the network) and a *corrupt* sender party (who does not send any messages). Consequently, in any AMPC protocol with up to t_a corruptions, at any stage of the protocol, no party can afford to receive messages from *all* the parties. This is because the corrupt parties may never send their messages and hence the wait could turn out be an endless wait. Hence, as soon as a party receives messages from any subset of $n - t_a$ parties, it has to proceed to the next stage of the protocol. However, in this process, messages from up to t_a potentially slow, but honest parties, may get ignored. In fact, in *any* AMPC protocol, it is *impossible* to ensure that the inputs of all *honest* parties are considered for the computation and inputs of up to t_a (potentially honest) parties may have to be ignored, since waiting for all n inputs may turn out to be an endless wait. The advantage of AMPC protocols over SMPC protocols is that the time taken to produce the output depends upon the *actual speed* of the underlying network. In more detail, for an SMPC protocol, the participants have to *pessimistically*

set the global delay Δ on the message delivery to a large value to ensure that the messages sent by every party at the beginning of a round reach to their destination within time Δ. But if the actual delay δ in the network is such that $\delta << \Delta$, then the protocol *fails* to take advantage of the faster network and its running time will be still proportional to Δ.

The focus of this work is *perfectly-secure* MPC. It is well known that perfectly-secure SMPC is possible if and only if adversary can corrupt up to $t_s < n/3$ parties [14]. On the other hand, perfectly-secure AMPC is possible if and only if adversary can corrupt up to $t_a < n/3$ parties [13].

Our Motivation and Our Results: As discussed above, known SMPC and AMPC protocols are designed under the *assumption* that the parties are *aware* of the exact network type. We envision a scenario where the parties *are not* aware of the *exact* network type and aim to design a single MPC protocol, which remains secure, *both* in a synchronous, *as well as* in an asynchronous network. We call such a protocol as a *best-of-both-worlds* protocol, since it offers the best security properties, both in the synchronous and the asynchronous communication model. While there exist best-of-both-worlds *conditionally-secure* MPC protocols [19, 30], to the best of our knowledge, *no* prior work has ever addressed the problem of getting a best-of-both-worlds *perfectly-secure* MPC protocol. Motivated by this, we ask the following question:

Is there a best-of-both-worlds perfectly-secure MPC protocol, that remains secure under t_s corruptions in a synchronous network, and under t_a corruptions in an asynchronous network, where $t_a < t_s$?

We show the existence of a perfectly-secure MPC protocol with the above guarantees, provided $3t_s + t_a < n$ holds.[1] Note that we are interested in the case where $t_a < t_s$, as otherwise the question is trivial to solve. More specifically, if $t_s = t_a$, then the necessary condition of AMPC implies that $t_s < n/4$ holds. Hence, one can use *any existing* perfectly-secure AMPC protocol, which will be secure under t_s corruptions *even* in a synchronous network. Moreover, by ensuring appropriate time-outs, it can be guaranteed that in the protocol, the inputs of *all* honest parties are considered for the computation, if the network is *synchronous*. Our goal is to achieve a resilience *strictly greater* than $n/4$ and *close* to $n/3$, if the underlying network is *synchronous*. For example, if $n = 8$, then *existing* perfectly-secure SMPC protocols can tolerate up to 2 corrupt parties, while *existing* perfectly-secure AMPC protocols can tolerate up to 1 fault. On the other hand, using our best-of-both-worlds protocol, one can tolerate up to 2 faults in a *synchronous* network and up to 1 fault in an *asynchronous* network, *even* if the parties are *not aware* of the exact network type.

1.1 Technical Overview

We assume that the function to be securely computed is represented by some arithmetic circuit cir over a finite field $\mathbb{F}$, consisting of linear and non-linear (multiplication) gates. Following [14], the goal is then to securely "evaluate" cir in a *secret-shared* fashion, such that all the values during the circuit-evaluation are t-shared, as per the Shamir's secret-sharing scheme [48], where t is the maximum number of corrupt parties.[2] Intuitively, this guarantees that an adversary controlling up to t parties *does not* learn any additional information during the circuit-evaluation, as the shares of the corrupt parties does not reveal anything additional about the actual shared values. The *degree-of-sharing* t is set to $t < n/3$ and $t < n/4$ in SMPC and AMPC protocols respectively.

[1] This automatically implies that $t_s < n/3$ and $t_a < n/4$ holds, which are necessary for designing perfectly-secure MPC protocol in a synchronous and an asynchronous network respectively.

[2] A value $s \in \mathbb{F}$ is said to be t-shared, if there is some t-degree polynomial $f_s(\cdot)$ over $\mathbb{F}$ with $f_s(0) = s$ and every (honest) P_i has a distinct point on $f_s(\cdot)$, which is called P_i's share of s.

Since, in our best-of-both-worlds protocol, the parties will *not* be aware of the *exact* network type, we need to ensure that *all* the values during circuit-evaluation are *always* secret-shared with the degree-of-sharing being $t = t_s$, *even* if the network is *asynchronous*.

For shared circuit-evaluation, we follow the Beaver's paradigm [8], where multiplication gates are evaluated using random t_s-shared *multiplication-triples* of the form (a, b, c), where $c = a \cdot b$ (due to the linearity of Shamir's secret-sharing, linear gates can be evaluated *non-interactively*). The shared multiplication-triples are generated in a *circuit-independent* preprocessing phase, using the framework of [26], which shows how to use any polynomial-based *verifiable secret-sharing* (VSS) [25] and a *Byzantine agreement* (BA) protocol [45] to generate shared random multiplication-triples. The framework works both in a synchronous as well as in an asynchronous network, where the parties are *aware* of the exact network type. However, there are several challenges to adapt the framework if the parties are *unaware* of the exact network type, which we discuss next.

First Challenge — A Best-of-Both-Worlds Byzantine Agreement (BA) Protocol: Informally, a BA protocol [45] allows the parties with private inputs to reach agreement on a common output (*consistency*), where the output is the input of the honest parties, if all honest parties participate in the protocol with the *same* input (*validity*). *Perfectly-secure* BA protocols can be designed tolerating $t < n/3$ corruptions, both in a *synchronous* network [45], as well as in an *asynchronous* network [22, 3, 7]. However, the *termination* (also called *liveness*) guarantees are *different* for *synchronous* BA (SBA) and *asynchronous* BA (ABA). (Deterministic) SBA protocols ensure that all honest parties obtain their output after some *fixed* time (*guaranteed liveness*). On the other hand, to circumvent the FLP impossibility result [33], ABA protocols are randomized and provide what is called as *almost-surely liveness* [3, 7]. Namely, the parties obtain an output, *asymptotically* with probability 1, if they continue running the protocol. SBA protocols become *insecure* when executed in an *asynchronous* network, if even a single expected message from an *honest* party is delayed. On the other hand, ABA protocols when executed in a *synchronous* network, can provide *only* almost-surely liveness, instead of guaranteed liveness.

The *first* challenge to adapt the framework of [26] in the best-of-both-worlds setting is to get a *perfectly-secure* BA protocol, which provides security *both* in a synchronous as well as in an asynchronous network. Namely, apart from providing the consistency and validity properties in both types of network, the protocol should provide guaranteed liveness in a *synchronous* network and almost-surely liveness in an *asynchronous* network. We are *not* aware of any BA protocol with the above properties. Hence, we present a perfectly-secure BA protocol tolerating $t < n/3$ corruptions, with the above properties. Since our BA protocol is slightly technical, we defer the details to Section 3.

Second Challenge — A best-of-both-worlds VSS Protocol: Informally, in a polynomial based VSS protocol, there exists a designated *dealer* D with a t-degree polynomial, where t is the maximum number of *corrupt* parties, possibly including D. The protocol allows D to distribute points on this polynomial to the parties in a "verifiable" fashion, such that the view of the adversary remains independent of D's polynomial for an *honest* D.[3] In a *synchronous* VSS (SVSS) protocol, every party has the correct point after some known time-out, say T (*correctness* property). The *verifiability* guarantees that even a *corrupt* D is bound to distribute points on some t-degree polynomial within time T (*strong-commitment* property). Perfectly-secure SVSS is possible if and only if $t < n/3$ [31]. For an *asynchronous* VSS (AVSS) protocol, the *correctness* property guarantees that

[3]Hence the protocol allows D to generate a t-sharing of the constant term of the polynomial, which is also called as D's *secret*.

for an *honest* D, the honest parties *eventually* receive points on D's polynomial. However, a *corrupt* D may not invoke the protocol in the first place and the parties *cannot* distinguish this scenario from the case when D's messages are arbitrarily delayed. This is unlike the *strong-commitment* of SVSS where, if the parties do not obtain an output within time T, then the parties *publicly* conclude that D is corrupt. Hence, the *strong-commitment* of AVSS guarantees that if D is *corrupt and* if some honest party obtains a point on D's polynomial, then all honest parties eventually obtain their respective points on this polynomial. Perfectly-secure AVSS is possible if and only if $t < n/4$ [13, 4].

Existing SVSS protocols [35, 34, 41, 23] become completely insecure in an *asynchronous* network, even if a single expected message from an *honest* party is delayed. On the other hand, existing AVSS protocols [13, 10, 44, 23] only work when D's polynomial has degree $t < n/4$ and become insecure if there are *more* than $n/4$ corruptions (which can happen in our context, if the network is *synchronous*).

The *second* challenge to adapt the framework of [26] in our setting is to get a perfectly-secure VSS protocol, which provides security against t_s and t_a corruptions in a synchronous and in an asynchronous network respectively, where D's polynomial is *always* a t_s-degree polynomial, *irrespective* of the network type. We are not aware of any VSS protocol with these guarantees. We present a best-of-both-worlds perfectly-secure VSS protocol satisfying the above properties, provided $3t_s + t_a < n$ holds. Our VSS protocol satisfies the *correctness* requirement of SVSS and AVSS in a *synchronous* and an *asynchronous* network respectively. However, it *only* satisfies the *strong-commitment* requirement of AVSS, *even* if the network is *synchronous*. This is because a potentially *corrupt* D *may not* invoke the protocol and the parties will *not* be aware of the exact network type. We stress that this does not hinder us from deploying our VSS protocol in the framework of [26]. Since our VSS protocol is slightly technical, we defer the details to Section 4.

1.2 Related Work

best-of-both-worlds protocols have been studied very recently. The work of [17] shows that the condition $2t_s + t_a < n$ is necessary and sufficient for best-of-both-worlds *conditionally-secure* BA, tolerating *computationally bounded* adversaries. Using the same condition, the works of [19, 30] present *conditionally-secure* MPC protocols. Moreover, the same condition has been used in [18] to design a best-of-both-worlds protocol for atomic broadcast (a.k.a. state machine replication). Furthermore, the same condition has been used recently in [37] to design a best-of-both-worlds approximate agreement protocol against computationally-bounded adversaries.

A common principle used in [17, 19, 30] to design best-of-both-worlds protocol for a specific task T, which could be either BA or MPC, is the following: the parties first run a *synchronous* protocol for task T with threshold t_s assuming a synchronous network, which *also* provides certain security guarantees in an asynchronous environment, tolerating t_a corruptions. After the known "time-out" of the synchronous protocol, the parties run an *asynchronous* protocol for T with threshold t_a, which also provides certain security guarantees in the presence of t_s corruptions. The input for the asynchronous protocol is decided based on the output the parties receive after the time-out of the synchronous protocol. The overall output is then decided based on the output parties receive from the asynchronous protocol. If the task T is MPC, then this means that the parties need to evaluate the circuit *twice*. We also follow a similar design principle as above, for our BA protocol. However, for MPC, we *do not* require the parties to run two protocols and evaluate the circuit twice. Rather the parties need to evaluate the circuit *only once*.

2 Preliminaries and Definitions

We follow the pairwise secure-channel model, where the parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$ are connected by pairwise private and authentic channels. The distrust in the system is modeled by a *computationally unbounded* Byzantine (malicious) adversary Adv, who can corrupt a subset of the parties and force them to behave in any *arbitrary* fashion, during the execution of a protocol. We assume a *static* adversary, who decides the set of corrupt parties at the beginning of the protocol execution. The underlying network can be synchronous or asynchronous, with parties being *unaware* about the exact type. In a *synchronous* network, every sent message is delivered in the same order, within some known fixed time Δ. The adversary Adv can control up to t_s parties in a synchronous network.

In an *asynchronous* network, messages are sent with an arbitrary, yet finite delay, and *need not* be delivered in the same order. The only guarantee is that every sent message is *eventually* delivered. The exact sequence of message delivery is decided by a *scheduler* and to model the worst case scenario, the scheduler is assumed to be under the control of Adv. The adversary can control up to t_a parties in an asynchronous network.

We assume that $t_a < t_s$ and $3t_s + t_a < n$ holds. This automatically implies that $t_s < n/3$ and $t_a < n/4$ holds, which are necessary for any SMPC and AMPC protocol respectively. All computations in our protocols are done over a finite field $\mathbb{F}$, where $|\mathbb{F}| > 2n$ and where $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$ are publicly-known, distinct, non-zero elements from $\mathbb{F}$. For simplicity and without loss of generality, we assume that each P_i has a private input $x^{(i)} \in \mathbb{F}$, and the parties want to securely compute a function $f : \mathbb{F}^n \rightarrow \mathbb{F}$. Without loss of generality, f is represented by an arithmetic circuit cir over $\mathbb{F}$, consisting of linear and non-linear (multiplication) gates [38], where cir has c_M number of multiplication gates and has a multiplicative depth of D_M.

Termination Guarantees of Our Sub-Protocols: For simplicity, we will *not* be specifying any termination criteria for our sub-protocols. And the parties will keep on participating in these sub-protocol instances, even after receiving their outputs. The termination criteria of our MPC protocol will ensure that once a party terminates the MPC protocol, it terminates all underlying sub-protocol instances. We will use existing *randomized* ABA protocols which ensure that the honest parties (eventually) obtain their respective output *almost-surely*. This means that the probability that an honest party obtains its output after participating for *infinitely* many rounds approaches 1 *asymptotically* [3, 43, 7]. That is:

$$\lim_{T \rightarrow \infty} \Pr[\text{An honest } P_i \text{ obtains its output by local time } T] = 1,$$

where the probability is over the random coins of the honest parties and the adversary in the protocol. The property of *almost-surely* obtaining the output carries over to the "higher" level protocols, where ABA is used as a building block. We will say that the "*honest parties obtain some output almost-surely from (an asynchronous) protocol* Π" to mean that every *honest* P_i *asymptotically* obtains its output in Π with probability 1, in above the sense.

We next discuss the properties of polynomials over $\mathbb{F}$, which are used in our protocols.

Polynomials Over a Field: A d-degree *univariate polynomial* over $\mathbb{F}$ is of the form

$$f(x) = a_0 + \ldots + a_d x^d,$$

where each $a_i \in \mathbb{F}$. An (ℓ, ℓ)-degree *symmetric bivariate polynomial* over $\mathbb{F}$ is of the form

$$F(x, y) = \sum_{i,j=0}^{i=\ell, j=\ell} r_{ij} x^i y^j,$$

where each $r_{ij} \in \mathbb{F}$ and where $r_{ij} = r_{ji}$ holds for all i, j. This automatically implies that $F(\alpha_j, \alpha_i) = F(\alpha_i, \alpha_j)$ holds, for all α_i, α_j. Moreover, $F(x, \alpha_i) = F(\alpha_i, y)$ also holds, for every α_i. Given an $i \in \{1, \ldots, n\}$ and an ℓ-degree polynomial $F_i(x)$, we say that $F_i(x)$ *lies* on an (ℓ, ℓ)-degree symmetric bivariate polynomial $F(x, y)$, if $F(x, \alpha_i) = F_i(x)$ holds.

We now state some standard known results, related to polynomials over $\mathbb{F}$. It is a well known fact that there always exists a unique d-degree univariate polynomial, passing through $d + 1$ distinct points. A generalization of this result for bivariate polynomials is that if there are "sufficiently many" univariate polynomials which are "pair-wise consistent", then together they lie on a unique bivariate polynomial. Formally:

Lemma 2.1 (**[27, 6]**). *Let $f_{i_1}(x), \ldots, f_{i_q}(x)$ be ℓ-degree univariate polynomials over $\mathbb{F}$, where $q \geq \ell + 1$ and $i_1, \ldots, i_q \in \{1, \ldots, n\}$, such that $f_i(\alpha_j) = f_j(\alpha_i)$ holds for all $i, j \in \{i_1, \ldots, i_q\}$. Then $f_{i_1}(x), \ldots, f_{i_q}(x)$ lie on a unique (ℓ, ℓ)-degree symmetric bivariate polynomial, say $F^\star(x, y)$.*

In existing (as well as our) VSS protocol, D on having a t-degree polynomial $q(\cdot)$ as input, embeds $q(\cdot)$ into a random (t, t)-degree symmetric bivariate polynomial $F(x, y)$ at $x = 0$. And each party P_i then receives the t-degree univariate polynomial $f_i(x) = F(x, \alpha_i)$. Here t is the maximum number of parties which can be under the control of Adv. This ensures that Adv by learning at most t polynomials lying on $F(x, y)$, does not learn anything about $F(0, 0)$. Intuitively, this is because Adv will fall short of at least one point on $F(x, y)$ to uniquely interpolate it. In fact, it can be shown that for every pair of t-degree polynomials $q_1(\cdot), q_2(\cdot)$ such that $q_1(\alpha_i) = q_2(\alpha_i) = f_i(0)$ holds for every $P_i \in \mathcal{C}$ (where $\mathcal{C}$ is the set of parties under Adv), the distribution of the polynomials $\{f_i(x)\}_{P_i \in \mathcal{C}}$ when $F(x, y)$ is chosen based on $q_1(\cdot)$, is identical to the distribution when $F(x, y)$ is chosen based on $q_2(\cdot)$. Formally:

Lemma 2.2 (**[27, 6]**). *Let $\mathcal{C} \subset \mathcal{P}$ and $q_1(\cdot) \neq q_2(\cdot)$ be d-degree polynomials where $d \geq |\mathcal{C}|$ such that $q_1(\alpha_i) = q_2(\alpha_i)$ for all $P_i \in \mathcal{C}$. Then the probability distributions $\left\{\{F(x, \alpha_i)\}_{P_i \in \mathcal{C}}\right\}$ and $\left\{\{F'(x, \alpha_i)\}_{P_i \in \mathcal{C}}\right\}$ are identical, where $F(x, y)$ and $F'(x, y)$ are random (d, d)-degree symmetric bivariate polynomials, such that $F(0, y) = q_1(\cdot)$ and $F'(0, y) = q_2(\cdot)$ holds.*

We next give the definition of d-sharing, which is central to our protocols.

Definition 2.3 (d**-sharing).** A value $s \in \mathbb{F}$ is said to be d-shared, if there exists a d-degree *sharing-polynomial*, say $f_s(\cdot)$, with $f_s(0) = s$, such that every (honest) P_i has the *share* $s_i = f_s(\alpha_i)$. The vector of shares of s corresponding to the (honest) parties P_i is called a d-sharing of s, denoted by $[s]_d$. We will omit the degree d from the notation $[\cdot]_d$ if it is clear from the context.

It is easy to see that d-sharing satisfies the *linearity* property; i.e. given $[a]_d$ and $[b]_d$, then $[c_1 \cdot a + c_2 \cdot b]_d = c_1 \cdot [a]_d + c_2 \cdot [b]_d$ holds, where $c_1, c_2 \in \mathbb{F}$ are *publicly-known*. In general, consider any arbitrary linear function $g : \mathbb{F}^\ell \to \mathbb{F}^m$ and let $u^{(1)}, \ldots, u^{(\ell)}$ be d-shared. When we say that *parties locally compute* $([v^{(1)}]_d, \ldots, [v^{(m)}]_d) = g([u^{(1)}]_d, \ldots, [u^{(\ell)}]_d)$, we mean that the parties *locally* apply the function g on their respective shares of $u^{(1)}, \ldots, u^{(\ell)}$, to get their respective shares of $v^{(1)}, \ldots, v^{(m)}$.

2.1 Existing Primitives

We next discuss the existing primitives used in our protocols.

Online Error-Correction (OEC) [13]: Let $\mathcal{P}'$ be a subset of parties, containing at most t corrupt parties. And let there exist some d-degree polynomial $q(\cdot)$ with every (honest) $P_i \in \mathcal{P}'$ having a point $q(\alpha_i)$. The goal is to make some *designated* party, say P_R, reconstruct $q(\cdot)$. For this, each $P_i \in \mathcal{P}'$ sends $q(\alpha_i)$ to P_R, who keeps waiting till it receives $d+t+1$ points, all of which lie on a *unique* d-degree polynomial. This step requires P_R to repeatedly apply the Reed-Solomon (RS) error-correction procedure [42] and try to recover $q(\cdot)$, upon receiving a new point from the parties in $\mathcal{P}'$. Once P_R receives $d+1+1$ points lying on a d-degree polynomial, say $q'(\cdot)$, then $q'(\cdot) = q(\cdot)$. This is because among these $d+t+1$ points, at least $d+1$ are from *honest* parties in $\mathcal{P}'$, which uniquely determine $q(\cdot)$. If $d < (|\mathcal{P}'| - 2t)$, then in an *asynchronous* network, P_R *eventually* receives $d+t+1$ points (from the *honest* parties in $\mathcal{P}'$) lying on $q(\cdot)$ and recovers $q(\cdot)$. Moreover, in a *synchronous* network, it will take at most Δ time for P_R to recover $q(\cdot)$, since the points of the honest parties will be delivered within Δ time. We denote the above procedure by $\mathsf{OEC}(d, t, \mathcal{P}')$, which is presented in Appendix A, along with its properties.

Finding $(n,t)-$star [13]: Let G be an undirected graph over $\mathcal{P}$. Then a pair $(\mathcal{E}, \mathcal{F})$ where $\mathcal{E} \subseteq \mathcal{F} \subseteq \mathcal{P}$ is called an $(n,t)-$star, if all the following hold.

- $|\mathcal{E}| \geq n - 2t$;
- $|\mathcal{F}| \geq n - t$;
- There exists an edge between every $P_i \in \mathcal{E}$ and every $P_j \in \mathcal{F}$.

The work of [13] presents an efficient algorithm (whose running time is polynomial in n), which we denote as $\mathsf{AlgStar}$. The algorithm always outputs an $(n,t)-$star $(\mathcal{E}, \mathcal{F})$, provided G contains a clique of size at least $n - t$.

Asynchronous Reliable Broadcast (Acast): We use the Bracha's Acast protocol [20], where there exists a designated *sender* $\mathsf{S} \in \mathcal{P}$ with input $m \in \{0,1\}^\ell$. The protocol allows S to send m *identically* to all the parties, in the presence of any $t < n/3$ corruptions, possibly including S. While the protocol has been primarily designed for an *asynchronous* network, it also provides certain guarantees in a *synchronous* network, as stated in Lemma 2.4. Notice that the protocol *does not* provide any liveness if S is *corrupt*, irrespective of the network type. This is because a *corrupt* S may not invoke the protocol in the first place. Moreover in a *synchronous* network, if S is *corrupt* and if the honest parties compute an output, then they *may not* get the output at the same time. And there may be a difference of at most 2Δ time within which the honest parties compute their output. The Acast protocol and proof of Lemma 2.4 are available in Appendix A.

Lemma 2.4. *Bracha's Acast protocol Π_{ACast} achieves the following in the presence of up to $t < n/3$ corruptions, where S has an input $m \in \{0,1\}^\ell$ for the protocol.*

- *Asynchronous Network:*
  - *(a) t-Liveness: If S is honest, then all honest parties eventually obtain an output.*
  - *(b) t-Validity: If S is honest, then every honest party with an output, outputs m.*
  - *(c) t-Consistency: If S is corrupt and some honest party outputs $m^\star$, then every honest party eventually outputs $m^\star$.*
- *Synchronous Network:*
  - *(a) t-Liveness: If S is honest, then all honest parties obtain an output within time 3Δ.*
  - *(b) t-Validity: If S is honest, then every honest party with an output, outputs m.*
  - *(c) t-Consistency: If S is corrupt and some honest party outputs $m^\star$ at time T, then every honest P_i outputs $m^\star$ by the end of time $T + 2\Delta$.*
- *Irrespective of the network type, $\mathcal{O}(n^2\ell)$ bits are communicated by the honest parties.*

We next discuss few terminologies with respect to Π_{ACast}, which we use throughout the paper.

Terminologies for Using Π_{ACast}: We will say that "P_i *Acasts* m" to mean that P_i acts as a sender S and invokes an instance of Π_{ACast} with input m and the parties participate in this instance. Similarly, we will say that "P_j *receives* m *from the Acast of* P_i" to mean that P_j outputs m in the corresponding instance of Π_{ACast}.

3 best-of-both-worlds Perfectly-Secure Byzantine Agreement

In this section, we present our best-of-both-worlds perfectly-secure Byzantine agreement (BA) protocol. We begin with the definition of BA, which is a modified version of [17], as we *do not* require any *termination* guarantees. In the definition, we consider the case where the inputs of the parties is a single bit. However, the definition can be easily extended for the case when the inputs are bit-strings.

Definition 3.1 (**Byzantine Agreement (BA)** [17])**.** Let Π be a protocol for the parties in $\mathcal{P}$ with up to t corrupt parties, where every P_i has an input $b_i \in \{0,1\}$ and a possible output from $\{0,1,\perp\}$.

- *t-Guaranteed Liveness*: Π has guaranteed liveness, if all honest parties obtain an output.
- *t-Almost-Surely Liveness*: Π has almost-surely liveness, if almost-surely, all honest parties obtain some output.
- *t-Validity*: Π has t-validity, if the following hold: if all honest parties have input b, then every honest party with an output, outputs b.
- *t-Weak Validity*: Π has t-weak validity, if the following hold: if all honest parties have input b, then every honest party with an output, outputs b or $\perp$.
- *t-Consistency*: Π has t-consistency, if all honest parties with an output, output the same value.
- *t-Weak Consistency*: Π has t-Weak Consistency, if all honest parties with an output, output either a common $v \in \{0,1\}$ or $\perp$.

Protocol Π is called a *t-perfectly-secure synchronous-BA* (SBA) protocol, if in a *synchronous* network, it achieves all the following:

- *t-guaranteed liveness*;
- *t-Validity*;
- *t-Consistency*.

Protocol Π is called a *t-perfectly-secure asynchronous-BA* (ABA) protocol, if in an *asynchronous network*, it achieves the following:

- *t-almost-surely liveness*;
- *t-Validity*;
- *t-Consistency*.

To design our best-of-both-worlds BA protocol, we will be using an *existing* perfectly-secure SBA and a perfectly-secure ABA protocol, whose properties we review next.

Existing t-Perfectly-Secure SBA: We assume the existence of a t-perfectly-secure SBA protocol tolerating $t < n/3$ corruptions, which *also* provides *t-guaranteed liveness* in an *asynchronous* network.[4]. For the sake of communication efficiency, we choose the recursive phase-king based *t-perfectly-secure* SBA protocol Π_{BGP} of [16]. The protocol incurs a communication of $\mathcal{O}(n^2\ell)$ bits, if the inputs of the parties are of size ℓ bits. If the network is *synchronous*, then in protocol Π_{BGP}, at time $T_{\mathsf{BGP}} = (12n-6)\cdot\Delta$, all honest parties have an output (see Lemma 10.7 of [1]). To ensure

[4]We stress that we *do not* require any other property from the SBA protocol in an *asynchronous* network.

guaranteed liveness in an *asynchronous* network, the parties can simply run the protocol and then check if any output is obtained at local time $(12n-6)\cdot\Delta$. In case no output is obtained, then $\perp$ is taken as the output. The properties of Π_{BGP} are summarized in Lemma 3.2.

Lemma 3.2 ([16, 1]). *Let $t < n/3$. Then there exists a protocol Π_{BGP} with the following properties, where all parties participate with an input of size ℓ bits.*

- *The protocol incurs a communication of $\mathcal{O}(n^2\ell)$ bits from the honest parties.*
- *The protocol is a t-perfectly-secure SBA protocol, where all honest parties have an output within time $T_{\mathsf{BGP}} = (12n-6)\cdot\Delta$.*
- *In an asynchronous network, all honest parties have an output from $\{0,1\}^\ell \cup \{\perp\}$, within local time $(12n-6)\cdot\Delta$.*

Existing t-Perfectly-Secure ABA: Existing perfectly-secure ABA protocols achieve the following properties.

Lemma 3.3 ([3, 7]). *Let $t < n/3$. Then there exists a randomized protocol Π_{ABA}, achieving the following properties, where the inputs of each party is a bit.*

- *Asynchronous Network: The protocol is a t-perfectly-secure ABA protocol and provides the following liveness guarantees.*
  - *If the inputs of all honest parties are same, then Π_{ABA} achieves t-guaranteed liveness;*
  - *Else Π_{ABA} achieves t-almost-surely liveness.*
- *Synchronous Network: The protocol achieves t-validity, t-consistency and the following liveness guarantees.*
  - *If the inputs of all honest parties are same, then Π_{ABA} achieves t-guaranteed liveness and all honest parties obtain their output within time $T_{\mathsf{ABA}} = k\cdot\Delta$ for some constant k.*
  - *Else Π_{ABA} achieves t-almost-surely liveness and requires $\mathcal{O}(poly(n)\cdot\Delta)$ expected time to generate the output.*
- *Irrespective of the network type, the protocol incurs the following amount of communication from the honest parties.*
  - *If the inputs of all honest parties are the same, then the protocol incurs a communication of $\mathcal{O}(poly(n)\log|\mathbb{F}|)$ bits;*
  - *Else, it incurs an expected communication of $\mathcal{O}(poly(n)\log|\mathbb{F}|)$ bits.*[5]

Protocol Π_{ABA} is designed using a *weaker* "variant" of AVSS called *shunning* AVSS (SAVSS) [3, 7], which *cannot* be used for circuit-evaluation. We provide a brief overview of the ABA protocols of [3, 7] and a brief outline of the proof of Lemma 3.3 in Appendix B.

From the above discussion, we note that protocol Π_{ABA} *cannot* be considered as a *best-of-both-worlds* BA protocol. This is because the protocol achieves *t-guaranteed liveness* in a *synchronous* network, *only* when *all* honest parties have the same input. In case, the parties have a mixed bag of inputs, then the parties may end up running the protocol forever, without having an output, even if the network is *synchronous*, though the probability of this event is asymptotically 0. We design a *perfectly-secure* BA protocol, which solves this problem and which is secure in *any* network. To design the protocol, we need a special type of broadcast protocol, which we design first.

3.1 Synchronous Broadcast with Asynchronous Guarantees

We begin with the definition of broadcast, adapted from [17], where we *do not* put any *termination* requirement.

[5] Looking ahead, the number of invocations of Π_{ABA} in our protocol will be a *constant* and *independent* of the size of the circuit cir. Hence, we do not focus on the "exact" communication complexity of Π_{ABA}.

Definition 3.4 (**Broadcast** [17])**.** Let Π be a protocol for the parties in $\mathcal{P}$ consisting of up to t corrupt parties, where a sender $\mathsf{S} \in \mathcal{P}$ has input $m \in \{0,1\}^\ell$, and parties obtain a possible output from $\{0,1\}^\ell \cup \{\perp\}$.

- t-*Liveness*: Π has t-liveness, if all honest parties obtain some output.
- t-*Validity*: Π has t-validity, if the following holds: if S is *honest*, then every honest party with an output, outputs m.
- t-*Weak Validity*: Π has t-validity, if the following holds: if S is *honest*, then every honest party outputs either m or $\perp$.
- t-*Consistency*: Π has t-consistency, if the following holds: if S is *corrupt*, then every honest party with an output, has a common output.
- t-*Weak Consistency*: Π has t-weak consistency, if the following holds: if S is *corrupt*, then every honest party with an output, outputs a common $m^\star \in \{0,1\}^\ell$ or $\perp$.

Protocol Π is called a t-*perfectly-secure broadcast* protocol, if it has the following properties:

- t-*Liveness*;
- t-*Validity*;
- t-*Consistency*.

We next design a *special* broadcast protocol Π_{BC}, which is a t-perfectly-secure broadcast protocol in a *synchronous* network. Additionally, in an *asynchronous* network, the protocol achieves t-*liveness*, t-*weak validity* and t-*weak consistency*. Looking ahead, we will combine the protocols $\Pi_{\mathsf{BC}}, \Pi_{\mathsf{BGP}}$ and Π_{ABA} to get our best-of-both-worlds BA protocol.

Before proceeding to design Π_{BC}, we note that the existing Bracha's Acast protocol Π_{ACast} *does not* guarantee the same properties as Π_{BC}. Specifically, for a *corrupt* S, there is *no* liveness guarantee (irrespective of the network type). Moreover, in a *synchronous* network, if S is *corrupt* and honest parties obtain an output, then they *may not* obtain an output within the same time (see Lemma 2.4).[6] Interestingly, our instantiation of Π_{BC} is based on Π_{ACast}, by carefully "stitching" it with the protocol Π_{BGP}.

The idea behind Π_{BC} is the following: sender S first Acasts its message. If the network is *synchronous* and S is *honest*, then within time 3Δ, every honest party should have received S's message. To verify this, the parties start participating in an instance of Π_{BGP} at (local) time 3Δ, with their respective inputs being the output obtained from S's Acast at time 3Δ. If there is no output at time 3Δ from S's Acast, then the input for Π_{BGP} is $\perp$. Finally, at time $3\Delta + T_{\mathsf{BGP}}$, parties output $m^\star$, if it has been received from the Acast of S *and* if it is the output of Π_{BGP} as well; otherwise the parties output $\perp$.

It is easy to see that the protocol has now *guarantees* liveness in *any* network (irrespective of S), since all parties will have some output at (local) time $3\Delta + T_{\mathsf{BGP}}$. Moreover, *consistency* is achieved for a *corrupt* S in a *synchronous* network, with *all* honest parties obtaining a common output at the *same* time. This is because if any *honest* party obtains an output $m^\star \neq \perp$ at time $3\Delta + T_{\mathsf{BGP}}$, then *at least* one *honest* party must have received $m^\star$ from S's Acast by time 3Δ. And so by time $3\Delta + T_{\mathsf{BGP}}$, *all* honest parties will receive $m^\star$ from S's Acast.

Eventual Consistency and Validity in Asynchronous Network: In Π_{BC}, the parties set a "time-out" of $3\Delta + T_{\mathsf{BGP}}$, due to which it provides *weak validity* and *weak consistency* in an *asynchronous* network. This is because some *honest* parties may receive S's message from the Acast of S within the timeout, while others may fail to do so. The time-out is essential, as we need *liveness* from Π_{BC} in *both* synchronous and asynchronous network, when Π_{BC} is used later in our best-of-both-worlds BA protocol.

[6] Looking ahead, this property from Π_{BC} will be crucial when we use it in our best-of-both-worlds BA protocol.

Looking ahead, we will use Π_{BC} in our VSS protocol for broadcasting protocol. Due to the *weak validity* and *weak consistency* properties, we may end up in a scenario where one subset of *honest* parties may output a common value different from $\perp$ at the end of the time-out, while others may output $\perp$. For the security of the VSS protocol, we would require even the latter subset of (honest) parties to *eventually* output the common non-$\perp$ value, if the parties *continue* participating in Π_{BC}. To achieve this goal, every party who outputs $\perp$ at time $3\Delta + T_{\mathsf{BGP}}$, "switches" its output to $m^\star$, if it *eventually* receives $m^\star$ from S's Acast. We stress that this switching is *only* for the parties who obtained $\perp$ at time $3\Delta + T_{\mathsf{BGP}}$. To differentiate between the two ways of obtaining output, we use the terms *regular-mode* and *fallback-mode*. The regular-mode refers to the process of deciding the output at time $3\Delta + T_{\mathsf{BGP}}$, while the fallback-mode refers to the process of deciding the output beyond time $3\Delta + T_{\mathsf{BGP}}$.[7]

If the network is *asynchronous* and S is *honest*, then from the *liveness* and *validity* of Π_{ACast}, every honest party *eventually* obtains m from S's Acast. Hence, through the fallback-mode, every honest party who outputs $\perp$ at the time-out of $3\Delta + T_{\mathsf{BGP}}$, eventually outputs m. Moreover, even if S is *corrupt*, the fallback-mode *will not* lead to different honest parties obtaining different non-$\perp$ outputs due to the *consistency* property of Π_{ACast}.

Protocol Π_{BC}

(Regular Mode)

- On having the input $m \in \{0,1\}^\ell$, sender S Acasts m.
- At time 3Δ, each $P_i \in \mathcal{P}$ participates in an instance of Π_{BGP}, where the input for Π_{BGP} is set as follows:
  - P_i sets $m^\star$ as the input, if $m^\star \in \{0,1\}^\ell$ is received from the Acast of S;
  - Else P_i sets $\perp$ as the input (encoded as a default ℓ-bit string).
- **(Local Computation)**: At time $3\Delta + T_{\mathsf{BGP}}$, each $P_i \in \mathcal{P}$ computes its output through *regular-mode* as follows:
  - P_i outputs $m^\star \neq \perp$, if $m^\star$ is received from the Acast of S *and* $m^\star$ is computed as the output during the instance of Π_{BGP};
  - Else, P_i outputs $\perp$.

  Each $P_i \in \mathcal{P}$ keeps participating in the protocol, even after computing the output.

(Fallback Mode)

- Every $P_i \in \mathcal{P}$ who has computed the output $\perp$ at time $3\Delta + T_{\mathsf{BGP}}$, changes it to $m^\star$, if $m^\star$ is received by P_i from the Acast of S.

Figure 1: Synchronous broadcast with asynchronous guarantees.

We next prove the properties of the protocol Π_{BC}.

Theorem 3.5. *Protocol Π_{BC} achieves the following properties in the presence of any $t < n/3$ corruptions, where S has an input $m \in \{0,1\}^\ell$ and where $T_{\mathsf{BC}} = (12n-3)\cdot\Delta$.*

- *Synchronous network:*
  - *(a) t-Liveness: At time T_{BC}, every honest party has an output, through regular-mode.*
  - *(b) t-Validity: If S is honest, then at time T_{BC}, each honest party outputs m through regular-mode.*
  - *(c) t-Consistency: If S is corrupt, then the output of every honest party is the same at time T_{BC} through regular-mode.*
  - *(d) t-Fallback Consistency: If S is corrupt and some honest party outputs $m^\star \neq \perp$ at time*

[7] The fallback-mode is never triggered when Π_{BC} is used in our best-of-both-worlds BA protocol. It will be triggered (along with the regular-mode) in our VSS protocol.

T through fallback-mode, then every honest party outputs $m^\star$ by time $T + 2\Delta$ through fallback-mode.

- *Asynchronous Network:*
  - *(a) t-Liveness: At local time T_{BC}, every honest party has an output, through regular-mode.*
  - *(b) t-Weak Validity: If S is honest, then at local time T_{BC}, each honest party outputs m or $\perp$ through regular-mode.*
  - *(c) t-Fallback Validity: If S is honest, then each honest party who outputs $\perp$ at local time T_{BC} through regular-mode, eventually outputs m through fallback-mode.*
  - *(d) t-Weak Consistency: If S is corrupt, then at local time T_{BC}, each honest party outputs either a common $m^\star \neq \perp$ or $\perp$, through regular-mode.*
  - *(e) t-Fallback Consistency: If S is corrupt and some honest party outputs $m^\star \neq \perp$ at local time T, either through regular or fallback-mode, then every honest party eventually outputs $m^\star$, either through regular or fallback-mode.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^2\ell)$ bits from the honest parties.*

Proof. The *liveness* (both for the *synchronous* as well *asynchronous* network) simply follows from the fact that every honest party outputs something (including $\perp$) at (local) time $T_{\mathsf{BC}} = 3\Delta + T_{\mathsf{BGP}}$, where $T_{\mathsf{BGP}} = (12n - 6) \cdot \Delta$. We next prove the rest of the properties of the protocol in the *synchronous* network, for which we rely on the properties of Acast and Π_{BGP} in the *synchronous* network.

If S is *honest*, then due to the *liveness* and *validity* properties of Π_{ACast} in the *synchronous* network, at time 3Δ, every *honest* party P_i receives m from the Acast of S. Hence, every honest party participates with input m in the instance of Π_{BGP}. From the *guaranteed liveness* and *validity* properties of Π_{BGP} in *synchronous* network, at time $3\Delta + T_{\mathsf{BGP}}$, every honest party will have m as the output from Π_{BGP}. Hence, each honest party has the output m at time T_{BC}, thus proving that *validity* is achieved.

For *consistency*, we consider a *corrupt* S. We first note that each honest party will have the *same* output from the instance of Π_{BGP} at time T_{BC}, which follows from the *consistency* property of Π_{BGP} in *synchronous* network. If all honest honest parties have the output $\perp$ for Π_{BC} at time T_{BC}, then consistency holds trivially. So consider the case when some *honest* party, say P_i, has the output $m^\star \neq \perp$ for Π_{BC} at time T_{BC}. This implies that the output of Π_{BGP} is $m^\star$ for every honest party. Moreover, it also implies that at time 3Δ, at least one *honest* party, say P_h, has received $m^\star$ from the Acast of S. Otherwise, all honest parties would participate with input $\perp$ in the instance of Π_{BGP} and from the *validity* of Π_{BGP} in the *synchronous* network, every honest party would compute $\perp$ as the output during Π_{BGP}, which is a contradiction. Since P_h has received $m^\star$ from S's Acast at time 3Δ, it follows from the *consistency* property of Π_{ACast} in the *synchronous* network that *all* honest parties will receive $m^\star$ from S's Acast by time 5Δ. Moreover, $5\Delta < (12n - 3) \cdot \Delta$ holds. Consequently, by time $(12n-3) \cdot \Delta$, *all* honest parties will receive $m^\star$ from S's Acast and will have $m^\star$ as the output of Π_{BGP} and hence, output $m^\star$ for Π_{BC}.

For *fallback consistency*, we have to consider a *corrupt* S. Let P_h be an *honest* party who outputs $m^\star \neq \perp$ at time T through fallback-mode. Since the steps of fallback-mode are executed after time T_{BC}, it follows that $T > T_{\mathsf{BC}}$. We first note that this implies that *every* honest party has output $\perp$ at time T_{BC}, through regular-mode. This is because, from the proof of the *consistency* property of Π_{BC}, if any *honest* party has an output $m' \neq \perp$ at time T_{BC}, then *all* honest parties (including P_h) also must have computed the output m' at time T_{BC}, through regular-mode. And hence, P_h will never change its output to $m^\star$.[8] Since P_h has computed the output $m^\star$, it means that at time

[8]Recall that in the protocol the parties who obtain an output different from $\perp$ at time T_{BC}, never change their

T, it has received $m^\star$ from the Acast of S. It then follows from the *consistency* of Π_{ACast} in the *synchronous* network that every honest party will also receive $m^\star$ from the Acast of S, latest by time $T + 2\Delta$ and output $m^\star$ through fallback-mode.

We next prove the properties of the protocol Π_{BC} in an *asynchronous* network, for which we depend upon the properties of Π_{ACast} in the *asynchronous* network. The *weak-validity* property follows from the *validity* property of Π_{ACast} in the *asynchronous* network, which ensures that no honest party P_i ever receives an m' from the Acast of S where $m' \neq m$. So if at all P_i outputs a value different from $\perp$ at time T_{BC}, it has to be m. The *weak-consistency* property follows using similar arguments as used to prove *consistency* in the *synchronous* network, but relying instead on the *validity* and *consistency* properties of Π_{ACast} in the asynchronous network. The latter property ensures that even if the adversary has full control over message scheduling in the *asynchronous* network, it cannot ensure that for a *corrupt* S, two different honest parties end up receiving m_1 and m_2 from the Acast of S, where $m_1 \neq m_2$.

For *fallback validity*, consider an *honest* S and let P_i be an *honest* party, who outputs $\perp$ at (local) time T_{BC} through regular-mode. Since the parties keep on participating in the protocol beyond time T_{BC}, it follows from the *liveness* and *validity* properties of Π_{ACast} in the *asynchronous* network that party P_i will *eventually* receive m from the Acast of S through the fallback-mode of Π_{BC}. Consequently, party P_i eventually changes its output from $\perp$ to m.

For *fallback consistency*, we consider a *corrupt* S and let P_j be an *honest* party, who outputs some $m^\star$ at time T where $T \geq T_{\mathsf{BC}}$. This implies that P_j has obtained $m^\star$ from the Acast of S. Now, consider an arbitrary *honest* P_i. From the *liveness* and *weak consistency* properties of Π_{BC} in *asynchronous* network, it follows that P_i outputs either $m^\star$ or $\perp$ at local time T_{BC}, through the regular-mode. If P_i has output $\perp$, then from the *consistency* property of Π_{ACast} in the *asynchronous* network, it follows that P_i will also eventually obtain $m^\star$ from the Acast of S through the fallback-mode of Π_{BC}. Consequently, party P_i eventually changes its output from $\perp$ to $m^\star$.

The *communication complexity* follows from the communication complexity of Π_{BGP} and Π_{ACast}. □

We next discuss few terminologies for Π_{BC}, which we use in the rest of the paper.

Terminologies for Π_{BC}: When we say that "*P_i broadcasts m*", we mean that P_i invokes Π_{BC} as S with input m and the parties participate in this instance. Similarly, when we say that "*P_j receives m from the broadcast of P_i through regular-mode*", we mean that P_j has the output m at time T_{BC}, during the instance of Π_{BC}. Finally, when we say that "*P_j receives m from the broadcast of P_i through fallback-mode*", we mean that P_j has the output m after time T_{BC} during the instance of Π_{BC}.

3.2 $\Pi_{\mathsf{BC}} + \Pi_{\mathsf{ABA}} \Rightarrow$ best-of-both-worlds BA

We now show how to combine the protocols Π_{BC} and Π_{ABA} to get our best-of-both-worlds BA protocol Π_{BA}. For this, we use an idea used in [17], to get a best-of-both-worlds BA protocol with *conditional* security. In the protocol, every party first broadcasts its input bit through an instance of Π_{BC}. If the network is *synchronous*, then all honest parties should have received the inputs of all the (honest) sender parties from their broadcasts through regular-mode, within time T_{BC}. Consequently, at time T_{BC}, the parties decide an output for all the n instances of Π_{BC}. Based on these outputs, the parties decide their respective inputs for an instance of the Π_{ABA} protocol. Specifically, if "sufficiently many" outputs from the Π_{BC} instances are found to be the same, then the

output.

parties consider it as their input for the Π_{ABA} instance. Else, they stick to their original inputs. The overall output of the protocol is then set to be the output from Π_{ABA}.

Protocol Π_{BA}

- On having input $b_i \in \{0,1\}$, broadcast b_i.
- For $j = 1, \ldots, n$, let $b_i^{(j)} \in \{0,1,\perp\}$ be received from the broadcast of P_j through regular-mode. Include P_j to a set $\mathcal{R}$, if $b_i^{(j)} \neq \perp$. Compute the input $v_i^\star$ for an instance of Π_{ABA} as follows.
  - If $|\mathcal{R}| \geq n - t$, then set $v_i^\star$ to the majority bit among the $b_i^{(j)}$ values of the parties in $\mathcal{R}$.[a]
  - Else set $v_i^\star = b_i$.
- At time T_{BC}, participate in an instance of Π_{ABA} with input $v_i^\star$. Output the result of Π_{ABA}.

[a] If there is no majority, then set $v_i^\star = 1$.

Figure 2: The best-of-both-worlds BA protocol. The above code is executed by every $P_i \in \mathcal{P}$.

We next prove the properties of the protocol Π_{BA}. We note that protocol Π_{BA} is invoked only $\mathcal{O}(n^3)$ times in our MPC protocol, which is *independent* of cir. Consequently, we do not focus on the *exact* communication complexity of Π_{BA}.

Theorem 3.6. *Let $t < n/3$ and let Π_{ABA} be a randomized protocol, satisfying the conditions as per Lemma 3.3. Then Π_{BA} achieves the following, where every party participates with an input bit.*

- **Synchronous Network**: *The protocol is a t-perfectly-secure SBA protocol, where all honest parties obtain an output within time $T_{\mathsf{BA}} = T_{\mathsf{BC}} + T_{\mathsf{ABA}}$. The protocol incurs a communication of $\mathcal{O}(poly(n) \log |\mathbb{F}|)$ bits from the honest parties.*
- **Asynchronous Network**: *The protocol is a t-perfectly-secure ABA protocol with an expected communication of $\mathcal{O}(poly(n) \log |\mathbb{F}|)$ bits.*

Proof. We start with the properties in a *synchronous* network. The *t-liveness* property of Π_{BC} in the *synchronous* network guarantees that all honest parties will have some output, from each instance of Π_{BC} through regular-mode, at time T_{BC}. Moreover, the *t-validity* and *t-consistency* properties of Π_{BC} in the *synchronous* network guarantee that irrespective of the sender parties, *all* honest parties will have a common output from *each* individual instance of Π_{BC}, at time T_{BC}. Now since the parties decide their respective inputs for the instance of Π_{ABA} *deterministically* based on the individual outputs from the n instances of Π_{BC} at time T_{BC}, it follows that all honest parties participate with a *common* input in the protocol Π_{ABA}. Hence, all honest parties obtain an output by the end of time $T_{\mathsf{BC}} + T_{\mathsf{ABA}}$, thus ensuring *$t$-guaranteed liveness* of Π_{BA}. Moreover, the *t-consistency* property of Π_{ABA} in the *synchronous* network guarantees that all honest parties have a *common* output from the instance of Π_{ABA}, which is taken as the output of Π_{BA}, thus proving the *t-consistency* of Π_{BA}.

For proving the *validity* in the synchronous network, let all *honest* parties have the same input bit b. From the *t-consistency* of Π_{BC} in the *synchronous* network, all honest parties will receive b as the output at time T_{BC} in all the Π_{BC} instances, corresponding to the *honest* sender parties. Since there are at least $n - t$ honest parties, it follows that all honest parties will find a *common* subset $\mathcal{R}$ in the protocol, as the set of honest parties constitutes a candidate $\mathcal{R}$. Moreover, all honest parties will be present in $\mathcal{R}$, as $n - t > t$ holds. Since the set of honest parties constitute a majority in $\mathcal{R}$, it follows that all honest parties will participate with input b in the instance of Π_{ABA} and hence output b at the end of Π_{ABA}, which follows from the *t-validity* of Π_{ABA} in the *synchronous* network. This proves the *t-validity* of Π_{BA}.

We next prove the properties of Π_{BA} in an *asynchronous* network. The *t-consistency* of the protocol Π_{BA} follows from the *t-consistency* of the protocol Π_{ABA} in the *asynchronous* network,

since the overall output of the protocol Π_{BA} is same as the output of the protocol Π_{ABA}. The *t-liveness* of the protocol Π_{BC} in the *asynchronous* network guarantees that all honest parties will have some output from all the n instances of Π_{BC} at local time T_{BC} through regular-mode. Consequently, all honest parties will participate with some input in the instance of Π_{ABA}. The *t-almost-surely liveness* of Π_{ABA} in the *asynchronous* network then implies the *t-almost-surely liveness* of Π_{BA}.

For proving the validity in an *asynchronous* network, let all *honest* parties have the same input bit b. We claim that all *honest* parties participate with input b during the instance of Π_{ABA}. The *t-validity* of Π_{ABA} in the *asynchronous* network then automatically implies the *t-validity* of Π_{BA}.

To prove the above claim, consider an arbitrary *honest* party P_h. There are two possible cases. If P_h fails to find a subset $\mathcal{R}$ satisfying the protocol conditions, then the claim holds trivially, as P_h participates in the instance of Π_{ABA} with its input for Π_{BA}, which is the bit b. So consider the case when P_h finds a subset $\mathcal{R}$, such that $|\mathcal{R}| \geq n - t$ and where corresponding to each $P_j \in \mathcal{R}$, party P_h has computed an output $b_h^{(j)} \in \{0, 1\}$ at local time T_{BC} during the instance $\Pi_{\mathsf{BC}}^{(j)}$, through regular-mode. Now consider the subset of *honest* parties in the set $\mathcal{R}$. Since $t < n/3$, it follows that $n - 2t > t$ and hence the *majority* of the parties in $\mathcal{R}$ will be *honest*. Moreover, P_h will compute the output b at local time T_{BC} in the instance of Π_{BC}, corresponding to *every honest* P_j in $\mathcal{R}$, which follows from the *t-weak validity* of Π_{BC} in the *asynchronous* network. From these arguments, it follows that P_h will set b as its input for the instance of Π_{ABA}, thus proving the claim.

The communication complexity, both in a synchronous as well as in an asynchronous network, follows easily from the protocol steps and from the communication complexity of Π_{SBA} and Π_{ABA}. □

4 best-of-both-worlds Perfectly-Secure VSS

In this section, we present our best-of-both-worlds VSS protocol Π_{VSS}. In the protocol, there exists a designated *dealer* $\mathsf{D} \in \mathcal{P}$. The input for D consists of L number of t_s-degree polynomials $q^{(1)}(\cdot), \ldots, q^{(L)}$, where $L \geq 1$. And each (honest) P_i is supposed to "verifiably" receive the *shares* $\{q(\alpha_i)\}_{\ell=1,\ldots,L}$. Hence, the goal is to generate a t_s-sharing of $\{q(0)\}_{\ell=1,\ldots,L}$.[9] If D is *honest*, then in an *asynchronous* network, each (honest) P_i *eventually* gets its shares, while in a *synchronous* network, P_i gets its shares after some *fixed* time, such that the view of the adversary remains independent of D's polynomials. The *verifiability* here ensures that if D is *corrupt*, then either no honest party obtains any output (if D does not invoke the protocol), or there exist L number of t_s-degree polynomials, such that D is "committed" to these polynomials and each honest P_i gets its shares lying on these polynomials. Note that in the latter case, we *cannot* bound the time within which honest parties will have their shares, even if the network is *synchronous*. This is because a *corrupt* D may delay sending the messages arbitrarily and the parties *will not* know the exact network type. To design Π_{VSS}, we first design a "weaker" primitive called *weak polynomial-sharing* (WPS), whose security guarantees are *identical* to that of VSS for an *honest* D. However, for a *corrupt* D, the security guarantees are "weakened", as only a subset of the honest parties may get their shares of the committed polynomials.

4.1 The best-of-both-worlds Weak Polynomial-Sharing (WPS) Protocol

For simplicity, we explain our WPS protocol Π_{WPS}, assuming D has a *single* t_s-degree polynomial $q(\cdot)$ as input. Later we discuss the modifications needed to handle L polynomials efficiently. Protocol Π_{WPS} is obtained by carefully "stitching" a *synchronous* WPS protocol with an *asynchronous*

[9] Note that the degree of D's polynomials is *always* t_s, *irrespective* of the underlying network type.

WPS protocol. We first explain these two individual protocols, followed by the procedure to stitch them together, where the parties will *not be* knowing the exact network type.

WPS in an Asynchronous Network: In an *asynchronous* network, one can consider the following protocol Π_{AWPS}: D embeds $q(\cdot)$ in a random (t_s, t_s)-degree symmetric bivariate polynomial $Q(x, y)$ at $x = 0$ and distributes univariate polynomials lying on $Q(x, y)$ to respective parties. To verify whether D has distributed "consistent" polynomials, the parties check for the pair-wise consistency of their supposedly common points and make public the results through `OK` messages, if the tests are "positive". Based on the `OK` messages, the parties prepare a *consistency graph* and look for an $(n, t_a)-$star, say $(\mathcal{E}', \mathcal{F}')$. If D is *honest*, then $(\mathcal{E}', \mathcal{F}')$ will be obtained eventually, since the honest parties form a clique of size at least $n - t_a$. The existence of $(\mathcal{E}', \mathcal{F}')$ guarantees that the polynomials of the *honest* parties in $\mathcal{F}'$ lie on a single (t_s, t_s)-degree symmetric bivariate polynomial $Q^\star(x, y)$, where $Q^\star(x, y) = Q(x, y)$ for an *honest* D. This is because $\mathcal{E}'$ has at least $t_s + 1$ *honest* parties with pair-wise consistent polynomials, defining $Q^\star(x, y)$. And the polynomial of every *honest* party in $\mathcal{F}'$ is pair-wise consistent with the polynomials of every *honest* party in $\mathcal{E}'$. The parties *outside* $\mathcal{F}'$ obtain their polynomials lying on $Q^\star(x, y)$ by applying OEC on the common points on these polynomials received from the parties in $\mathcal{F}'$. Every P_i then outputs $Q^\star(0, \alpha_i)$ as its share, which is same as $q^\star(\alpha_i)$, where $q^\star(\cdot) = Q^\star(0, y)$. Note that Π_{AWPS} actually constitutes a VSS in the *asynchronous* network, as for an *honest* D every honest party eventually gets its share. On the other hand, for a *corrupt* D, every honest party eventually gets its share, if some honest party gets its share.

WPS for Synchronous Network: Π_{AWPS} fails in a *synchronous* network if there are t_s corruptions. This is because only $n - t_s$ honest parties are guaranteed and hence the parties may *fail* to find an $(n, t_a)-$star. The existence of an $(n, t_s)-$star, say $(\mathcal{E}, \mathcal{F})$, in the consistency graph is not "sufficient" to conclude that D has distributed consistent polynomials, lying on a (t_s, t_s)-degree symmetric bivariate polynomial. This is because if D is *corrupt*, then $\mathcal{E}$ is guaranteed to have *only* $n - 2t_s - t_s > t_a$ *honest* parties, with pair-wise consistent polynomials. Whereas to define a (t_s, t_s)-degree symmetric bivariate polynomial, we need more than t_s pair-wise consistent polynomials. On the other hand, if D is *corrupt*, then the honest parties in $\mathcal{F}$ *need not* constitute a clique. As a result, the polynomials of the honest parties in $\mathcal{F}$ need not be pair-wise consistent and hence need not lie on a (t_s, t_s)-degree symmetric bivariate polynomial.

To get rid of the above problem, the parties instead look for a "special" $(n, t_s)-$star $(\mathcal{E}, \mathcal{F})$, where the polynomials of all *honest* parties in $\mathcal{F}$ are guaranteed to lie on a single (t_s, t_s)-degree symmetric bivariate polynomial. Such a special $(\mathcal{E}, \mathcal{F})$ is bound to exist for an *honest* D. Based on the above idea, protocol Π_{SWPS} for a *synchronous* network proceeds as follows. For ease of understanding, we explain the protocol as a sequence of communication *phases*, with the parties being *synchronized* in each phase.

In the *first* phase, D distributes the univariate polynomials, during the *second* phase the parties perform pair-wise consistency tests and during the *third* phase the parties make public the results of *positive* tests. Additionally, the parties *also* make public the results of "negative" tests through `NOK` messages and their respective versions of the disputed points (the `NOK` messages *were not* required for Π_{AWPS}). The parties then construct the consistency graph. Next D *removes* all the parties from consideration in its consistency graph, who have made public "incorrect" `NOK` messages, whose version of the disputed points are *incorrect*. Among the *remaining* parties, D checks for the presence of a set of at least $n-t_s$ parties $\mathcal{W}$, such that the polynomial of *every* party in $\mathcal{W}$ is publicly confirmed to be pair-wise consistent with the polynomials of at least $n - t_s$ parties within $\mathcal{W}$. If a

$\mathcal{W}$ is found, then D checks for the presence of an $(n, t_s)-$star, say $(\mathcal{E}, \mathcal{F})$ among $\mathcal{W}$ and broadcasts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ during the *fourth* phase, if D finds $(\mathcal{E}, \mathcal{F})$. The parties upon receiving $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ verify if $\mathcal{W}$ is of size at least $n-t_s$ and every party in $\mathcal{W}$ has an edge with at least $n-t_s$ parties within $\mathcal{W}$ in their local copy of the consistency graph. The parties also check whether indeed $(\mathcal{E}, \mathcal{F})$ constitutes an $(n, t_s)-$star among the parties within $\mathcal{W}$. Furthermore, the parties now *additionally* verify whether any pair of parties P_j, P_k from $\mathcal{W}$ have made public "conflicting" `NOK` messages during the *third* phase. That is, if there exists any $P_j, P_k \in \mathcal{W}$ who made public `NOK` messages with q_{jk} and q_{kj} respectively during the *third* phase such that $q_{jk} \neq q_{kj}$, then $\mathcal{W}$ is *not* accepted. The idea here is that if D is *honest*, then at least one of P_j, P_k is bound to be *corrupt*, whose corresponding `NOK` message is incorrect. Since D also would have seen these public `NOK` messages during the third phase, it should have have discarded the corresponding corrupt party, before finding $\mathcal{W}$. Hence if a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted at the end of fourth phase, then the polynomials of all *honest* parties in $\mathcal{W}$ are guaranteed to be pair-wise consistent and lie on a single (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$, where $Q^\star(x, y) = Q(x, y)$ for an *honest* D. This will further guarantee that the polynomials of all *honest* parties in $\mathcal{F}$ *also* lie on $Q^\star(x, y)$, as $\mathcal{F} \subseteq \mathcal{W}$.

If a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted, then each $P_i \in \mathcal{W}$ outputs the constant term of its univariate polynomial as its share. On the other hand, the parties outside $\mathcal{W}$ attempt to obtain their corresponding polynomials lying on $Q^\star(x, y)$ by applying OEC on the common points on these polynomials received from the parties in $\mathcal{F}$. And if a t_s-degree polynomial is obtained, then the constant term of the polynomial is set as the share. For an *honest* D, each *honest* P_i will be present in $\mathcal{W}$ and hence will have the share $q(\alpha_i)$. On the other hand, if a $\mathcal{W}$ is accepted for a *corrupt* D, then all the *honest* parties in $\mathcal{W}$ (which are at least $t_s + 1$ in number) will have their shares lying on $q^\star(\cdot) = Q^\star(0, y)$. Moreover, even if an *honest* party P_i *outside* $\mathcal{W}$ is able to compute its share, then it is the same as $q^\star(\alpha_i)$ due to the OEC mechanism. However, for a *corrupt* D, all the *honest* parties *outside* $\mathcal{W}$ may *not* be able to obtain their desired share, as $\mathcal{F}$ is guaranteed to have only $n - 2t_s > t_s + t_a$ *honest* parties and OEC may fail. It is precisely for this reason that Π_{SWPS} *fails* to qualify as a VSS.

$\Pi_{\mathsf{SWPS}} + \Pi_{\mathsf{AWPS}} \Rightarrow$ best-of-both-worlds WPS Protocol Π_{WPS}: We next discuss how to combine protocols Π_{SWPS} and Π_{AWPS} to get our best-of-both-worlds WPS protocol called Π_{WPS}. In protocol Π_{WPS} (Fig 3), the parties first run Π_{SWPS} *assuming* a *synchronous* network, where Π_{BC} is used to make any value public by setting $t = t_s$ in the protocol Π_{BC}. If D is *honest* then in a *synchronous* network, the first, second, third and fourth phase of Π_{SWPS} would have been over by time $\Delta, 2\Delta, 2\Delta + T_{\mathsf{BC}}$ and $2\Delta + 2T_{\mathsf{BC}}$ respectively and by time $2\Delta + 2T_{\mathsf{BC}}$, the parties should have accepted a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$. However, in an *asynchronous* network, parties may have different "opinion" regarding the acceptance of a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$. This is because it may be possible that only a subset of the honest parties accept a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ within local time $2\Delta + 2T_{\mathsf{BC}}$. Hence at time $2\Delta + 2T_{\mathsf{BC}}$, the parties run an instance of our best-of-both-worlds BA protocol Π_{BA}, to check whether any $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted.

If the parties conclude that a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted, then the parties compute their *WPS-shares* as per Π_{SWPS}. However, we need to ensure that for a *corrupt* D in a *synchronous* network, if the polynomials of the *honest* parties in $\mathcal{W}$ are *not* pair-wise consistent, then the corresponding conflicting `NOK` messages are received within time $2\Delta + T_{\mathsf{BC}}$ (the time required for the *third* phase of Π_{SWPS} to be over), so that $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is *not* accepted. This is ensured by enforcing even a *corrupt* D to send the respective polynomials of all the *honest* parties in $\mathcal{W}$ by time Δ, so that the pair-wise consistency test between every pair of *honest* parties in $\mathcal{W}$ is over by time 2Δ. For this, the parties are asked to wait for some "appropriate" time, before starting the pair-wise consistency tests and also before making public the results of pair-wise consistency tests. The idea is to ensure that

if the polynomials of the *honest* parties in $\mathcal{W}$ are *not* delivered within time Δ (in a *synchronous* network), then the results of the pair-wise consistency tests also get *delayed* beyond time $2\Delta + T_{\mathsf{BC}}$ (the time-out of the *third* phase of Π_{SWPS}). This in turn will ensure that no $\mathcal{W}$ is accepted within time $2\Delta + 2T_{\mathsf{BC}}$.

If the parties conclude that no $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted within time $2\Delta + 2T_{\mathsf{BC}}$, then it implies that either D is *corrupt* or the network is *asynchronous* and hence the parties resort to Π_{AWPS}. However, D *need not* have to start afresh and distribute polynomials on a "fresh" bivariate polynomial. Instead, D continues with the consistency graph formed using the $\mathtt{OK}$ messages received as part of Π_{SWPS} and searches for an $(n, t_a)-$star. If D is *honest* and the network is *asynchronous*, then the parties eventually obtain their shares.

Notice that in an *asynchronous* network, it might be possible that the parties (through Π_{BA}) conclude that $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted, if some honest party(ies) accepts a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ within the time-out $2\Delta + 2T_{\mathsf{BC}}$. Even in this case, the polynomials of all *honest* parties in $\mathcal{W}$ lie on a single (t_s, t_s)-degree symmetric bivariate polynomial $Q^\star(x, y)$ for a *corrupt* D. This is because there will be at least $n - 2t_s - t_a > t_s$ *honest* parties in $\mathcal{E}$ with pair-wise consistent polynomials, defining $Q^\star(x, y)$, and the polynomial of every *honest* party in $\mathcal{F}$ will be pair-wise consistent with the polynomials of *all honest* parties in $\mathcal{E}$ and hence lie on $Q^\star(x, y)$ as well. Now consider any *honest* $P_i \in (\mathcal{W} \setminus \mathcal{F})$. As part of Π_{SWPS}, it is ensured that the polynomial of P_i is consistent with the polynomials of at least $n - t_s$ parties among $\mathcal{W}$. Among these $n - t_s$ parties, at least $n - 2t_s - t_a > t_s$ will be *honest* parties from $\mathcal{F}$. Thus, the polynomial of P_i will also lie on $Q^\star(x, y)$.

Protocol $\Pi_{\mathsf{WPS}}(\mathsf{D}, q(\cdot))$

- **Phase I — Sending Polynomials**:
  - D on having the input $q(\cdot)$, chooses a random (t_s, t_s)-degree symmetric bivariate polynomial $Q(x, y)$ such that $Q(0, y) = q(\cdot)$ and sends $q_i(x) = Q(x, \alpha_i)$ to each party $P_i \in \mathcal{P}$.
- **Phase II — Pair-Wise Consistency**: Each $P_i \in \mathcal{P}$ on receiving a t_s-degree polynomial $q_i(x)$ from D does the following.
  - Wait till the local time becomes a multiple of Δ and then send $q_{ij} = q_i(\alpha_j)$ to P_j, for $j = 1, \ldots, n$.
- **Phase III — Publicly Declaring the Results of Pair-Wise Consistency Test**: Each $P_i \in \mathcal{P}$ does the following.
  - Upon receiving q_{ji} from P_j, wait till the local time becomes a multiple of Δ. If a t_s-degree polynomial $q_i(x)$ has been received from D, then do the following.
    - Broadcast $\mathtt{OK}(i, j)$, if $q_{ji} = q_i(\alpha_j)$ holds.
    - Broadcast $\mathtt{NOK}(i, j, q_i(\alpha_j))$, if $q_{ji} \neq q_i(\alpha_j)$ holds.
- **Local computation — Constructing Consistency Graph:** Each $P_i \in \mathcal{P}$ does the following.
  - Construct a *consistency graph* G_i over $\mathcal{P}$, where the edge (P_j, P_k) is included in G_i, if $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ is received from the broadcast of P_j and P_k respectively, either through the regular-mode or fall-back mode.
- **Phase IV — Checking for an $(n, t_s)-$star**: D does the following in its consistency graph G_{D} at time $2\Delta + T_{\mathsf{BC}}$.
  - Remove edges incident with P_i, if $\mathtt{NOK}(i, j, q_{ij})$ is received from the broadcast of P_i through regular-mode and $q_{ij} \neq Q(\alpha_j, \alpha_i)$.
  - Set $\mathcal{W} = \{P_i : \mathsf{deg}(P_i) \geq n - t_s\}$, where $\mathsf{deg}(P_i)$ denotes the degree of P_i in G_{D}.
  - Remove P_i from $\mathcal{W}$, if P_i is *not* incident with at least $n - t_s$ parties in $\mathcal{W}$. Repeat this step till no more parties can be removed from $\mathcal{W}$.
  - Run algorithm $\mathsf{AlgStar}$ on $G_{\mathsf{D}}[\mathcal{W}]$, where $G_{\mathsf{D}}[\mathcal{W}]$ denotes the subgraph of G_{D} induced by the vertices in $\mathcal{W}$. If an $(n, t_s)-$star, say $(\mathcal{E}, \mathcal{F})$, is obtained, then broadcast $(\mathcal{W}, \mathcal{E}, \mathcal{F})$.
- **Local Computation — Verifying and Accepting $(\mathcal{W}, \mathcal{E}, \mathcal{F})$:** Each $P_i \in \mathcal{P}$ does the following at time $2\Delta + 2T_{\mathsf{BC}}$.
  - If a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is received from D's broadcast through regular-mode, then *accept* it if following *were true* at time $2\Delta + T_{\mathsf{BC}}$:

- There exist no $P_j, P_k \in \mathcal{W}$, such that $\texttt{NOK}(j, k, q_{jk})$ and $\texttt{NOK}(k, j, q_{kj})$ messages *were* received from the broadcast of P_j and P_k respectively through regular-mode, where $q_{jk} \neq q_{kj}$.
- In the consistency graph G_i, $\mathsf{deg}(P_j) \geq n - t_s$ for all $P_j \in \mathcal{W}$.
- In the consistency graph G_i, every $P_j \in \mathcal{W}$ has edges with at least $n - t_s$ parties from $\mathcal{W}$.
- $(\mathcal{E}, \mathcal{F})$ *was* an $(n, t_s)-$star in the induced graph $G_i[\mathcal{W}]$.
- For every $P_j, P_k \in \mathcal{W}$ where the edge (P_j, P_k) is present in G_i, the $\texttt{OK}(j, k)$ and $\texttt{OK}(k, j)$ messages *were* received from the broadcast of P_j and P_k respectively, through regular-mode.

- **Phase V — Deciding Whether to Go for an $(n, t_a)-$star**: At time $2\Delta + 2T_{\mathsf{BC}}$, each $P_i \in \mathcal{P}$ participates in an instance of Π_{BA} with input $b_i = 0$ if a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ was accepted, else with input $b_i = 1$, and waits for time T_{BA}.
- **Local Computation — Computing WPS-share Through $\mathcal{W}$**: If the output of Π_{BA} is 0, then each $P_i \in \mathcal{P}$ computes its *WPS-Share* s_i (initially set to $\perp$) as follows.
  - If a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is not yet received then wait till a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is received from D's broadcast through fall-back mode.
  - If $P_i \in \mathcal{W}$, then output $s_i = q_i(0)$.
  - Else, initialise a support set $\mathcal{SS}_i$ to $\emptyset$. If q_{ji} is received from $P_j \in \mathcal{F}$, include q_{ji} to $\mathcal{SS}_i$. Keep executing $\mathsf{OEC}(t_s, t_s, \mathcal{SS}_i)$, till a t_s-degree polynomial, say $q_i(\cdot)$, is obtained. Then, output $s_i = q_i(0)$.
- **Phase VI — Broadcasting an $(n, t_a)-$star**: If the output of Π_{BA} is 1, then D does the following.
  - After every update in the consistency graph G_{D}, run $\mathsf{AlgStar}$ on G_{D}. If an $(n, t_a)-$star, say $(\mathcal{E}', \mathcal{F}')$, is obtained, then broadcast $(\mathcal{E}', \mathcal{F}')$.
- **Local Computation — Computing WPS-share Through $(n, t_a)-$star**: If the output of Π_{BA} is 1, then each P_i does the following to compute its *WPS-Share*.
  - Waits till an $(n, t_a)-$star $(\mathcal{E}', \mathcal{F}')$ is obtained from the broadcast of D, either through regular or fall-back mode. Upon receiving, wait till $(\mathcal{E}', \mathcal{F}')$ becomes an $(n, t_a)-$star in G_i.
  - If $P_i \in \mathcal{F}'$, then output $s_i = q_i(0)$.
  - Else, initialise a support set $\mathcal{SS}_i$ to $\emptyset$. If q_{ji} is received from $P_j \in \mathcal{F}'$, include q_{ji} to $\mathcal{SS}_i$. Keep executing $\mathsf{OEC}(t_s, t_s, \mathcal{SS}_i)$, till a t_s-degree polynomial, say $q_i(\cdot)$, is obtained. Then, output $s_i = q_i(0)$.

Figure 3: The best-of-both-worlds weak polynomial-sharing protocol for a single polynomial.

We next proceed to prove the properties of the protocol Π_{WPS}. We begin with showing that if D is *honest*, then the adversary does not learn anything additional about $q(\cdot)$, irrespective of the network type.

Lemma 4.1 (t_s*-Privacy).* *In protocol Π_{WPS}, if D is honest, then irrespective of the network type, the view of the adversary remains independent of $q(\cdot)$.*

Proof. Let D be *honest*. We consider the worst case scenario, when the adversary controls up to t_s parties. We claim that throughout the protocol, the adversary learns at most t_s univariate polynomials lying on $Q(x, y)$. Since $Q(x, y)$ is a random (t_s, t_s)-degree- symmetric-bivariate polynomial, it then follows from Lemma 2.2 that the view of the adversary will be independent of $q(\cdot)$. We next proceed to prove the claim.

Corresponding to every *corrupt* P_i, the adversary learns $Q(x, \alpha_i)$. Corresponding to every *honest* P_i, the adversary learns t_s distinct points on P_i's univariate polynomial $Q(x, \alpha_i)$, through the pair-wise consistency checks. However, these points were already included in the view of the adversary (through the univariate polynomials under adversary's control). Hence no additional information about the polynomials of the honest parties is revealed during the pair-wise consistency checks. Furthermore, no *honest* P_i ever broadcasts $\texttt{NOK}(i, j, q_i(\alpha_j))$, corresponding to any *honest* P_j. This is because the pair-wise consistency check will always pass for every pair of *honest* parties. □

We next prove the correctness property in a *synchronous* network.

Lemma 4.2 (t_s*-Correctness).* *In protocol* Π_{WPS}*, if* D *is honest and the network is synchronous, then each honest* P_i *outputs* $q(\alpha_i)$ *at time* $T_{\mathsf{WPS}} = 2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$.

Proof. Let D be *honest* and the network be *synchronous* with up to t_s corruptions. During phase I, every *honest* party P_j receives $q_j(x) = Q(x, \alpha_j)$ from D within time Δ. Hence during phase II, every *honest* P_j sends q_{jk} to every P_k, which takes at most Δ time to be delivered. Hence, by time 2Δ, every *honest* P_j receives q_{kj} from every *honest* P_k, such that $q_{kj} = q_j(\alpha_k)$ holds. Consequently, during phase III, every *honest* P_j broadcasts $\mathtt{OK}(j,k)$ corresponding to every *honest* P_k, and vice versa. From the t_s*-validity* property of Π_{BC} in the *synchronous* network, it follows that every honest P_i receives $\mathtt{OK}(j,k)$ and $\mathtt{OK}(k,j)$ from the broadcast of every *honest* P_j and every *honest* P_k respectively, through regular-mode, at time $2\Delta + T_{\mathsf{BC}}$. Hence, the edge (P_j, P_k) will be added to the consistency graph G_i, corresponding to every honest P_j, P_k. Furthermore, from the t_s*-consistency* property of Π_{BC}, the graph G_i will be the same for every honest party P_i (including D) at time $2\Delta + T_{\mathsf{BC}}$. Moreover, if D receives an *incorrect* $\mathtt{NOK}(i,j,q_{ij})$ message from the broadcast of any *corrupt* P_i through regular-mode at time $2\Delta + T_{\mathsf{BC}}$, where $q_{ij} \neq Q(\alpha_j, \alpha_i)$, then D removes all the edges incident with P_i in D's consistency graph G_{D}. Dealer D then computes the set $\mathcal{W}$, and all *honest* parties will be present in $\mathcal{W}$. Moreover, the honest parties will form a clique of size at least $n - t_s$ in the induced subgraph $G_{\mathsf{D}}[\mathcal{W}]$ at time $2\Delta + T_{\mathsf{BC}}$ and D will find an $(n, t_s)-$star, say $(\mathcal{E}, \mathcal{F})$, in $G_{\mathsf{D}}[\mathcal{W}]$ and broadcast $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ during phase IV. By the t_s*-validity* of Π_{BC} in the *synchronous* network, all honest parties will receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ through regular-mode at time $2\Delta + 2T_{\mathsf{BC}}$. Moreover, all honest parties will *accept* $(\mathcal{E}, \mathcal{F})$ and participate with input 0 in the instance of Π_{BA}. Hence, by the t_s*-validity* and t_s*-guaranteed liveness* of Π_{BA} in the *synchronous* network, *every* honest party obtains the output 0 in the instance of Π_{BA}, by time $2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Now, consider an arbitrary *honest* party P_i. Since $P_i \in \mathcal{W}$, party P_i outputs $s_i = q_i(0) = Q(0, \alpha_i) = q(\alpha_i)$. □

We next prove the correctness property in an *asynchronous* network.

Lemma 4.3 (t_a*-Correctness).* *In protocol* Π_{WPS}*, if* D *is honest and network is asynchronous, then almost-surely, each honest* P_i *eventually outputs* $q(\alpha_i)$.

Proof. Let D be *honest* and network be *asynchronous* with up to t_a corruptions. We first note that every honest party participates with some input in the instance of Π_{BA} at local time $2\Delta + 2T_{\mathsf{BC}}$. Hence from the t_a*-almost-surely liveness* and t_a*-consistency* of Π_{BA} in an *asynchronous* network, it follows that almost-surely, the instance of Π_{BA} eventually generates some common output, for all honest parties. Now there are two possible cases:

- **The output of** Π_{BA} **is 0**: From the t_a*-validity* of Π_{BA} in the *asynchronous* network, it follows that at least one *honest* party, say P_h, participated with input 0 during the instance of Π_{BA}. This implies that P_h has accepted a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ at local $2\Delta + 2T_{\mathsf{BC}}$, which is received from the broadcast of D, through *regular-mode*. Hence, by the t_a*-weak validity* and t_a*-fallback validity* properties of Π_{BC} in the *asynchronous* network, all honest parties will eventually receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D and *accept* the triplet. This is because the consistency graphs of all honest parties will eventually have all the edges which were present in the consistency graph G_h of P_h, at time $2\Delta + 2T_{\mathsf{BC}}$. We claim that *every honest* P_i will eventually get $Q(x, \alpha_i)$. This will imply that eventually, every *honest* P_i outputs $s_i = Q(0, \alpha_i) = q(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are two possible cases.
  - $P_i \in \mathcal{W}$: In this case, P_i already has received $Q(x, \alpha_i)$ from D.
  - $P_i \notin \mathcal{W}$: In this case, there will be at least $n - t_s > 2t_s + t_a$ parties in $\mathcal{F}$, of which at most t_a could be *corrupt*. Since $Q(x, \alpha_i)$ is a t_s-degree polynomial and $t_s < |\mathcal{F}| - 2t_a$, from

Lemma A.1, it follows that by applying the OEC procedure on the common points on the polynomials $Q(x, \alpha_i)$, received from the parties in $\mathcal{F}$, party P_i will eventually obtain $Q(x, \alpha_i)$.

- **The output of Π_{BA} is 1**: Since D is *honest*, every pair of honest parties P_j, P_k eventually broadcast $\mathtt{OK}(j,k)$ and $\mathtt{OK}(k,j)$ messages respectively, as the pair-wise consistency check between them will eventually be successful. From the t_a-*weak validity* and t_a-*fallback validity* of Π_{BC}, these messages are eventually delivered to every honest party. Also from the t_a-*weak consistency* and t_a-*fallback consistency* of Π_{BC} in the *asynchronous* network, any $\mathtt{OK}$ message which is received by D from the broadcast of any *corrupt* party, will be eventually received by every other honest party as well. As there will be at least $n - t_a$ honest parties, a clique of size at least $n - t_a$ will eventually form in the consistency graph of every honest party. Hence D will eventually find an $(n, t_a)-$star, say $(\mathcal{E}', \mathcal{F}')$, in its consistency graph and broadcast it. From the t_a-*weak validity* and t_a-*fallback validity* of Π_{BC}, this star will be eventually delivered to every honest party. Moreover, $(\mathcal{E}', \mathcal{F}')$ will be eventually an $(n, t_a)-$star in every honest party's consistency graph. We claim that *every honest* P_i will eventually get $Q(x, \alpha_i)$. This will imply that eventually, every *honest* P_i outputs $s_i = Q(0, \alpha_i) = q(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are two possible cases.
  - $P_i \in \mathcal{F}'$: In this case, P_i already has $Q(x, \alpha_i)$, received from D.
  - $P_i \notin \mathcal{F}'$: In this case, there will be at least $n - t_a > 3t_s$ parties in $\mathcal{F}'$, of which at most t_a could be *corrupt*, where $t_a < t_s$. Since $Q(x, \alpha_i)$ is a t_s-degree polynomial and $t_s < |\mathcal{F}'| - 2t_a$, from Lemma A.1 it follows that by applying the OEC procedure on the common points on the polynomial $Q(x, \alpha_i)$, received from the parties in $\mathcal{F}'$, party P_i will eventually obtain $Q(x, \alpha_i)$.

□

We next proceed to prove the weak commitment properties for a *corrupt* D. However, before that we prove a helping lemma.

Lemma 4.4. *Let the network be synchronous and let D be corrupt in the protocol Π_{WPS}. If any one honest party receives a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepts it at time $2\Delta + 2T_{\mathsf{BC}}$, then all the following hold.*

- *All honest parties in $\mathcal{W}$ receive their respective t_s-degree univariate polynomials from D, within time Δ.*
- *The univariate polynomials $q_i(x)$ of all honest parties P_i in the set $\mathcal{W}$ lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$.*
- *Within time $2\Delta + 2T_{\mathsf{BC}}$, every honest party accepts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$.*

Proof. Let D be *corrupt* and network be *synchronous* with up to t_s corruptions. As per the lemma condition, let P_h be an *honest* party, who receives a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepts it at time $2\Delta + 2T_{\mathsf{BC}}$. Then from the protocol steps, the following must be true for P_h at time $2\Delta + T_{\mathsf{BC}}$:

- There does not exist any $P_j, P_k \in \mathcal{W}$, such that $\mathtt{NOK}(j, k, q_{jk})$ and $\mathtt{NOK}(k, j, q_{kj})$ messages are received by P_h, from the broadcast of P_j and P_k respectively through regular-mode, where $q_{jk} \neq q_{kj}$.
- In P_h's consistency graph G_h, $\mathsf{deg}(P_j) \geq n - t_s$ for all $P_j \in \mathcal{W}$ and P_j has edges with at least $n - t_s$ parties from $\mathcal{W}$.
- $(\mathcal{E}, \mathcal{F})$ constitutes an $(n, t_s)-$star in the induced subgraph $G_h[\mathcal{W}]$, such that for every $P_j, P_k \in \mathcal{W}$ where the edge (P_j, P_k) is present in G_h, the messages $\mathtt{OK}(j,k)$ and $\mathtt{OK}(k,j)$ are received by P_h, from the broadcast of P_j and P_k respectively, through regular-mode.

We prove the first part of the lemma through a contradiction. So let $P_j \in \mathcal{W}$ be an *honest* party, who receives its t_s-degree univariate polynomial, say $q_j(x)$, from D at time $\Delta + \delta$, where $\delta > 0$. Moreover, let $P_k \in \mathcal{W}$ be an *honest* party, different from P_j (note that there are at least $n - 2t_s$ *honest* parties in $\mathcal{W}$). As stated above, at time $2\Delta + T_{\mathsf{BC}}$, party P_h receives the message $\mathtt{OK}(k, j)$ from the broadcast of P_k through regular-mode. From the protocol steps, P_j waits till its local time becomes a multiple of Δ, before it sends the points on its polynomial to other parties for pair-wise consistency tests. Hence, P_j must have started sending the points after time $c \cdot \Delta$, where $c \geq 2$. Since the network is *synchronous*, the point $q_{jk} = q_j(\alpha_k)$ must have been received by P_k by time $(c + 1) \cdot \Delta$. Moreover, from the protocol steps, even if P_k receives these points at time T, where $c \cdot \Delta < T < (c + 1) \cdot \Delta$, it waits till time $(c + 1) \cdot \Delta$, before broadcasting the $\mathtt{OK}(k, j)$ message. Since P_k is *honest*, from the t_s-*validity* property of Π_{BC} in the *synchronous* network, it will take *exactly* T_{BC} time for the message $\mathtt{OK}(k, j)$ to be received through regular-mode, once it is broadcast. This implies that P_h will receive the message $\mathtt{OK}(k, j)$ at time $(c + 1) \cdot \Delta + T_{\mathsf{BC}}$, where $(c+1) > 2$. However, this is a contradiction, since the $\mathtt{OK}(k, j)$ message has been received by P_h at time $2\Delta + T_{\mathsf{BC}}$.

To prove the second part of the lemma, we will show that the univariate polynomials $q_j(x)$ of all the *honest* parties in $\mathcal{W}$ are pair-wise consistent. Since there are at least $n-2t_s > t_s$ *honest* parties in $\mathcal{W}$, from Lemma 2.1, it follows that the univariate polynomials $q_j(x)$ of all the *honest* parties in $\mathcal{W}$ lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. So consider an arbitrary pair of *honest* parties $P_j, P_k \in \mathcal{W}$. From the first part of the lemma, both P_j and P_k must have received their respective univariate polynomials $q_j(x)$ and $q_k(x)$ by time Δ. This further implies that P_j and P_k must have received the points $q_{kj} = q_k(\alpha_j)$ and $q_{jk} = q_j(\alpha_k)$ respectively by time 2Δ. If $q_{kj} \neq q_{jk}$, then P_j and P_k would broadcast $\mathtt{NOK}(j, k, q_{jk})$ and $\mathtt{NOK}(k, j, q_{kj})$ messages respectively, at time 2Δ. Consequently, from the t_s-*validity* property of Π_{BC} in the *synchronous* network, P_h will receive these messages through regular-mode at time $2\Delta + T_{\mathsf{BC}}$. Consequently, P_h *will not* accept $(\mathcal{W}, \mathcal{E}, \mathcal{F})$, which is a contradiction.

To prove the third part of the lemma, we note that since P_h has received $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode at time $2\Delta + 2T_{\mathsf{BC}}$, it implies that D must have started broadcasting $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ latest at time $2\Delta + T_{\mathsf{BC}}$. This is because it takes T_{BC} time for the regular-mode of Π_{BC} to produce an output. From the t_s-*consistency* property of Π_{BC} in the *synchronous* network, it follows that every honest party will also receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode, at time $2\Delta+2T_{\mathsf{BC}}$. Since at time $2\Delta+T_{\mathsf{BC}}$, party P_h has received the $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ messages through regular-mode from the broadcast of every $P_j, P_k \in \mathcal{W}$ where (P_j, P_k) is an edge in P_h's consistency graph, it follows that these messages started getting broadcast, latest at time 2Δ. From the t_s-*validity* and t_s-*consistency* properties of Π_{BC} in the *synchronous* network, it follows that every honest party receives these broadcast messages through regular-mode at time $2\Delta + T_{\mathsf{BC}}$. Hence $(\mathcal{E}, \mathcal{F})$ will constitute an $(n, t_s)-$star in the induced subgraph $G_i[\mathcal{W}]$ of every honest party P_i's consistency-graph at time $2\Delta+T_{\mathsf{BC}}$ and consequently, every honest party accepts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$. □

Now based on the above helping lemma, we proceed to prove the weak commitment properties of the protocol Π_{WPS}.

Lemma 4.5 (t_s-*Weak Commitment*)**.** *In protocol* Π_{WPS}*, if* D *is corrupt and network is synchronous, then either no honest party computes any output or there exists some* t_s*-degree polynomial, say* $q^\star(\cdot)$*, such that all the following hold.*

- *There are at least* $t_s + 1$ *honest parties* P_i *who output the WPS-shares* $q^\star(\alpha_i)$*.*
- *If any honest* P_j *outputs a WPS-share* $s_j \in \mathbb{F}$*, then* $s_j = q^\star(\alpha_j)$ *holds.*

Proof. Let D be *corrupt* and network be *synchronous* with up to t_s corruptions. If no honest party outputs any wps-share, then the lemma holds trivially. So consider the case when some honest party outputs a wps-share, which is an element of $\mathbb{F}$. Now, there are two possible cases.

- *At time* $2\Delta + 2T_{\mathsf{BC}}$*, at least one honest party, say* P_h*, accepts a* $(\mathcal{W}, \mathcal{E}, \mathcal{F})$*, received from the broadcast of* D *through regular-mode*: In this case, from Lemma 4.4, at time $2\Delta + 2T_{\mathsf{BC}}$, every honest party will accept $(\mathcal{W}, \mathcal{E}, \mathcal{F})$. Hence every honest party participates in the instance of Π_{BA} with input 0. From the t_s*-validity* and t_s*-guaranteed liveness* properties of Π_{BA} in the *synchronous* network, all honest parties will get the output 0 during the instance of Π_{BA} by time $T_{\mathsf{WPS}} = 2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. From Lemma 4.4, the univariate polynomials of all the *honest* parties in $\mathcal{W}$ will lie on some (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. Now consider an arbitrary *honest* party P_i, who outputs a wps-share $s_i \in \mathbb{F}$. We want to show that the condition $s_i = q^\star(\alpha_i)$ holds. And there are at least $t_s + 1$ such *honest* parties P_i who output their wps-share. There are two possible cases.
  - $P_i \in \mathcal{W}$: From the protocol steps, P_i sets $s_i = Q^\star(0, \alpha_i)$, which is the same as $q^\star(\alpha_i)$. Since $\mathcal{W}$ contains at least $t_s + t_a + 1$ honest parties, this also shows that at least $t_s + 1$ honest parties P_i output their respective wps-share $s_i \in \mathbb{F}$, which is the same as $q^\star(\alpha_i)$.
  - $P_i \notin \mathcal{W}$: In this case, P_i sets $s_i = q_i(0)$, where $q_i(\cdot)$ is a t_s-degree univariate polynomial, obtained by applying the OEC procedure with $d = t = t_s$, on the values q_{ji}, received from the parties $P_j \in \mathcal{F}$, during the pair-wise consistency checks. Note that as part of OEC (see the proof of Lemma A.1), party P_i verifies that at least $2t_s + 1$ q_{ji} values from the parties in $\mathcal{F}$ lie on $q_i(\cdot)$. Now out of these $2t_s + 1$ q_{ji} values, at least $t_s + 1$ values are from the *honest* parties in $\mathcal{F}$. Furthermore, these q_{ji} values from the *honest* parties in $\mathcal{F}$ are the same as $Q^\star(\alpha_i, \alpha_j)$, which is equal to $Q^\star(\alpha_j, \alpha_i)$ and uniquely determine $Q^\star(x, \alpha_i)$; the last property holds since $Q^\star(x, y)$ is a symmetric bivariate polynomial. This automatically implies that $q_i(x)$ is the *same* as $Q(x, \alpha_i)$ and hence $s_i = q^\star(\alpha_i)$, since two different t_s-degree polynomials can have at most t_s common values.
- *At time* $2T_{\mathsf{BC}} + 2\Delta$*, no honest party has accepted any* $(\mathcal{W}, \mathcal{E}, \mathcal{F})$: This implies that all honest parties participate in the instance of Π_{BA} with input 1. So by the t_s*-validity* and t_s*-guaranteed liveness* of Π_{BA} in the *synchronous* network, all honest parties obtain the output 1 in the instance of Π_{BA}. Let P_h be the *first honest* party who outputs a wps-share, consisting of an element from $\mathbb{F}$. This means that P_h has received a pair $(\mathcal{E}', \mathcal{F}')$, from the broadcast of D, such that $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph. By the t_s*-consistency* and t_s*-fallback consistency* properties of Π_{BC} in the synchronous network, *all* honest parties receive $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D. Moreover, since $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph, it will also constitute an $(n, t_a)-$star in every other honest party's consistency graph as well. This is because the $\mathtt{OK}(\star, \star)$ messages which are received by P_h from the broadcast of the various parties in $\mathcal{E}'$ and $\mathcal{F}'$, are also received by every other honest party, either through regular-mode or fallback-mode. The last property follows from the t_s*-validity,* t_s*-consistency* and t_s*-fallback consistency* properties of Π_{BC} in the synchronous network. Since $|\mathcal{E}'| \geq n - 2t_a > 2t_s + (t_s - t_a) > 2t_s$, it follows that $\mathcal{E}'$ has at least $t_s + 1$ *honest* parties P_i, whose univariate polynomials $q_i(x)$ are pair-wise consistent. Hence, from Lemma 2.1, these univariate polynomials lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Similarly, since the univariate polynomial $q_i(x)$ of every *honest* party in $\mathcal{F}'$ is pair-wise consistent with the univariate polynomials $q_j(x)$ of the *honest* parties in $\mathcal{E}'$, it implies that the univariate polynomials $q_i(x)$ of all the *honest* parties in $\mathcal{F}'$ also lie on $Q^\star(x, y)$. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. We show that *every honest* P_i outputs a wps-share, which is the same as $q^\star(\alpha_i)$. For this it is enough to show that each honest P_i gets $q_i(x) = Q^\star(x, \alpha_i)$,

as P_i outputs $q_i(0)$ as its wps-share, which will be then same as $q^\star(\alpha_i)$. Consider an arbitrary *honest* party P_i. There are two possible cases.

- $P_i \in \mathcal{F}'$: In this case, P_i already has $Q^\star(x, \alpha_i)$, received from D.
- $P_i \notin \mathcal{F}'$: In this case, there will be $n - t_a > 3t_s$ parties in $\mathcal{F}'$, of which at most t_s could be corrupt. Moreover, $Q^\star(x, \alpha_i)$ is a t_s-degree polynomial and $t_s < |\mathcal{F}'| - 2t_s$ holds. Hence from the properties of OEC (Lemma A.1), by applying the OEC procedure on the common points on the polynomial $Q^\star(x, \alpha_i)$ received from the parties in $\mathcal{F}'$, party P_i will compute $Q^\star(x, \alpha_i)$.

□

We finally prove the commitment property in an *asynchronous* network.

Lemma 4.6 (t_a*-Strong Commitment).* *In protocol* Π_{WPS}*, if* D *is corrupt and network is asynchronous, then either no honest party computes any output or there exist some* t_s*-degree polynomial, say* $q^\star(\cdot)$*, such that almost-surely, every honest* P_i *eventually outputs a wps-share* $q^\star(\alpha_i)$*.*[10]

Proof. Let D be *corrupt* and network be *asynchronous* with up to t_a corruptions. If no honest party computes any output, then the lemma holds trivially. So consider the case when some honest party outputs a wps-share, consisting of an element of $\mathbb{F}$. We note that every honest party participates with some input in the instance of Π_{BA} at local time $2\Delta + 2T_{\mathsf{BC}}$. Hence, from the t_a*-almost-surely liveness* and t_a*-consistency* properties of Π_{BA} in the *asynchronous* network, almost-surely, all honest parties eventually compute a common output during the instance of Π_{BA}. Now there are two possible cases:

- **The output of** Π_{BA} **is 0**: From the t_a*-validity* of Π_{BA} in the *asynchronous* network, it implies that at least one honest party, say P_h, participated with input 0 during the instance of Π_{BA}. This further implies that at local time $2\Delta+2T_{\mathsf{BC}}$, party P_h has accepted a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$, which has been received by P_h from the broadcast of D, through regular-mode. Hence, by the t_a*-weak consistency* and t_a*-fallback consistency* of Π_{BC} in the *asynchronous* network, all honest parties will eventually receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D. There will be at least $n - 2t_s - t_a > t_s$ *honest* parties in $\mathcal{E}$, whose univariate polynomials $q_i(x)$ are pair-wise consistent and hence from Lemma 2.1 lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Similarly, the univariate polynomial $q_j(x)$ of every *honest* $P_j \in \mathcal{F}$ will be pair-wise consistent with the univariate polynomials $q_i(x)$ of all the *honest* parties in $\mathcal{E}$ and hence lie on $Q^\star(x, y)$ as well. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. We claim that *every honest* P_i will eventually get $Q^\star(x, \alpha_i)$. This will imply that eventually every *honest* P_i outputs the wps-share $s_i = Q^\star(0, \alpha_i) = q^\star(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are three possible cases.
  - $P_i \in \mathcal{W}$ *and* $P_i \in \mathcal{F}$: In this case, P_i already has $q_i(x)$, received from D. And since $P_i \in \mathcal{F}$, the condition $q_i(x) = Q^\star(x, \alpha_i)$ holds.
  - $P_i \in \mathcal{W}$ *and* $P_i \notin \mathcal{F}$: In this case, P_i already has $q_i(x)$, received from D. Since $|\mathcal{W}| \geq n - t_s$ and $|\mathcal{F}| \geq n - t_s$, $|\mathcal{W} \cap \mathcal{F}| \geq n - 2t_s > t_s + t_a$. From the protocol steps, the polynomial $q_i(x)$ is pair-wise consistent with the polynomial $q_j(x)$ of at least $n - t_s$ parties $P_j \in \mathcal{W}$ (since P_i has edges with at least $n - t_s$ parties P_j within $\mathcal{W}$). Now among these $n - t_s$ parties, at least $n - 2t_s$ parties will be from $\mathcal{F}$, of which at least $n - 2t_s - t_a > t_s$ parties will be *honest*. Hence, $q_i(x)$ is pair-wise consistent with the $q_j(x)$ polynomials of at least

[10] Note that *unlike* the synchronous network, the commitment property in the *asynchronous* network is *strong*. That is, if at all any honest party outputs a wps-share, then *all* the honest parties are guaranteed to eventually output their wps-shares.

t_s+1 *honest* parties $P_j \in \mathcal{F}$. Now since the $q_j(x)$ polynomial of all the *honest* parties in $\mathcal{F}$ lie on $Q^\star(x,y)$, it implies that $q_i(x) = Q^\star(x,\alpha_i)$ holds.

- $P_i \notin \mathcal{W}$: In this case, there will be $n - t_s$ parties in $\mathcal{F}$, of which at most t_a could be *corrupt*. Since $Q^\star(x,\alpha_i)$ is a t_s-degree polynomial, and $t_s < |\mathcal{F}| - 2t_a$, from Lemma A.1 it follows that by applying the OEC procedure on the common points on the $Q^\star(x,\alpha_i)$ polynomial received from the parties in $\mathcal{F}$, party P_i will eventually obtain $Q^\star(x,\alpha_i)$.

- **The output of Π_{BA} is 1**: Let P_h be the *first honest* party, who outputs a wps-share. This means that P_h has received a pair, say $(\mathcal{E}', \mathcal{F}')$, from the broadcast of D, such that $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph. By the t_a-*weak consistency* and t_a-*fallback consistency* properties of Π_{BC} in the *asynchronous* network, *all* honest parties eventually receive $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D. Moreover, since the consistency graphs are constructed based on the broadcast $\mathtt{OK}$ messages and since $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph, from the t_a-*weak validity,* t_a-*fallback validity,* t_a-*weak consistency and* t_a-*fallback consistency* properties of Π_{BC} in the *asynchronous* network, the pair $(\mathcal{E}', \mathcal{F}')$ will eventually constitute an $(n, t_a)-$star in every honest party's consistency graph. Since $|\mathcal{E}'| \geq n - 2t_a > 2t_s + (t_s - t_a) > 2t_s$, it follows that $\mathcal{E}'$ has at least $t_s + 1$ *honest* parties, whose univariate polynomials $q_i(x)$ are pair-wise consistent and hence from Lemma 2.1, lie on a unique degree-(t_s, t_s) symmetric bivariate polynomial, say $Q^\star(x,y)$. Similarly, since the univariate polynomial $q_j(x)$ of every *honest* party P_j in $\mathcal{F}'$ is pair-wise consistent with the univariate polynomials $q_i(x)$ of the *honest* parties in $\mathcal{E}'$, it implies that the univariate polynomial $q_j(x)$ of all the *honest* parties in $\mathcal{F}'$ also lie on $Q^\star(x,y)$. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0,y)$. We show that *every honest* P_i eventually outputs $q^\star(\alpha_i)$ as its wps-share. For this it is enough to show that each honest P_i eventually gets $q_i(x) = Q^\star(x,\alpha_i)$, as P_i outputs $q_i(0)$ as its wps-share, which will be the same as $q^\star(\alpha_i)$. Consider an arbitrary *honest* P_i. There are two possible cases.
  - $P_i \in \mathcal{F}'$: In this case, P_i already has $Q^\star(x,\alpha_i)$, received from D.
  - $P_i \notin \mathcal{F}'$: In this case, $\mathcal{F}'$ has at least $n - t_a > 3t_s$ parties, of which at most t_a could be corrupt. Since $Q^\star(x,\alpha_i)$ is a t_s-degree polynomial and $t_s < |\mathcal{F}'| - 2t_a$, from Lemma A.1, it follows that by applying the OEC procedure on the common points on the polynomial $Q^\star(x,\alpha_i)$ received from the parties in $\mathcal{F}'$, party P_i will eventually obtain $Q^\star(x,\alpha_i)$.

□

Lemma 4.7. *Protocol Π_{WPS} incurs a communication of $\mathcal{O}(n^4 \log |\mathbb{F}|)$ bits from the honest parties and invokes 1 instance of Π_{BA}.*

Proof. In the protocol, D sends a t_s-degree univariate polynomial to every party. As part of the pair-wise consistency checks, each pair of parties exchange 2 field elements. In addition, an *honest* party may broadcast an $\mathtt{NOK}$ message, corresponding to every other party. As part of the $\mathtt{NOK}$ message, the honest party also broadcasts the corresponding common point on its univariate polynomial. Each such common point can be represented by $\log |\mathbb{F}|$ bits. The communication complexity now follows from the communication complexity of the protocol Π_{BC} (see Theorem 3.5). □

We next discuss the modifications needed in the protocol Π_{WPS}, if the input for D consists of L number of t_s-degree polynomials.

Π_{WPS} for L Polynomials: If D has L polynomials as input in protocol Π_{WPS}, then it embeds them into L random (t_s, t_s)-degree symmetric bivariate polynomials and distributes the univariate polynomials lying on these bivariate polynomials, to the respective parties. The parties then

perform the pair-wise consistency tests, by exchanging their supposedly common points on the bivariate polynomials. However, P_i broadcasts a *single* $\texttt{OK}(i,j)$ message for P_j, if the pair-wise consistency test is positive for *all* the L supposedly common values between P_i and P_j. On the other hand, if the test fails for *any* of the L supposedly common values, then P_i broadcasts a *single* NOK message, corresponding to the *least indexed* common value for which the test fails. Hence, instead of constructing L consistency graphs, a *single* consistency graph is constructed by each party. As a result, D finds a *single* $(\mathcal{W},\mathcal{E},\mathcal{F})$ triplet and broadcast it. Similarly, a *single* instance of Π_{BA} is used to decide whether any $(\mathcal{W},\mathcal{E},\mathcal{F})$ is accepted. Finally, if no $(\mathcal{W},\mathcal{E},\mathcal{F})$ triplet is found and broadcast, then D looks for a *single* $(n,t_a)-$star $(\mathcal{E}',\mathcal{F}')$ and broadcasts it.

To void repetition, we skip the formal details of the modified protocol and the proof of its properties, as stated in Theorem 4.8.

Theorem 4.8. *Let $n > 3t_s + t_a$ and let D has L number of t_s-degree polynomials $q^{(1)}(\cdot),\ldots,q^{(L)}(\cdot)$ as input for Π_{WPS}, where $L \geq 1$. Moreover, let $T_{\mathsf{WPS}} = 2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Then protocol Π_{WPS} achieves the following properties.*

- *If D is honest then the following hold.*
  - *t_s-correctness: In a synchronous network, each (honest) P_i outputs $\{q(\alpha_i)\}_{\ell=1,\ldots,L}$ at time T_{WPS}.*
  - *t_a-correctness: In an asynchronous network, almost-surely, each (honest) P_i eventually outputs $\{q(\alpha_i)\}_{\ell=1,\ldots,L}$.*
  - *t_s-privacy: Irrespective of the network type, the view of the adversary remains independent of the polynomials $q^{(1)}(\cdot),\ldots,q^{(L)}(\cdot)$.*
- *If D is corrupt, then either no honest party computes any output or there exist L number of t_s-degree polynomials, say $\{q^\star(\cdot)\}_{\ell=1,\ldots,L}$, such that the following hold.*
  - *t_s-Weak Commitment: In a synchronous network, at least t_s+1 honest parties P_i output wps-shares $\{q^\star(\alpha_i)\}_{\ell=1,\ldots,L}$. Moreover, if any honest P_j outputs wps-shares $s_j^{(1)},\ldots,s_j^{(L)} \in \mathbb{F}$, then $s_j = q^\star(\alpha_j)$ holds for $\ell = 1,\ldots,L$.*
  - *t_a-Strong Commitment: In an asynchronous network, almost-surely, each (honest) P_i eventually outputs $\{q^\star(\alpha_i)\}_{\ell=1,\ldots,L}$ as wps-shares.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^2 L\log|\mathbb{F}| + n^4\log|\mathbb{F}|)$ bits from the honest parties and invokes 1 instance of Π_{BA}.*

4.2 The VSS Protocol

Protocol Π_{WPS} fails to serve as a VSS because if D is *corrupt* and the network is *synchronous*, then the (honest) parties *outside* $\mathcal{W}$ may not obtain their required shares, lying on D's committed polynomials. Protocol Π_{VSS} (see Fig 4) fixes this shortcoming. For ease of understanding, we present the protocol assuming D has a single t_s-degree polynomial as input and later discuss the modifications needed when D has L such polynomials. The protocol has two "layers" of communication involved. The first layer is similar to Π_{WPS} and identifies whether the parties accepted some $(\mathcal{W},\mathcal{E},\mathcal{F})$ within a specified time-out, such that the polynomials of all honest parties in $\mathcal{W}$ lie on a single (t_s,t_s)-degree symmetric bivariate polynomial, say $Q^\star(x,y)$. If some $(\mathcal{W},\mathcal{E},\mathcal{F})$ is accepted, then the second layer of communication (which is coupled with the first layer) enables even the (honest) parties outside $\mathcal{W}$ to get their corresponding polynomials lying on $Q^\star(x,y)$.

In more detail, to perform the pair-wise consistency check of the polynomials received from D, each P_j upon receiving $q_j(x)$ from D, shares the polynomial $q_j(x)$ by invoking an instance of Π_{WPS} as a dealer. Any party P_i who computes a WPS-Share in this instance of Π_{WPS} either broadcasts an OK or NOK message for P_j, depending on whether the WPS-share lies on the polynomial which

P_i has received from D. The rest of the steps for computing $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ and accepting it remains the same. If some $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted, then any P_i *outside* $\mathcal{W}$ computes its polynomial lying on $Q^\star(x, y)$ as follows: P_i checks for a subset $\mathcal{SS}_i \subseteq \mathcal{F}$ of $t_s + 1$ parties P_j, such that P_i is able to compute its WPS-share in the Π_{WPS} instance invoked by P_j as a dealer. Such an $\mathcal{SS}_i$ is bound to exist as there are at least $t_s + 1$ *honest* parties in $\mathcal{F}$ who are always included in $\mathcal{SS}_i$. While the WPS-shares corresponding to the *honest* parties in $\mathcal{SS}_i$ will be the common points on $Q^\star(x, \alpha_i)$, the same holds even for *corrupt* parties in $\mathcal{SS}_i$. This is because in order to be included in $\mathcal{F}$, such parties are "forced" to share polynomials lying on $Q^\star(x, y)$, in their respective instances of Π_{WPS}. Now using the WPS-shares corresponding to the parties in $\mathcal{SS}_i$, party P_i will be able to compute $Q^\star(x, \alpha_i)$ and hence, its share.

Protocol $\Pi_{\mathsf{VSS}}(\mathsf{D}, q(\cdot))$

- **Phase I — Sending Polynomials**: D on having the input $q(\cdot)$, chooses a random (t_s, t_s)-degree symmetric bivariate polynomial $Q(x, y)$ such that $Q(0, y) = q(\cdot)$ and sends $q_i(x) = Q(x, \alpha_i)$ to each party $P_i \in \mathcal{P}$.
- **Phase II — Exchanging Common Values**: Each $P_i \in \mathcal{P}$, upon receiving a t_s-degree polynomial $q_i(x)$ from D, waits till the current local time becomes a multiple of Δ and then does the following.
  - Act as a dealer and invoke an instance $\Pi^{(i)}_{\mathsf{WPS}}$ of Π_{WPS} with input $q_i(x)$.
  - For $j = 1, \ldots, n$, participate in the instance $\Pi^{(j)}_{\mathsf{WPS}}$, if invoked by P_j as a dealer, and wait for time T_{WPS}.
- **Phase III — Publicly Declaring the Results of Pair-Wise Consistency Test**: Each $P_i \in \mathcal{P}$ waits till the local time becomes a multiple of Δ and then does the following.
  - If a WPS-share q_{ji} is computed during the instance $\Pi^{(j)}_{\mathsf{WPS}}$ and $q_i(x)$ has been received from D, then:
    - Broadcast $\mathtt{OK}(i, j)$, if $q_{ji} = q_i(\alpha_j)$ holds.
    - Broadcast $\mathtt{NOK}(i, j, q_i(\alpha_j))$, if $q_{ji} \neq q_i(\alpha_j)$ holds.
- **Local Computation — Constructing Consistency Graph**: Each $P_i \in \mathcal{P}$ does the following.
  - Construct a *consistency graph* G_i over $\mathcal{P}$, where the edge (P_j, P_k) is included in G_i, if $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ is received from the broadcast of P_j and P_k respectively, either through the regular-mode or fall-back mode.
- **Phase IV — Constructing $(n, t_s)-$star**: D does the following in its consistency graph G_{D} at time $2\Delta + T_{\mathsf{BC}}$.
  - Remove edges incident with P_i, if $\mathtt{NOK}(i, j, q_{ij})$ is received from the broadcast of P_i through regular-mode and $q_{ij} \neq Q(\alpha_j, \alpha_i)$.
  - Set $\mathcal{W} = \{P_i : \mathsf{deg}(P_i) \geq n - t_s\}$, where $\mathsf{deg}(P_i)$ denotes the degree of P_i in G_{D}.
  - Remove P_i from $\mathcal{W}$, if P_i is *not* incident with at least $n - t_s$ parties in $\mathcal{W}$. Repeat this step till no more parties can be removed from $\mathcal{W}$.
  - Run algorithm $\mathsf{AlgStar}$ on $G_{\mathsf{D}}[\mathcal{W}]$, where $G_{\mathsf{D}}[\mathcal{W}]$ denotes the subgraph of G_{D} induced by the vertices in $\mathcal{W}$. If an $(n, t_s)-$star, say $(\mathcal{E}, \mathcal{F})$, is obtained, then broadcast $(\mathcal{W}, \mathcal{E}, \mathcal{F})$.
- **Local Computation — Verifying and Accepting** $(\mathcal{W}, \mathcal{E}, \mathcal{F})$: Each $P_i \in \mathcal{P}$ does the following at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$.
  - If a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is received from D's broadcast through regular-mode, then *accept* it if following *were true* at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$:
    - There exist no $P_j, P_k \in \mathcal{W}$, such that $\mathtt{NOK}(j, k, q_{jk})$ and $\mathtt{NOK}(k, j, q_{kj})$ messages *were* received from the broadcast of P_j and P_k respectively through regular-mode, where $q_{jk} \neq q_{kj}$.
    - In the consistency graph G_i, $\mathsf{deg}(P_j) \geq n - t_s$ for all $P_j \in \mathcal{W}$.
    - In the consistency graph G_i, every $P_j \in \mathcal{W}$ has edges with at least $n - t_s$ parties from $\mathcal{W}$.
    - $(\mathcal{E}, \mathcal{F})$ *was* an $(n, t_s)-$star in the induced graph $G_i[\mathcal{W}]$.
    - For every $P_j, P_k \in \mathcal{W}$ where the edge (P_j, P_k) is present in G_i, the $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ messages *were* received from the broadcast of P_j and P_k respectively, through regular-mode.

- **Phase V — Deciding Whether to Go for (n, t_a)−star**: At time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$, each party P_i participates in an instance of Π_{BA} with input $b_i = 0$ if a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted, else with input $b_i = 1$ and waits for time T_{BA}.
- **Local Computation — Computing VSS-Share Through $(\mathcal{W}, \mathcal{E}, \mathcal{F})$:** If 0 is the output during the instance of Π_{BA}, then each P_i does the following.
  - If a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is not yet received, then wait till it is received from D's broadcast through fall-back mode.
  - If $P_i \in \mathcal{W}$, then output $q_i(0)$.
  - Else, initialize $\mathcal{SS}_i$ to $\emptyset$. Include $P_j \in \mathcal{F}$ to $\mathcal{SS}_i$ if a wps-share q_{ji} is computed during $\Pi^{(j)}_{\mathsf{WPS}}$. Wait till $|\mathcal{SS}_i| \geq t_s + 1$. Then interpolate $\{(\alpha_j, q_{ji})\}_{P_j \in \mathcal{SS}_i}$ to get a t_s-degree polynomial, say $q_i(x)$, and output $q_i(0)$.
- **Phase VI — Broadcasting (n, t_a)−star**: If the output during the instance of Π_{BA} is 1, then D runs $\mathsf{AlgStar}$ after every update in its consistency graph G_{D} and broadcasts $(\mathcal{E}', \mathcal{F}')$, if it finds an (n, t_a)−star $(\mathcal{E}', \mathcal{F}')$.
- **Local Computation — Computing VSS-Share Through (n, t_a)−star**: If the output during the instance of Π_{BA} is 1, then each P_i does the following.
  - Participate in any instance of Π_{BC} invoked by D for broadcasting an (n, t_a)−star *only* after time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Wait till some $(\mathcal{E}', \mathcal{F}')$ is obtained from D's broadcast (through any mode), which constitutes an (n, t_a)−star in G_i.
  - If $P_i \in \mathcal{F}'$, then output $q_i(0)$. Else, include $P_j \in \mathcal{F}'$ to $\mathcal{SS}_i$ (initialized to $\emptyset$) if a wps-share q_{ji} is computed in $\Pi^{(j)}_{\mathsf{WPS}}$. Wait till $|\mathcal{SS}_i| \geq t_s + 1$. Then interpolate $\{(\alpha_j, q_{ji})\}_{P_j \in \mathcal{SS}_i}$ to get a t_s-degree polynomial $q_i(x)$ and output $q_i(0)$.

Figure 4: best-of-both-worlds VSS protocol for a single polynomial.

We next proceed to prove the properties of the protocol Π_{VSS}. We first start by showing that if D is *honest*, then the view of the adversary remains independent of dealer's polynomial.

Lemma 4.9 (t_s*-Privacy*)**.** *In protocol Π_{VSS}, if D is honest, then irrespective of the network type, the view of the adversary remains independent of $q(\cdot)$.*

Proof. Let D be *honest*. We consider the worst case scenario when adversary controls up to t_s parties. We claim that throughout the protocol, the adversary learns at most t_s univariate polynomials lying on $Q(x, y)$. Since $Q(x, y)$ is a random (t_s, t_s)-degree symmetric-bivariate polynomial, it then follows from Lemma 2.2, that the view of the adversary will be independent of $q(\cdot)$. We next proceed to prove the claim.

Corresponding to every *corrupt* P_i, the adversary learns $Q(x, \alpha_i)$. Corresponding to every *honest* P_i, the adversary learns t_s number of $q_i(\alpha_j)$ values through pair-wise consistency tests, as these values are computed as wps-shares, during the instance $\Pi^{(i)}_{\mathsf{WPS}}$. However, these values are already included in the view of the adversary (through the univariate polynomials under adversary's control). Additionally, from the t_s*-privacy* property of Π_{WPS}, the view of the adversary remains independent of $q_i(x)$ during $\Pi^{(i)}_{\mathsf{WPS}}$, if P_i is *honest*. Hence no additional information about the polynomials of the honest parties is revealed during the pair-wise consistency checks. Furthermore, no *honest* P_i ever broadcasts $\mathtt{NOK}(i, j, q_{ij})$ corresponding to any *honest* P_j, since the pair-wise consistency check will always pass for every pair of *honest* parties. □

We next prove the correctness property in a *synchronous* network.

Lemma 4.10 (t_s*-Correctness*)**.** *In protocol Π_{VSS}, if D is honest and network is synchronous, then each honest P_i outputs $q(\alpha_i)$ within time $T_{\mathsf{VSS}} = \Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$.*

Proof. Let D be *honest* and network be *synchronous* with up to t_s corruptions. During phase I, all honest parties receive $q_i(x) = Q(x, \alpha_i)$ from D within time Δ. Consequently during phase II,

each *honest* P_i invokes the instance $\Pi^{(i)}_{\mathsf{WPS}}$ with input $q_i(x)$. From the t_s-*correctness* of Π_{WPS} in the *synchronous* network, corresponding to each *honest* P_j, every *honest* P_i computes the wps-share $q_{ji} = q_j(\alpha_i)$, at time $\Delta + T_{\mathsf{WPS}}$. Consequently, during phase III, every *honest* party broadcasts an $\mathtt{OK}$ message for every other *honest* party, since $q_{ji} = q_i(\alpha_j)$ holds, for every pair of honest parties P_i, P_j. From the t_s-*validity* property of Π_{BC} in the *synchronous* network, these $\mathtt{OK}$ messages are received by every honest party through regular-mode at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$. Hence, there will be an edge between every pair of *honest* parties in the consistency graph of every honest party. Moreover, if D receives an *incorrect* $\mathtt{NOK}(i, j, q_{ij})$ message from the broadcast of any *corrupt* P_j through regular-mode at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$ where $q_{ij} \neq Q(\alpha_j, \alpha_i)$, then D removes all the edges incident with P_i in D's consistency graph G_{D}. D then computes the set $\mathcal{W}$ and all *honest* parties will be present in $\mathcal{W}$. Moreover, the honest parties will form a clique of size at least $n - t_s$ in the subgraph $G_{\mathsf{D}}[\mathcal{W}]$ at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$. Hence, D will find an $(n, t_s)-$star $(\mathcal{E}, \mathcal{F})$ in $G_{\mathsf{D}}[\mathcal{W}]$ and broadcast $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ during phase IV. By the t_s-*validity* of Π_{BC} in the *synchronous* network, all honest parties will receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ through regular-mode at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Moreover, all honest parties will accept *accept* $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ and participate with input 0 in the instance of Π_{BA}. By the t_s-*validity* and t_s-*guaranteed liveness* of Π_{BA} in the *synchronous* network, the output of the Π_{BA} instance will be 0 for every honest party at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Now consider an arbitrary *honest* party P_i. Since $P_i \in \mathcal{W}$, P_i outputs $s_i = q_i(0) = Q(0, \alpha_i) = q(\alpha_i)$. □

We next prove the correctness property in the asynchronous network.

Lemma 4.11 (t_a-*Correctness*)**.** *In protocol* Π_{VSS}*, if* D *is honest and network is asynchronous, then almost-surely, each honest* P_i *eventually outputs* $q(\alpha_i)$*.*

Proof. Let D be *honest* and network be *asynchronous* with up to t_a corruptions. We first note that every *honest* P_i eventually broadcasts $\mathtt{OK}(i, j)$ message, corresponding to every *honest* P_j. This is because both P_i and P_j eventually receive $q_i(x) = Q(x, \alpha_i)$ and $q_j(x) = Q(x, \alpha_j)$ respectively from D. Moreover, P_j participates with input $q_j(\cdot)$ during $\Pi^{(j)}_{\mathsf{WPS}}$. And from the t_a-*correctness* of Π_{WPS} in the *asynchronous* network, party P_i eventually computes the wps-share $q_{ji} = q_j(\alpha_i)$ during $\Pi^{(j)}_{\mathsf{WPS}}$. Moreover, $q_{ji} = q_{ij}$ holds. Note that every honest party participates with some input in the instance of Π_{BA} at local time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Hence, from the t_a-*almost-surely liveness* and t_a-*consistency* properties of Π_{BA} in the asynchronous network, almost-surely, all honest parties eventually compute a common output during the instance of Π_{BA}. Now there are two possible cases:

- **The output of Π_{BA} is** 0: From the t_a-*validity* of Π_{BA} in the *asynchronous* network, this means that at least one *honest* party, say P_h, participated with input 0 during the instance of Π_{BA}. This implies that P_h has received $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepted it. Hence, by the t_a-*weak validity* and t_a-*fallback validity* of Π_{BC} in the *asynchronous* network, all honest parties will eventually receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D. We claim that *every honest* P_i will eventually get $Q(x, \alpha_i)$. This will imply that eventually every *honest* P_i outputs $s_i = Q(0, \alpha_i) = q(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are two possible cases.
  - $P_i \in \mathcal{W}$: In this case, P_i already has $Q(x, \alpha_i)$, received from D.
  - $P_i \notin \mathcal{W}$: In this case, we first note that there will be at least $t_s + 1$ parties, who are eventually included in $\mathcal{SS}_i$. This follows from the fact that there are at least $t_s + 1$ *honest* parties P_j in $\mathcal{F}$. And corresponding to every *honest* $P_j \in \mathcal{F}$, party P_i will eventually compute the wps-share q_{ji} in the instance $\Pi^{(j)}_{\mathsf{WPS}}$, which follows from the t_a-*correctness* of Π_{WPS} in the *asynchronous* network. We next claim that corresponding to *every* $P_j \in \mathcal{SS}_i$, the value q_{ji} computed by P_i is the *same* as $Q(\alpha_j, \alpha_i)$.

The claim is obviously true for every *honest* $P_j \in \mathcal{SS}_i$, so consider a *corrupt* $P_j \in \mathcal{SS}_i$. We first note that the input polynomial $q_j(x)$ of P_j during $\Pi^{(j)}_{\mathsf{WPS}}$ is the *same* as $Q(x, \alpha_j)$. This is because $P_j \in \mathcal{W}$, since $\mathcal{F} \subseteq \mathcal{W}$. And hence P_j has edges with at least $n - t_s$ parties in $\mathcal{W}$ and hence with at least $n - t_s - t_a > t_s$ *honest* parties from $\mathcal{W}$ in $G_{\mathsf{D}}[\mathcal{W}]$. Let $\mathcal{H}$ be the set of *honest* parties in $\mathcal{W}$ with which P_j has edges in $G_{\mathsf{D}}[\mathcal{W}]$. This implies that *every* $P_k \in \mathcal{H}$ has broadcast $\mathtt{OK}(k, j)$ message after verifying that $q_{jk} = q_k(\alpha_j)$ holds, where the polynomial $q_k(x)$ held by P_k is the same as $Q(x, \alpha_k)$ and where the WPS-share q_{jk} computed by P_k during $\Pi^{(j)}_{\mathsf{WPS}}$ is the *same* as $q_j(\alpha_k)$; the last property follows from the t_a-*strong commitment* of Π_{WPS} in the *synchronous* network. Since $|\mathcal{H}| > t_s$, it implies that at least $t_s + 1$ *honest* parties P_k have verified that $q_j(\alpha_k) = Q(\alpha_j, \alpha_k)$ holds. This further implies that $q_j(x) = Q(x, \alpha_j)$, since two different t_s-degree polynomials can have at most t_s common values. Since P_i has computed the wps-share q_{ji} during $\Pi^{(j)}_{\mathsf{WPS}}$, from the t_a-*strong commitment* of Π_{WPS} in *synchronous* network, it follows that $q_{ji} = q_j(\alpha_i) = Q(\alpha_i, \alpha_j) = Q(\alpha_j, \alpha_i)$. The last equality follows since each $Q(x, y)$ is a symmetric bivariate polynomial.

- **The output of Π_{BA} is** 1: As mentioned earlier, since D is *honest*, every pair of honest parties eventually broadcast $\mathtt{OK}$ messages corresponding to each other, as the pair-wise consistency check between them will be eventually positive. From the t_a-*weak validity* and t_a-*fallback validity* of Π_{BC} in the *asynchronous* network, these messages are eventually delivered to every honest party. Also from the t_a-*weak consistency* and t_a-*fallback consistency* of Π_{BC} in the *asynchronous* network, any $\mathtt{OK}$ message which is received by D, will be eventually received by every other honest party as well. As there will be at least $n - t_a$ honest parties, a clique of size at least $n - t_a$ will eventually form in the consistency graph of every honest party. Hence D will eventually find an $(n, t_a)-$star, say $(\mathcal{E}', \mathcal{F}')$, in its consistency graph and broadcast it. From the t_a-*weak validity* and t_a-*fallback validity* of Π_{BC} in the *asynchronous* network, $(\mathcal{E}', \mathcal{F}')$ will be eventually received by every honest party. Moreover, $(\mathcal{E}', \mathcal{F}')$ will be eventually an $(n, t_a)-$star in every honest party's consistency graph. We now claim that *every honest* P_i will eventually get $Q(x, \alpha_i)$. This will imply that eventually every *honest* P_i outputs $s_i = Q(0, \alpha_i) = q(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are two possible cases.
  - $P_i \in \mathcal{F}'$: In this case, P_i already has $Q(x, \alpha_i)$, received from D.
  - $P_i \notin \mathcal{F}'$: In this case, we note that there will be will be at least $t_s + 1$ parties, who are eventually included in $\mathcal{SS}_i$, such that corresponding to *every* $P_j \in \mathcal{SS}_i$, the value q_{ji} computed by P_i during $\Pi^{(j)}_{\mathsf{WPS}}$ is the same as $Q(\alpha_j, \alpha_i)$. The proof for this will be similar as for the case when $P_i \notin \mathcal{W}$ and the output of Π_{BA} is 0 and so we skip the proof.

□

Before we proceed to prove the *strong commitment* property in the *synchronous* network, we prove a helping lemma.

Lemma 4.12. *Let D be corrupt and network be synchronous. If any honest party receives a $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$, then all the following hold:*

- *All honest parties in $\mathcal{W}$ have received their respective t_s-degree univariate polynomials from D within time Δ.*
- *The univariate polynomials $q_i(x)$ of all honest parties P_i in $\mathcal{W}$ lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$.*
- *At time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$, every honest party accepts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$.*

Proof. Let D be *corrupt* and network be *synchronous* with up to t_s corruptions. As per the lemma condition, let P_h be an *honest* party, who receives some $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepts it at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. From the protocol steps, it then follows that the following must be true for P_h at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$:

- There does not exist any $P_j, P_k \in \mathcal{W}$, such that $\mathtt{NOK}(j, k, q_{jk})$ and $\mathtt{NOK}(k, j, q_{kj})$ messages *were* received from the broadcast of P_j and P_k respectively through regular-mode, such that $q_{jk} \neq q_{kj}$.
- In P_h's consistency graph G_h, $\mathsf{deg}(P_j) \geq n - t_s$ for all $P_j \in \mathcal{W}$ and P_j has edges with at least $n - t_s$ parties from $\mathcal{W}$.
- $(\mathcal{E}, \mathcal{F})$ *was* an $(n, t_s)-$star in the induced subgraph $G_h[\mathcal{W}]$, such that for every $P_j, P_k \in \mathcal{W}$ where the edge (P_j, P_k) is present in G_h, the $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ messages *were* received from the broadcast of P_j and P_k respectively through regular-mode.

We prove the first part of the lemma through a contradiction. So let $P_j \in \mathcal{W}$ be an *honest* party, who receives its t_s-degree univariate polynomial from D, say $q_j(x)$, at time $\Delta + \delta$, where $\delta > 0$. Moreover, let $P_k \in \mathcal{W}$ be an *honest* party such that P_j has an edge with P_k (note that P_j has edges with at least $n - 2t_s > t_s + t_a$ *honest* parties in $\mathcal{W}$). As stated above, at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$, party P_h has received the message $\mathtt{OK}(k, j)$ from the broadcast of P_k through regular-mode. From the protocol steps, P_j waits till its local time becomes a multiple of Δ, before it participates with input $q_j(\cdot)$ in the instance $\Pi^{(j)}_{\mathsf{WPS}}$. Hence, P_j must have invoked $\Pi^{(j)}_{\mathsf{WPS}}$ at time $c \cdot \Delta$, where $c \geq 2$. Since the network is *synchronous*, from the t_s-*correctness* of Π_{WPS} in the *synchronous* network, party P_k will compute its wps-share q_{jk} during $\Pi^{(j)}_{\mathsf{WPS}}$ at time $c \cdot \Delta + T_{\mathsf{WPS}}$. Hence the result of the pair-wise consistency test with P_j will be available to P_k at time $c \cdot \Delta + T_{\mathsf{WPS}}$. As a result, P_k starts broadcasting $\mathtt{OK}(k, j)$ message only at time $c \cdot \Delta + T_{\mathsf{WPS}}$. Since P_k is *honest*, from the t_s-*validity* property of Π_{BC} in the *synchronous* network, it will take *exactly* T_{BC} time for the message $\mathtt{OK}(k, j)$ to be received through regular-mode, once it is broadcast. This implies that P_h will receive the message $\mathtt{OK}(k, j)$ at time $c \cdot \Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$, where $c \geq 2$. However, this is a contradiction, since the $\mathtt{OK}(k, j)$ message has been received by P_h at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$.

To prove the second part of the lemma, we will show that the univariate polynomials $q_i(x)$ of all the *honest* parties $P_i \in \mathcal{W}$ are pair-wise consistent. Since there will be at least $n - 2t_s > t_s$ *honest* parties in $\mathcal{W}$, from Lemma 2.2 this will imply that all these polynomials lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. So consider an arbitrary pair of *honest* parties $P_j, P_k \in \mathcal{W}$. From the first part of the claim, both P_j and P_k must have received their respective univariate polynomials $q_j(x)$ and $q_k(x)$ by time Δ. Moreover, from the t_s-*correctness* property of Π_{WPS} in the *synchronous* network, P_j and P_k will compute the wps-shares $q_{kj} = q_k(\alpha_j)$ and $q_{jk} = q_j(\alpha_k)$ at time $\Delta + T_{\mathsf{WPS}}$ during $\Pi^{(k)}_{\mathsf{WPS}}$ and $\Pi^{(j)}_{\mathsf{WPS}}$ respectively. Since P_j and P_k are honest, if $q_{kj} \neq q_{jk}$, they would broadcast $\mathtt{NOK}(j, k, q_{jk})$ and $\mathtt{NOK}(k, j, q_{kj})$ messages respectively at time $\Delta + T_{\mathsf{WPS}}$. From the t_s-*validity* property of Π_{BC} in the *synchronous* network, P_h will receive these messages through regular-mode at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$. Consequently, P_h *will not* accept $(\mathcal{W}, \mathcal{E}, \mathcal{F})$, which is a contradiction.

To prove the third part of the lemma, we note that since P_h receives $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$, it implies that D must have started broadcasting $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ latest at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$. This is because it takes T_{BC} time for the regular-mode of Π_{BC} to generate an output. From the t_s-*consistency* property of Π_{BC} in the *synchronous* network, it follows that every honest party will also receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Similarly, since at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$, party P_h has received the $\mathtt{OK}(j, k)$ and $\mathtt{OK}(k, j)$ messages through regular-mode from the broadcast of every $P_j, P_k \in \mathcal{W}$ where (P_j, P_k) is an edge in P_h's consistency graph, it follows that these messages started getting

broadcast latest at time $\Delta + T_{\mathsf{WPS}}$. From the t_s-*validity* and t_s-*consistency* properties of Π_{BC} in the *synchronous* network, it follows that every honest party receives these broadcast messages through regular-mode at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$. Hence $(\mathcal{E}, \mathcal{F})$ will constitute an $(n, t_s)-$star in the induced subgraph $G_i[\mathcal{W}]$ of every honest party P_i's consistency-graph at time $\Delta + T_{\mathsf{WPS}} + T_{\mathsf{BC}}$ and consequently, every honest party accepts $(\mathcal{W}, \mathcal{E}, \mathcal{F})$. □

We next prove the strong commitment property in the synchronous network.

Lemma 4.13 (t_s-*Strong Commitment*)**.** *In protocol* Π_{VSS}*, if* D *is corrupt and network is synchronous, then either no honest party computes any output or there exist a* t_s*-degree polynomial, say* $q^\star(\cdot)$*, such that each honest* P_i *eventually outputs* $q^\star(\alpha_i)$*, where the following hold.*

- *If any honest* P_i *computes its output at time* $T_{\mathsf{VSS}} = \Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$*, then every honest party obtains its output at time* T_{VSS}*.*
- *If any honest* P_i *computes its output at time* T *where* $T > T_{\mathsf{VSS}}$*, then every honest party computes its output by time* $T + 2\Delta$*.*

Proof. Let D be *corrupt* and network be *synchronous* with up to t_s corruptions. If no honest party computes any output, then the lemma holds trivially. So consider the case when some *honest* party computes an output. Now, there are two possible cases.

- **At least one honest party, say** P_h**, has received some** $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ **from the broadcast of** D **through regular-mode and accepted** $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ **at time** $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$: In this case, from Lemma 4.12, the polynomials $q_i(x)$ of all *honest* parties in $\mathcal{W}$ are guaranteed to lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. As per the protocol steps, P_h has also verified that $\mathcal{F} \subseteq \mathcal{W}$, by checking that $(\mathcal{E}, \mathcal{F})$ constitutes an $(n, t_s)-$star in the induced subgraph $G_h[\mathcal{W}]$. Hence the polynomials $q_i(x)$ of all *honest* parties in $\mathcal{F}$ also lie on $Q^\star(x, y)$. Moreover, from Lemma 4.12, all honest parties accept $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Hence, every honest party participates in the instance of Π_{BA} with input 0. Consequently, by the t_s-*validity* and t_s-*guaranteed liveness* properties of Π_{BA} in the *synchronous* network, all honest parties compute the output 0 during the instance of Π_{BA} at time $T_{\mathsf{VSS}} = \Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Let $q^\star(\cdot) = Q^\star(0, y)$ and consider an arbitrary *honest* party P_i. We wish to show that P_i has $q_i(x) = Q^\star(x, \alpha_i)$ at time T_{VSS}, which will imply that P_i outputs $s_i = q_i(0)$ at time T_{VSS}, which will be the same as $q^\star(\alpha_i)$. For this, we consider the following two possible cases.
  - $P_i \in \mathcal{W}$: In this case, P_i has already received $q_i(x)$ from D within time Δ. This follows from Lemma 4.12.
  - $P_i \notin \mathcal{W}$: In this case, we claim that at time T_{VSS}, there will be will be at least t_s+1 parties from $\mathcal{F}$, who are included in $\mathcal{SS}_i$, such that corresponding to *every* $P_j \in \mathcal{SS}_i$, party P_i will have the value q_{ji}, which will be the same as $Q^\star(\alpha_j, \alpha_i)$. Namely, there are at least $t_s + 1$ *honest* parties in $\mathcal{F}$, who will be included in $\mathcal{SS}_i$ and the claim will be trivially true for those parties, due to the t_s-*correctness* property of Π_{WPS} in the synchronous network. On the other hand, if any *corrupt* $P_j \in \mathcal{F}$ is included in $\mathcal{SS}_i$, then the input polynomial of P_j during $\Pi^{(j)}_{\mathsf{WPS}}$ will be pair-wise consistent with the polynomials of at least $t_s + 1$ *honest* parties in $\mathcal{W}$ and hence will be the same as $Q^\star(x, \alpha_j)$. Moreover, from the t_s-*weak commitment* of Π_{WPS} in the *synchronous* network, the WPS-share q_{ji} computed by P_i during $\Pi^{(j)}_{\mathsf{WPS}}$ will be the same as $Q^\star(\alpha_i, \alpha_j)$, which will be the same as $Q^\star(\alpha_j, \alpha_i)$, since $Q^\star(x, y)$ is a symmetric bivariate polynomial. Hence, P_i will interpolate $q_i(x)$.
- **No honest party has received any** $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ **from the broadcast of** D **through regular-mode and accepted** $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ **at time** $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$: This implies that

all honest parties participate in the instance of Π_{BA} with input 1. Hence, by the t_s-*validity* and t_s-*guaranteed liveness* of Π_{BA} in the *synchronous* network, all honest parties obtain the output 1 during the instance of Π_{BA} at time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Let P_h be the *first honest* party, who computes an output. This means that P_h has received a pair $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D, such that $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph. Let T be the time when $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph. This implies that at time T, party P_h has $(\mathcal{E}', \mathcal{F}')$ from D's broadcast and also all the $\mathtt{OK}(\star, \star)$ messages, from the broadcast of respective parties in $\mathcal{E}'$ and $\mathcal{F}'$. From the protocol steps, $T > T_{\mathsf{VSS}}$, since the honest parties participate in the instance of Π_{BC} through which D has broadcast $(\mathcal{E}', \mathcal{F}')$ *only* after time T_{VSS}. By the t_s-*consistency* and t_s-*fallback consistency* properties of Π_{BC} in the *synchronous* network, *all* honest parties will receive $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D by time $T + 2\Delta$. Moreover, $(\mathcal{E}', \mathcal{F}')$ will constitute an $(n, t_a)-$star in every honest party's consistency graph by time $T + 2\Delta$. This is because all the $\mathtt{OK}$ messages which are received by P_h from the broadcast of various parties in $\mathcal{E}'$ and $\mathcal{F}'$ are guaranteed to be received by every honest party by time $T + 2\Delta$. Since $|\mathcal{E}'| \geq n - 2t_a > 2t_s + (t_s - t_a) > 2t_s$, it follows that $\mathcal{E}'$ has at least $t_s + 1$ *honest* parties. Moreover, the univariate polynomials $(q_j(x), q_k(x))$ of every pair of *honest* parties $P_j, P_k \in \mathcal{E}'$ will be pair-wise consistent and hence lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Similarly, the univariate polynomial $q_i(x)$ of every *honest* party P_i in $\mathcal{F}'$ is pair-wise consistent with the univariate polynomials $q_j(x)$ of *all* the *honest* parties in $\mathcal{E}'$ and hence lie on $Q^\star(x, y)$ as well. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. We show that *every honest* P_i outputs $q^\star(\alpha_i)$, by time $T + 2\Delta$. For this it is enough to show that each honest P_i has $q_i(x) = Q^\star(x, \alpha_i)$ by time $T + 2\Delta$, as P_i outputs $q_i(0)$, which will be the same as $q^\star(\alpha_i)$. Consider an arbitrary *honest* party P_i. There are two possible cases.

- $P_i \in \mathcal{F}'$: In this case, P_i has already received $Q^\star(x, \alpha_i)$ from D, well before time $T + 2\Delta$.
- $P_i \notin \mathcal{F}'$: In this case, we claim that by time $T + 2\Delta$, there will be will be at least $t_s + 1$ parties from $\mathcal{F}'$, who are included in $\mathcal{SS}_i$, such that corresponding to *every* $P_j \in \mathcal{SS}_i$, party P_i will have the value q_{ji}, which will be the same as $Q^\star(\alpha_j, \alpha_i)$. The proof for this is very similar to the previous case when $P_i \notin \mathcal{W}$ and the output of Π_{BA} is 0. Namely every *honest* $P_j \in \mathcal{F}'$ will be included in $\mathcal{SS}_i$. This is because P_j starts broadcasting $\mathtt{OK}$ messages for other parties in $\mathcal{E}'$ only after invoking instance $\Pi^{(j)}_{\mathsf{WPS}}$ with input $q_j(x)$. Hence, by time $T + 2\Delta$, the WPS-share q_{ji} from the instance $\Pi^{(j)}_{\mathsf{WPS}}$ will be available with P_i. On the other hand, if a *corrupt* $P_j \in \mathcal{F}'$ is included in $\mathcal{SS}_i$, then also the claim holds (the proof for this is similar to the proof of the t_a-*correctness* property in the *asynchronous* network in Lemma 4.11).

□

We finally prove the strong commitment property in an asynchronous network.

Lemma 4.14 (t_a-*Strong Commitment*)**.** *In protocol* Π_{VSS}*, if* D *is corrupt and network is asynchronous, then either no honest party computes any output or there exist some* t_s*-degree polynomial, say* $q^\star(\cdot)$*, such that almost-surely, every honest* P_i *eventually outputs* $q^\star(\alpha_i)$*.*

Proof. Let D be *corrupt* and the network be *asynchronous* with up to t_a corruptions. If no honest party computes any output, then the lemma holds trivially. So, consider the case when some honest party computes an output. We note that every *honest* party participates with some input in the instance of Π_{BA} at local time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Hence, from the t_a-*almost-surely liveness* and t_a-*consistency* properties of Π_{BA} in the asynchronous network, almost-surely, all honest parties

eventually compute a common output during the instance of Π_{BA}. Now there are two possible cases:

- **The output of Π_{BA} is 0**: From the t_a-*validity* of Π_{BA} in the *asynchronous* network, it follows that at least one honest party, say P_h, participated with input 0 during the instance of Π_{BA}. This means that P_h has received some $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D through regular-mode and accepted it at local time $\Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}}$. Hence, by the t_a-*weak consistency* and t_a-*fallback consistency* of Π_{BC} in the *asynchronous* network, all honest parties will eventually receive $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ from the broadcast of D. There will be at least $n - 2t_s - t_a > t_s$ *honest* parties P_i in $\mathcal{E}$, whose univariate polynomials $q_i(x)$ are pair-wise consistent and hence lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Similarly, the univariate polynomial $q_i(x)$ of every *honest* $P_i \in \mathcal{F}$ will be pair-wise consistent with the univariate polynomials $q_j(x)$ of *all* the *honest* parties P_j in $\mathcal{E}$ and hence will lie on $Q^\star(x, y)$ as well. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. We claim that *every honest* P_i will eventually have $Q^\star(x, \alpha_i)$. This will imply that eventually every *honest* P_i outputs $s_i = Q^\star(0, \alpha_i) = q^\star(\alpha_i)$. To prove the claim, consider an arbitrary *honest* party P_i. There are three possible cases.
  - $P_i \in \mathcal{W}$ *and* $P_i \in \mathcal{F}$: In this case, P_i has received the polynomials $q_i(x)$ from D and since $P_i \in \mathcal{F}$, the condition $q_i(x) = Q^\star(x, \alpha_i)$ holds.
  - $P_i \in \mathcal{W}$ *and* $P_i \notin \mathcal{F}$: In this case, P_i has received the polynomial $q_i(x)$ from D. Since $|\mathcal{W}| \geq n - t_s$ and $|\mathcal{F}| \geq n - t_s$, $|\mathcal{W} \cap \mathcal{F}| \geq n - 2t_s > t_s + t_a$. From the protocol steps, the polynomial $q_i(x)$ is pair-wise consistent with the polynomials $q_j(x)$ at least $n - t_s$ parties $P_j \in \mathcal{W}$, since P_i has edges with at least $n - t_s$ parties P_j within $\mathcal{W}$. Now among these $n - t_s$ parties, at least $n - 2t_s$ parties will be from $\mathcal{F}$, of which at least $n - 2t_s - t_a > t_s$ parties will be *honest*. Hence, $q_i(x)$ is pair-wise consistent with the $q_j(x)$ polynomials of at least $t_s + 1$ *honest* parties $P_j \in \mathcal{F}$. Now since the $q_j(x)$ polynomials of all the *honest* parties in $\mathcal{F}$ lie on $Q^\star(x, y)$, it implies that $q_i(x) = Q^\star(x, \alpha_i)$ holds.
  - $P_i \notin \mathcal{W}$: In this case, similar to the proof of Lemma 4.11, one can show that P_i eventually includes at least $t_s + 1$ parties from $\mathcal{F}$ in $\mathcal{SS}_i$. And the value computed by P_i corresponding to *any* $P_j \in \mathcal{SS}_i$ will be the same as $Q^\star(\alpha_j, \alpha_i)$. Hence, P_i will eventually interpolate $Q^\star(x, \alpha_i)$.
- **The output of Π_{BA} is 1**: Let P_h be the *first honest* party, who computes an output in Π_{VSS}. This means that P_h has received some $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D, such that $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph. By the t_a-*weak consistency* and t_a-*fallback consistency* properties of Π_{BC} in the *asynchronous* network, *all* honest parties eventually receive $(\mathcal{E}', \mathcal{F}')$ from the broadcast of D. Moreover, since the consistency graphs are constructed based on the broadcast $\mathtt{OK}$ messages and since $(\mathcal{E}', \mathcal{F}')$ constitutes an $(n, t_a)-$star in P_h's consistency graph, from the t_a-*weak validity,* t_a-*fallback validity,* t_a-*weak consistency and* t_a-*fallback consistency* properties of Π_{BC} in the *asynchronous* network, the pair $(\mathcal{E}', \mathcal{F}')$ will eventually constitute an $(n, t_a)-$star in every honest party's consistency graph, as the corresponding $\mathtt{OK}$ messages are eventually received by every honest party. Since $|\mathcal{E}'| \geq n - 2t_a > 2t_s + (t_s - t_a) > 2t_s$, it follows that $\mathcal{E}'$ has at least $t_s + 1$ *honest* parties P_i, whose univariate polynomials $q_i(x)$ are pair-wise consistent and hence lie on a unique (t_s, t_s)-degree symmetric bivariate polynomial, say $Q^\star(x, y)$. Similarly, since the univariate polynomials $q_j(x)$ of every *honest* party P_j in $\mathcal{F}'$ is pair-wise consistent with the univariate polynomials $q_i(x)$ of all the *honest* parties P_i in $\mathcal{E}'$, it implies that the polynomials $q_j(x)$ of all the *honest* parties P_j in $\mathcal{F}'$ also lie on $Q^\star(x, y)$ as well. Let $q^\star(\cdot) \stackrel{def}{=} Q^\star(0, y)$. We show that *every honest* P_i eventually outputs $q^\star(\alpha_i)$. For this it is enough to show that each honest P_i eventually gets $q_i(x) = Q^\star(x, \alpha_i)$, as P_i outputs $q_i(0)$, which will be the same as $q^\star(\alpha_i)$. Consider an

arbitrary *honest* party P_i. There are two possible cases.

- $P_i \in \mathcal{F}'$: In this case, P_i already has received $Q^\star(x, \alpha_i)$ from D.
- $P_i \notin \mathcal{F}'$: Again in this case, one can show that P_i eventually includes at least t_s+1 parties from $\mathcal{F}'$ in $\mathcal{SS}_i$. And the value computed by P_i corresponding to *any* $P_j \in \mathcal{SS}_i$ will be the same as $Q^\star(\alpha_j, \alpha_i)$. Hence, P_i will eventually interpolate $Q^\star(x, \alpha_i)$.

□

Lemma 4.15. *Protocol* Π_{VSS} *incurs a communication of* $\mathcal{O}(n^5 \log |\mathbb{F}|)$ *bits and invokes* $n+1$ *instance of* Π_{BA}.

Proof. The proof follows from Lemma 4.7 and the fact that each party acts as a dealer and invokes an instance of Π_{WPS} with a t_s-degree polynomial. Hence, the total communication cost due to the instances of Π_{WPS} in Π_{VSS} will be $\mathcal{O}(n \cdot n^4 \log |\mathbb{F}|) = \mathcal{O}(n^5 \log |\mathbb{F}|)$ bits, along with n instances of Π_{BA}. Additionally, there is an instance of Π_{BA} invoked in Π_{VSS} to agree on whether some $(\mathcal{W}, \mathcal{E}, \mathcal{F})$ is accepted. □

We next discuss the modifications needed in the protocol Π_{VSS}, if the input for D consists of L number of t_s-degree polynomials.

Protocol Π_{VSS} for L Polynomials: If D has L polynomials as input for Π_{VSS}, then we make similar modifications as done for Π_{WPS} handling L polynomials, with each party broadcasting a *single* `OK`/`NOK` message for every other party. To void repetition, we skip the formal details of the modified protocol and the proof of its properties, as stated in Theorem 4.16.

Theorem 4.16. *Let* $n > 3t_s + t_a$ *and let* D *has* L *number of* t_s*-degree polynomials* $q^{(1)}(\cdot), \ldots, q^{(L)}(\cdot)$ *as input for* Π_{VSS} *where* $L \geq 1$. *Moreover, let* $T_{\mathsf{VSS}} = \Delta + T_{\mathsf{WPS}} + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. *Then protocol* Π_{VSS} *achieves the following properties.*

- *If* D *is honest, then the following hold.*
  - t_s*-correctness: In a synchronous network, each (honest)* P_i *outputs* $\{q(\alpha_i)\}_{\ell=1,\ldots,L}$ *at time* T_{VSS}.
  - t_a*-correctness: In an asynchronous network, almost-surely, each (honest)* P_i *eventually outputs* $\{q(\alpha_i)\}_{\ell=1,\ldots,L}$.
  - t_s*-privacy: Irrespective of the network type, the view of the adversary remains independent of the polynomials* $q^{(1)}(\cdot), \ldots, q^{(L)}(\cdot)$.
- *If* D *is corrupt, then either no honest party computes any output or there exist* t_s*-degree polynomials* $\{q^{\star(\ell)}(\cdot)\}_{\ell=1,\ldots,L}$, *such that the following hold.*
  - t_s*-strong commitment: every honest* P_i *eventually outputs* $\{q^{\star(\ell)}(\alpha_i)\}_{\ell=1,\ldots,L}$, *such that one of the following hold.*
    - *If any honest* P_i *computes its output at time* T_{VSS}, *then all honest parties compute their output at time* T_{VSS}.
    - *If any honest* P_i *computes its output at time* T *where* $T > T_{\mathsf{VSS}}$, *then every honest party computes its output by time* $T + 2\Delta$.
- *Irrespective of the network type, the protocol incurs a communication of* $\mathcal{O}(n^3 L \log |\mathbb{F}| + n^5 \log |\mathbb{F}|)$ *bits from the honest parties and invokes* $n+1$ *instances of* Π_{BA}.

5 Agreement on a Common Subset (ACS)

In this section, we present a best-of-both-worlds protocol for agreement on a common subset, which will be later used in our preprocessing phase protocol, as well as in our circuit-evaluation protocol.

In the protocol, each party $P_i \in \mathcal{P}$ has L number of t_s-degree polynomials as input for an instance of Π_{VSS}, which P_i is supposed to invoke as a dealer.[11] As *corrupt* parties may not invoke their instances of Π_{VSS} as dealer, the parties may obtain points lying on the polynomials of only $n - t_s$ parties (even in a *synchronous* network). However, in an *asynchronous* network, different parties may obtain points on the polynomials of different subsets of $n - t_s$ parties. The ACS protocol allows the parties to agree on a *common subset* $\mathcal{CS}$ of at least $n - t_s$ parties, such that all (honest) parties are guaranteed to receive points lying on the polynomials of the parties in $\mathcal{CS}$. Additionally, the protocol guarantees that in a *synchronous* network, all *honest* parties are present in $\mathcal{CS}$. Looking ahead, this property will be very crucial when the ACS protocol is used during circuit-evaluation, as it will ensure that in a *synchronous* network, the inputs of *all* honest parties are considered for the circuit-evaluation.

The ACS protocol is presented in Fig 5, where for simplicity we assume that $L = 1$. Later, we discuss the modifications required for $L > 1$. In the protocol, each party acts as a dealer and invokes an instance of Π_{VSS} to verifiably distribute points on its polynomial. If the network is *synchronous*, then after time T_{VSS}, all honest parties would have received points corresponding to the polynomials of the *honest* dealers. Hence after (local) time T_{VSS}, the parties *locally* check for the instances of Π_{VSS} in which they have received an output. Based on this, the parties start participating in n instances of Π_{BA}, where the j^{th} instance is used to decide whether P_j should be included in $\mathcal{CS}$. The input criteria for these Π_{BA} instances is the following: if a party has received an output in the Π_{VSS} instance with P_j as the dealer, then the party starts participating with input 1 in the corresponding Π_{BA} instance. Now once 1 is obtained as the output from $n - t_s$ instances of Π_{BA}, then the parties start participating with input 0 in any of the remaining Π_{BA} instances for which the parties may have not provided any input yet. Finally, once an output is obtained from all the n instances of Π_{BA}, party P_j is included in $\mathcal{CS}$ if and only if the output of the corresponding Π_{BA} instance is 1. Since the parties wait for time T_{VSS} *before* starting the Π_{BA} instances, it is ensured that all *honest* dealers are included in $\mathcal{CS}$ in a *synchronous* network.

Protocol Π_{ACS}

- **Phase I — Distributing Points on the Polynomials**
  - On having the input $f_i(\cdot)$, act as a dealer D and invoke an instance $\Pi^{(i)}_{\mathsf{VSS}}$ of Π_{VSS} with input $f_i(\cdot)$.
  - For $j = 1, \ldots, n$, participate in the instance $\Pi^{(j)}_{\mathsf{VSS}}$ invoked by P_j and wait for time T_{VSS}.
  - Initialize a set $\mathcal{C}_i = \emptyset$ after time T_{VSS} and include P_j in $\mathcal{C}_i$, if an output is computed during $\Pi^{(j)}_{\mathsf{VSS}}$.
- **Phase II — Identifying the Common Subset of Parties**:
  - For $j = 1, \ldots, n$, participate in an instance of $\Pi^{(j)}_{\mathsf{BA}}$ of Π_{BA} with input 1, if $P_j \in \mathcal{C}_i$.
  - Once $n - t_s$ instances of Π_{BA} have produced an output 1, then participate with input 0 in all the Π_{BA} instances $\Pi^{(j)}_{\mathsf{BA}}$, such that $P_j \notin \mathcal{C}_i$.
  - Once all the n instances of Π_{BA} have produced a binary output, then output $\mathcal{CS}$, which is the set of parties P_j, such that 1 is obtained as the output in the instance $\Pi^{(j)}_{\mathsf{BA}}$.

Figure 5: Agreement on common subset of $n - t_s$ parties where each party has a single t_s-degree polynomial as input. The above code is executed by every $P_i \in \mathcal{P}$.

We next prove the properties of the protocol Π_{ACS}.

Lemma 5.1. *Protocol Π_{ACS} achieves the following properties, where every party P_i has a t_s-degree polynomial $f_i(\cdot)$ as input.*

[11] The exact input of P_i will be determined, based on where exactly the ACS protocol is used.

- *Synchronous Network: The following is achieved in the presence of up to t_s corruptions.*
  - *t_s-Correctness: at time $T_{\mathsf{ACS}} = T_{\mathsf{VSS}} + 2T_{\mathsf{BA}}$, the parties output a common subset $\mathcal{CS}$ of size at least $n - t_s$, such that all the following hold:*
    - *All honest parties will be present in $\mathcal{CS}$.*
    - *Corresponding to every honest $P_j \in \mathcal{CS}$, every honest P_i has $f_j(\alpha_i)$.*
    - *Corresponding to every corrupt $P_j \in \mathcal{CS}$, there exists some t_s-degree polynomial, say $f_j^\star(\cdot)$, such that every honest P_i has $f_j^\star(\alpha_i)$.*
- *Asynchronous Network: The following is achieved in the presence of up to t_a corruptions.*
  - *t_a-Correctness: almost-surely, the honest parties eventually output a common subset $\mathcal{CS}$ of size at least $n - t_s$, such that all the following hold:*
    - *Corresponding to every honest $P_j \in \mathcal{CS}$, every honest P_i eventually has $f_j(\alpha_i)$.*
    - *Corresponding to every corrupt $P_j \in \mathcal{CS}$, there exists some t_s-degree polynomial, say $f_j^\star(\cdot)$, such that every honest P_i eventually has $f_j^\star(\alpha_i)$.*
- *t_s-Privacy: Irrespective of the network type, the view of the adversary remains independent of the $f_i(\cdot)$ polynomials of the honest parties.*
- *Irrespective of the network type, the protocol incurs a communication of $\mathcal{O}(n^6 \log |\mathbb{F}|)$ bits from the honest parties and invokes $\mathcal{O}(n^2)$ instances of Π_{BA}.*

Proof. The *t_s-privacy* property simply follows from the *t_s-privacy* property of Π_{VSS}, while communication complexity follows from the communication complexity of Π_{VSS} and the fact that $\mathcal{O}(n)$ instances of Π_{VSS} are invoked. We next prove the *correctness* property.

We first consider a *synchronous* network, with up to t_s corruptions. Let $\mathcal{H}$ be the set of parties, where $|\mathcal{H}| \geq n - t_s$. Corresponding to each $P_j \in \mathcal{H}$, every *honest* P_i computes the output $f_j(\alpha_i)$ at time T_{VSS} during $\Pi_{\mathsf{VSS}}^{(j)}$, which follows from the *t_s-correctness* of Π_{VSS} in the *synchronous* network. Consequently, at time T_{VSS}, the set $\mathcal{C}_i$ will be of size at least $n - t_s$ for every honest P_i. Now corresponding to each $P_j \in \mathcal{H}$, each honest P_i participates with input 1 in the instance $\Pi_{\mathsf{BA}}^{(j)}$ at time T_{VSS}. Hence, from the *t_s-validity* and *t_s-guaranteed liveness* of Π_{BA} in the *synchronous* network, it follows that at time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$, every honest P_i computes the output 1 during the instance $\Pi_{\mathsf{BA}}^{(j)}$, corresponding to every $P_j \in \mathcal{H}$. Consequently, at time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$, every *honest* party will start participating in the remaining Π_{BA} instances for which no input has been provided yet (if there are any). And from the *t_s-guaranteed liveness* and *t_s-consistency* of Π_{BA} in the *synchronous* network, these Π_{BA} instances will produce common outputs for every honest party at time $T_{\mathsf{ACS}} = T_{\mathsf{VSS}} + 2T_{\mathsf{BA}}$. Since the set $\mathcal{CS}$ is determined deterministically based on the outputs computed from the n instances of Π_{BA}, it follows that all the honest parties eventually output the same $\mathcal{CS}$ of size at least $n - t_s$, such that each $P_j \in \mathcal{H}$ will be present in $\mathcal{CS}$. We next wish to show that corresponding to *every* $P_j \in \mathcal{CS}$, every honest party has received its point on P_j's polynomial.

Consider an *arbitrary* party $P_j \in \mathcal{CS}$. If P_j is *honest*, then as argued above, every honest P_i gets $f_j(\alpha_i)$ at time T_{VSS} itself. Next, consider a *corrupt* $P_j \in \mathcal{CS}$. Since $P_j \in \mathcal{CS}$, it follows that the instance $\Pi_{\mathsf{BA}}^{(j)}$ produces the output 1. From the *t_s-validity* property of Π_{BA} in the *synchronous* network, it follows that at least one *honest* P_i must have participated with input 1 in the instance $\Pi_{\mathsf{BA}}^{(j)}$. This implies that P_i must have computed some output during the instance $\Pi_{\mathsf{VSS}}^{(j)}$ by time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$ and $P_j \in \mathcal{C}_i$. This is because if at time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$, party P_j *does not* belong to the $\mathcal{C}_i$ set of *any* honest P_i, then it implies that *all honest* parties participate with input 0 in the instance $\Pi_{\mathsf{BA}}^{(j)}$ from time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$. Then, from the *$t_s$-validity* of Π_{BA} in the *synchronous* network, every honest party would compute the output 0 in the instance $\Pi_{\mathsf{BA}}^{(j)}$ and hence P_j will not be present in $\mathcal{CS}$, which is a contradiction. Now if P_i has computed some output during $\Pi_{\mathsf{VSS}}^{(j)}$ at time $T_{\mathsf{VSS}} + T_{\mathsf{BA}}$, then from the *$t_s$-strong-commitment* of Π_{VSS}, it follows that P_j has some t_s-degree polynomial, say

$f_j^\star(\cdot)$, such that every honest party P_i computes $f_j^\star(\alpha_i)$ by time $T_{\mathsf{VSS}} + T_{\mathsf{BA}} + 2\Delta$. Since $2\Delta < T_{\mathsf{BA}}$, it follows that at time T_{ACS}, every honest P_i has $f_j^\star(\alpha_i)$, thus proving the t_s-*correctness* property in a *synchronous* network.

We next consider an *asynchronous* network, with up to t_a corruptions. Let $\mathcal{H}$ be the set of parties, where $|\mathcal{H}| \geq n - t_a \geq n - t_s$. We first note that irrespective of way messages are scheduled, there will be at least $n - t_s$ instances of Π_{BA} in which all honest parties eventually participate with input 1. This is because corresponding to every $P_j \in \mathcal{H}$, every *honest* P_i *eventually* computes the output $f_j(\alpha_i)$ during the instance $\Pi_{\mathsf{VSS}}^{(j)}$, which follows from the t_a-*correctness* of Π_{VSS} in the *asynchronous* network. So even if the *corrupt* parties P_j do not invoke their respective $\Pi_{\mathsf{VSS}}^{(j)}$ instances, there will be at least $n - t_s$ instances of Π_{BA} in which all *honest* parties eventually participate with input 1. Consequently, from the t_a-*almost-surely liveness* and t_a-*validity* properties of Π_{BA} in the *asynchronous* network, almost-surely, all honest parties eventually compute the output 1 during these Π_{BA} instances. Hence, all honest parties eventually participate with some input in the remaining Π_{BA} instances. Consequently, from the t_a-*almost-surely liveness* and t_a-*consistency* properties of Π_{BA} in the *asynchronous* network, almost-surely, all honest parties will compute some common output in these Π_{BA} instances as well. Since the set $\mathcal{CS}$ is determined deterministically based on the outputs computed from the n instances of Π_{BA}, it follows that all the honest parties eventually output the same $\mathcal{CS}$.

Now consider an arbitrary party $P_j \in \mathcal{CS}$. It implies that the honest parties compute the output 1 during the instance $\Pi_{\mathsf{BA}}^{(j)}$. From the t_a-*validity* of Π_{BA} in the *asynchronous* network, it follows that at least one *honest* P_i participated with input 1 during $\Pi_{\mathsf{BA}}^{(j)}$, after computing some output in the instance $\Pi_{\mathsf{VSS}}^{(j)}$. Now if P_j is *honest*, then the t_a-*correctness* of Π_{VSS} in the *asynchronous* network guarantees that every honest party P_i *eventually* computes the output $f_j(\alpha_i)$ during $\Pi_{\mathsf{VSS}}^{(j)}$. On the other hand, if P_j is *corrupt*, then the t_a-*strong commitment* of Π_{VSS} in the *asynchronous* network guarantees that there exists some t_s-degree polynomial, say $f_j^\star(\cdot)$, such that every honest party P_i eventually computes the output $f_j^\star(\alpha_i)$ during the instance $\Pi_{\mathsf{VSS}}^{(j)}$. □

We end this section by discussing the modifications needed in the protocol Π_{ACS}, if each party has L number of polynomials as input.

Protocol Π_{ACS} for Multiple Polynomials: Protocol Π_{ACS} can be easily extended if each party has L number of t_s-degree polynomials as input. In this case, each party P_j will invoke its instance of Π_{VSS} with L polynomials. The rest of the protocol steps remain the same. The protocol will incur a communication of $\mathcal{O}(n^4 L \log |\mathbb{F}| + n^6 \log |\mathbb{F}|)$ bits from the honest parties and invokes $\mathcal{O}(n^2)$ instances of Π_{BA}.

6 The Preprocessing Phase Protocol

In this section, we present our best-of-both-worlds protocol for the preprocessing phase. The goal of the protocol is to generate c_M number of t_s-shared multiplication-triples, which are random from the point of view of the adversary. The protocol is obtained by extending the framework of [26] to the best-of-both-worlds setting. We first start by discussing the various (best-of-both-worlds) building blocks used in the protocol.

6.1 best-of-both-worlds Beaver's Multiplication Protocol

Given t_s-shared x, y and a t_s-shared triple (a, b, c), protocol Π_{Beaver} [8] outputs a t_s-shared z, where $z = x \cdot y$, if and only if $c = a \cdot b$. If (a, b, c) is random for the adversary, then x and y remain random for the adversary. In the protocol, the parties first *publicly* reconstruct $x - a$ and $y - b$. A t_s-sharing of z can be then computed locally, since $[z] = (x - a) \cdot (y - b) + (x - a) \cdot [b] + (y - b) \cdot [a] + [c]$. The protocol takes Δ time in a *synchronous* network and in an *asynchronous* network, the parties *eventually* compute [z].

Protocol $\Pi_{\mathsf{Beaver}}(([x], [y]), ([a], [b], [c]))$

- **Masking Input Values** — parties locally compute $[e] = [x] - [a]$ and $[d] = [y] - [b]$.
- **Publicly Reconstructing Masked Inputs** — each $P_i \in \mathcal{P}$ does the following:
  - Send the share of e and d to every party in $\mathcal{P}$ and wait for Δ time.
  - Apply the $\mathsf{OEC}(t_s, t_s, \mathcal{P})$ procedure on the received shares of d to compute d. Similarly, apply the $\mathsf{OEC}(t_s, t_s, \mathcal{P})$ procedure on the received shares of e to compute e.
- **Output Computation** — parties locally compute $[z] = d \cdot e + e \cdot [b] + d \cdot [a] + [c]$ and output $[z]$.

Figure 6: Beaver's protocol for multiplying two t_s shared values.

Lemma 6.1. *Let x and y be two t_s-shared values and let (a, b, c) be a t_s-shared triple. Then protocol Π_{Beaver} achieves the following properties in the presence of up to t_s corruptions.*

- *If the network is synchronous, then within time Δ, the parties output a t_s-sharing of z.*
- *If the network is asynchronous, then the parties eventually output a t_s-sharing of z.*
- *Irrespective of the network type, $z = x \cdot y$ holds, if and only if (a, b, c) is a multiplication-triple.*
- *Irrespective of the network type, if (a, b, c) is random from the point of view of the adversary, then the view of the adversary remains independent of x and y.*
- *The protocol incurs a communication of $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits from the honest parties.*

Proof. Since x, y and the triple (a, b, c) are all t_s-shared, the values $d = (x - a)$ and $e = (y - b)$ will be t_s-shared, which follows from the linearity of t_s-sharing. Let there be up to t_s corruptions. If the network is *synchronous*, then from the properties of OEC in the *synchronous* network, within time Δ, *every* honest P_i will have d and e and hence the parties output a t_s-sharing of z within time Δ. On the other hand, if the network is *asynchronous*, then from the properties of OEC in the *asynchronous* network, *every* honest P_i *eventually* reconstructs d and e and hence the honest parties eventually output a t_s-sharing of z.

In the protocol, $z = (x - a) \cdot (y - b) + (x - a) \cdot b + (y - b) \cdot a + c = x \cdot y - a \cdot b + c$ holds. Hence it follows that $z = x \cdot y$ holds if and only if $c = a \cdot b$ holds.

In the protocol, adversary learns the values d and e, as they are publicly reconstructed. However, if a and b are random from the point of view of the adversary, then d and e leak no information about x and y. Namely, for every candidate x and y, there exist unique a and b, consistent with d and e.

The communication complexity follows from the fact each party needs to send 2 field elements to every other party. □

6.2 best-of-both-worlds Triple-Transformation Protocol

Protocol $\Pi_{\mathsf{TripTrans}}$ takes input a set of $2d + 1$ t_s-shared triples $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i=1,\ldots,2d+1}$, where the triples may not be "related". The output of the protocol are "co-related" t_s-shared triples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i=1,\ldots,2d+1}$, such that all the following hold (irrespective of the network type):

– There exist d-degree polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $2d$-degree polynomial $\mathsf{Z}(\cdot)$, such that $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}$, $\mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds for $i = 1, \ldots, 2d+1$.
– The triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a multiplication-triple if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication-triple. This further implies that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds if and only if all the $2d+1$ input triples are multiplication-triples.
– Adversary learns the triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ if and only if it knows the input triple $(x^{(i)}, y^{(i)}, z^{(i)})$.

The idea behind $\Pi_{\mathsf{TripTrans}}$ is as follows: the polynomials $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ are "defined" by the first and second components of the *first* $d+1$ input triples. Hence the first $d+1$ points on these polynomials are already t_s-shared. The parties then compute d "new" points on the polynomials $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ in a shared fashion. This step requires the parties to perform only *local computations*. This is because from the property of Lagrange's interpolation, computing any new point on $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ involves computing a *publicly-known linear* function (which we call *Lagrange's linear function*) of "old" points on these polynomials. Since the old points are t_s-shared, by applying corresponding Lagrange's functions, the parties can compute a t_s-sharing of the new points. Finally, the parties compute a t_s-sharing of the product of the d new points using Beaver's technique, making use of the *remaining* d input triples. The $\mathsf{Z}(\cdot)$ polynomial is then defined by the d computed products and the third component of the first $d+1$ input triples. The protocol is formally presented in Fig 7.

Protocol $\Pi_{\mathsf{TripTrans}}(d, \{[x^{(i)}], [y^{(i)}], [z^{(i)}]\}_{i=1,\ldots,2d+1})$

– **Defining $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ Polynomials** — The parties locally do the following:
  – For $i = 1, \ldots, d+1$, set
    – $[\mathbf{x}^{(i)}] = [x^{(i)}]$;
    – $[\mathbf{y}^{(i)}] = [y^{(i)}]$;
    – $[\mathbf{z}^{(i)}] = [z^{(i)}]$.
  – Let $\mathsf{X}(\cdot)$ be the unique d-degree polynomial, passing through the points $\{(\alpha_i, \mathbf{x}^{(i)})\}_{i=1,\ldots,d+1}$. And let $\mathsf{Y}(\cdot)$ be the unique d-degree polynomial, passing through $\{(\alpha_i, \mathbf{y}^{(i)})\}_{i=1,\ldots,d+1}$.
    – For $i = d+2, \ldots, 2d+1$, locally compute $[\mathbf{x}^{(i)}] = [\mathsf{X}(\alpha_i)]$ from $\{[\mathbf{x}^{(i)}]\}_{i=1,\ldots,d+1}$, by applying the corresponding Lagrange's linear function.
    – For $i = d+2, \ldots, 2d+1$, locally compute $[\mathbf{y}^{(i)}] = [\mathsf{Y}(\alpha_i)]$ from $\{[\mathbf{y}^{(i)}]\}_{i=1,\ldots,d+1}$, by applying the corresponding Lagrange's linear function.
– **Computing Points on the $\mathsf{Z}(\cdot)$ Polynomial** — The parties do the following:
  – For $i = d+2, \ldots, 2d+1$, participate in the instance $\Pi_{\mathsf{Beaver}}(([\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}]), ([x^{(i)}], [y^{(i)}], [z^{(i)}]))$ of Π_{Beaver}. Let $[\mathbf{z}^{(i)}]$ be the output obtained from this instance.
  – Output $\{[\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}], [\mathbf{z}^{(i)}]\}_{i=1,\ldots,2d+1}$.

Figure 7: Protocol for transforming a set of t_s-shared triples into a set of correlated t_s-shared triples.

We next prove the properties of the protocol $\Pi_{\mathsf{TripTrans}}$.

Lemma 6.2. *Let $\{[x^{(i)}], [y^{(i)}], [z^{(i)}]\}_{i=1,\ldots,2d+1}$ be a set of t_s-shared triples. Then protocol $\Pi_{\mathsf{TripTrans}}$ achieves the following properties in the presence of up to t_s corruptions.*

– *If the network is synchronous, then the parties output t_s-shared triples $\{[\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}], [\mathbf{z}^{(i)}]\}_{i=1,\ldots,2d+1}$, within time Δ.*
– *If the network is asynchronous, then the parties eventually output t_s-shared triples $\{[\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}], [\mathbf{z}^{(i)}]\}_{i=1,\ldots,2d+1}$.*
– *Irrespective of the network type, there exist d-degree polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $2d$-degree polynomial $\mathsf{Z}(\cdot)$, such that $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}$, $\mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds for $i = 1, \ldots, 2d+1$.*
– *Irrespective of the network type, $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a multiplication-triple if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication-triple.*

– *For $i = 1, \ldots, 2d+1$, no additional information about $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is revealed to the adversary, if the triple $(x^{(i)}, y^{(i)}, z^{(i)})$ is random from the point of view of the adversary.*
– *The protocol incurs a communication of $\mathcal{O}(dn^2 \log |\mathbb{F}|)$ bits from the honest parties.*

Proof. Consider an adversary who controls up to t_s parties. In the protocol, irrespective of the network type, the parties *locally* compute the t_s-sharings $\{[\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}]\}_{i=1,\ldots,2d+1}$ and t_s-sharings $\{[\mathbf{z}^{(i)}]\}_{i=1,\ldots,d+1}$. If the network is *synchronous*, then from the properties of Π_{Beaver} in the *synchronous* network, it follows that after time Δ, all honest parties will have their respective output in all the d instances of Π_{Beaver}. Hence after time Δ, the parties have t_s-sharings $\{[\mathbf{z}^{(i)}]\}_{i=d+2,\ldots,2d+1}$. On the other hand, if the network is *asynchronous*, then from the properties of Π_{Beaver} in the *asynchronous* network, all honest parties eventually compute their output in all the d instances of Π_{Beaver}. Hence the parties eventually compute the t_s-sharings $\{[\mathbf{z}^{(i)}]\}_{i=d+2,\ldots,2d+1}$ and hence eventually compute their output in the protocol.

Next consider an *arbitrary* $i \in \{1, \ldots, d+1\}$. Since $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)}) = (x^{(i)}, y^{(i)}, z^{(i)})$, it follows that $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ will be a multiplication-triple if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication-triple. Now consider an arbitrary $i \in \{d+2, \ldots, 2d+1\}$. Since $[\mathbf{z}^{(i)}]$ is the output of the instance $\Pi_{\mathsf{Beaver}}(([\mathbf{x}^{(i)}], [\mathbf{y}^{(i)}]), ([x^{(i)}], [y^{(i)}], [z^{(i)}]))$, it follows from the properties of Π_{Beaver} that $\mathbf{z}^{(i)} = \mathbf{x}^{(i)} \cdot \mathbf{y}^{(i)}$ holds, if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication-triple.

From the protocol steps, it is easy to see that the polynomials $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ defined in the protocols are d-degree polynomials, as they are defined through $d+1$ distinct points $\{(\alpha_i, \mathbf{x}^{(i)})\}_{i=1,\ldots,d+1}$ and $\{(\alpha_i, \mathbf{y}^{(i)})\}_{i=1,\ldots,d+1}$ respectively. On the other hand, $\mathsf{Z}(\cdot)$ is a $2d$-degree polynomial, as it is defined through the $2d+1$ distinct points $\{(\alpha_i, \mathbf{z}^{(i)})\}_{i=1,\ldots,2d+1}$.

For any $i \in \{1, \ldots, d+1\}$, if $(x^{(i)}, y^{(i)}, z^{(i)})$ is random from the point of view of the adversary, then $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is also random from the point of view of the adversary, since $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)}) = (x^{(i)}, y^{(i)}, z^{(i)})$. On the other hand for any $i \in \{d+2, \ldots, 2d+1\}$, if $(x^{(i)}, y^{(i)}, z^{(i)})$ is random from the point of view of the adversary, then from the properties of Π_{Beaver}, it follows that no additional information is learnt about $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$.

The communication complexity follows from the fact that there are d instances of Π_{Beaver} invoked in the protocol. □

6.3 best-of-both-worlds Triple-Sharing Protocol

We next present a triple-sharing protocol Π_{TripSh}, which allows a dealer D to *verifiably* t_s-share L multiplication-triples. If D is *honest*, then the triples remain random from the point of view of the adversary and all honest parties output the shares of D's multiplication-triples. On the other hand, if D is *corrupt*, then the protocol *need not* produce any output, even in a *synchronous* network, as a corrupt D may not invoke the protocol at the first place and the parties will not be aware of the network type. However, the "verifiability" of Π_{TripSh} guarantees that if the honest parties compute any output corresponding to a *corrupt* D, then D has indeed t_s-shared multiplication-triples.

For simplicity, we present the protocol assuming D has a *single* multiplication-triple to share and the protocol can be easily generalized for any $L > 1$. The idea behind the protocol is as follows: D picks a random multiplication-triple and t_s-shares it by invoking an instance of Π_{VSS}. To prove that it has indeed shared a multiplication-triple, D actually t_s-shares $2t_s+1$ random multiplication-triples. The parties then run an instance of $\Pi_{\mathsf{TripTrans}}$ and "transform" these shared triples into "co-related" shared triples, constituting distinct points on the triplet of polynomials $(\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot))$, which are guaranteed to exist during $\Pi_{\mathsf{TripTrans}}$. Then, to check if all the triples shared by D are multiplication-triples, it is sufficient to verify if $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds. To verify the latter, we incorporate a mechanism which enables the parties to *publicly* learn if $\mathsf{Z}(\alpha_j) = \mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j)$ holds

under P_j's "supervision" in such a way that if P_j is *honest*, then the supervised verification of the triplet $(\mathsf{X}(\alpha_j), \mathsf{Y}(\alpha_j), \mathsf{Z}(\alpha_j))$ is "successful" if and only if the triplet is a multiplication-triple. Moreover, the privacy of the triplet will be maintained during the supervised verification for an *honest* D *and* P_j. The goal is to check whether there are at least $2t_s + 1$ successful supervised-verifications, performed under the supervision of *honest* supervisors P_j, which will then confirm that indeed $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds. This is because $\mathsf{Z}(\cdot)$ is a $2t_s$-degree polynomial. Upon confirming that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds, the parties compute a "new" point on the polynomials (in a shared fashion), which is taken as the output triple shared *on behalf of* D. We stress that the output triple is well defined and will be "known" to D, as it is *deterministically* determined from the triples shared by D. If D is *honest*, then the privacy of the output triple is guaranteed from the fact that during the supervised verification, an adversary may learn at most t_s distinct points on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, corresponding to the *corrupt* supervisors.

The supervised verification of the (shared) points on the polynomials is performed as follows: the parties invoke an instance of Π_{ACS}, where the input for each party is a triplet of random t_s-degree polynomials, whose constant terms constitute a random multiplication-triple, called *verification-triple*. The instance of Π_{ACS} is invoked in parallel with D's invocation of Π_{VSS}. Through the instance of Π_{ACS}, the parties agree upon a set $\mathcal{W}$ of at least $n - t_s$ supervisors, whose shared verification-triples are used to verify the points on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$. Namely, if $P_j \in \mathcal{W}$ has shared the verification-triple $(u^{(j)}, v^{(j)}, w^{(j)})$, then in the supervised verification under P_j, parties *publicly* reconstruct and check if $\mathsf{Z}(\alpha_j) - \mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j) = 0$ holds. For this, the parties recompute $\mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j)$ in a shared fashion using Beaver's method, by deploying the shared verification-triple $(u^{(j)}, v^{(j)}, w^{(j)})$. If $\mathsf{Z}(\alpha_j) - \mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j)$ *does not* turn out to be 0 (implying that either D is *corrupt* or P_j's verification-triple is *not* a multiplication-triple), then the parties *publicly* reconstruct and check if $(\mathsf{X}(\alpha_j), \mathsf{Y}(\alpha_j), \mathsf{Z}(\alpha_j))$ is a multiplication-triple and discard D if the triple *does not* turn out to be a multiplication-triple.

An *honest* D will *never* be discarded. Moreover, in a *synchronous* network, *all honest* parties P_j are guaranteed to be present in $\mathcal{W}$ (follows from the t_s-*correctness* of Π_{ACS} in the *synchronous* network) and hence, there will be at least $n - t_s > 2t_s + 1$ *honest* supervisors in $\mathcal{W}$. On the other hand, even in an *asynchronous* network, there will be at least $n - t_s - t_a > 2t_s$ *honest* supervisors in $\mathcal{W}$. Hence if a *corrupt* D is *not* discarded, then it is guaranteed that D has shared multiplication-triples.

Protocol Π_{TripSh}

- **Phase I — Sharing Triples and Verification-Triples**:
  - D selects $2t_s + 1$ random multiplication-triples $\{(x^{(j)}, y^{(j)}, z^{(j)})\}_{j=1,\ldots,2t_s+1}$. It then selects random t_s-degree polynomials $\{f_{x^{(j)}}(\cdot), f_{y^{(j)}}(\cdot), f_{z^{(j)}}(\cdot)\}_{j=1,\ldots,2t_s+1}$, such that $f_{x^{(j)}}(0) = x^{(j)}$, $f_{y^{(j)}}(0) = y^{(j)}$ and $f_{z^{(j)}}(0) = z^{(j)}$. D then invokes an instance of Π_{VSS} with input $\{f_{x^{(j)}}(\cdot), f_{y^{(j)}}(\cdot), f_{z^{(j)}}(\cdot)\}_{j=1,\ldots,2t_s+1}$ and the parties in $\mathcal{P}$ participate in this instance.
  - In parallel, each party $P_i \in \mathcal{P}$ randomly selects a *verification multiplication-triple* $(u^{(i)}, v^{(i)}, w^{(i)})$ and random t_s-degree polynomials $f_{u^{(i)}}(\cdot), f_{v^{(i)}}(\cdot)$ and $f_{w^{(i)}}(\cdot)$ where $f_{u^{(i)}}(0) = u^{(i)}$, $f_{v^{(i)}}(0) = v^{(i)}$ and $f_{w^{(i)}}(0) = w^{(i)}$. With these polynomials as inputs, P_i participates in an instance of Π_{ACS} and waits for time T_{ACS}. Let $\mathcal{W}$ be the set of parties, computed as the output during the instance of Π_{ACS}, where $|\mathcal{W}| \geq n - t_s$.
- **Phase II — Transforming D's Triples**:
  - Upon computing an output in the instance of Π_{VSS} invoked by D, the parties participate in an instance $\Pi_{\mathsf{TripTrans}}(t_s, \{[x^{(j)}], [y^{(j)}], [z^{(j)}]\}_{j=1,\ldots,2t_s+1})$ of $\Pi_{\mathsf{TripTrans}}$.
  - Let $\{[\mathbf{x}^{(j)}], [\mathbf{y}^{(j)}], [\mathbf{z}^{(j)}]\}_{j=1,\ldots,2t_s+1})$ be the set of t_s-shared triples computed during $\Pi_{\mathsf{TripTrans}}$. And let $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ be the t_s-degree polynomials and $\mathsf{Z}(\cdot)$ be the $2t_s$-degree polynomial, which are guaranteed to exist during the instance of $\Pi_{\mathsf{TripTrans}}$, such that $\mathsf{X}(\alpha_j) = \mathbf{x}^{(j)}$,

$\mathsf{Y}(\alpha_j) = \mathbf{y}^{(j)}$ and $\mathsf{Z}(\alpha_j) = \mathbf{z}^{(j)}$, for $j = 1, \ldots, 2t_s + 1$.
- For $j = 2t_s + 2, \ldots, n$, the parties do the following.
  - Locally compute $[\mathbf{x}^{(j)}] = [\mathsf{X}(\alpha_j)]$ from $\{[\mathbf{x}^{(j)}]\}_{j=1,\ldots,t_s+1}$, by using appropriate Lagrange's linear functions.
  - Locally compute $[\mathbf{y}^{(j)}] = [\mathsf{Y}(\alpha_j)]$ from $\{[\mathbf{y}^{(j)}]\}_{j=1,\ldots,t_s+1}$, by using appropriate Lagrange's linear functions.
  - Locally compute $[\mathbf{z}^{(j)}] = [\mathsf{Z}(\alpha_j)]$ from $\{[\mathbf{z}^{(j)}]\}_{j=1,\ldots,2t_s+1}$ respectively, by using appropriate Lagrange's linear functions.

- **Phase III — Verifying Transformed Triples**: The parties do the following.
  - **Phase III(a) — Recomputing the Products**:
    - Corresponding to each $P_j \in \mathcal{W}$, participate in the instance $\Pi_{\mathsf{Beaver}}(([\mathbf{x}^{(j)}], [\mathbf{y}^{(j)}]), ([u^{(j)}], [v^{(j)}], [w^{(j)}]))$ of Π_{Beaver} to compute $[\mathfrak{z}^{(j)}]$.
  - **Phase III(b) — Computing and Publicly Reconstructing the Differences**:
    - Corresponding to every $P_j \in \mathcal{W}$, the parties locally compute $[\gamma^{(j)}] = [\mathbf{z}^{(j)}] - [\mathfrak{z}^{(j)}]$.
    - Corresponding to every $P_j \in \mathcal{W}$, the parties publicly reconstruct $\gamma^{(j)}$, by exchanging their respective shares of $\gamma^{(j)}$, followed by applying the $\mathsf{OEC}(t_s, t_s, \mathcal{P})$ procedure on the received shares.
    - Corresponding to $P_j \in \mathcal{W}$, party $P_i \in \mathcal{P}$ upon reconstructing $\gamma^{(j)}$, sets a Boolean variable $\mathsf{flag}_i^{(j)}$ to 0 if $\gamma^{(j)} = 0$, else it sets $\mathsf{flag}_i^{(j)}$ to 1.
  - **Phase III(c) — Checking the Suspected Triples**: Each $P_i \in \mathcal{P}$ does the following.
    - For every $P_j \in \mathcal{W}$ such that $\mathsf{flag}_i^{(j)} = 1$, send the shares corresponding to $[\mathbf{x}^{(j)}], [\mathbf{y}^{(j)}]$ and $[\mathbf{z}^{(j)}]$ to every party.
    - For every $P_j \in \mathcal{W}$ such that $\mathsf{flag}_i^{(j)} = 1$, apply the $\mathsf{OEC}(t_s, t_s, \mathcal{P})$ procedure on the received shares corresponding to $[\mathbf{x}^{(j)}], [\mathbf{y}^{(j)}]$ and $[\mathbf{z}^{(j)}]$, to reconstruct the triple $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$.
    - For every $P_j \in \mathcal{W}$ such that $\mathsf{flag}_i^{(j)} = 1$, reset $\mathsf{flag}_i^{(j)}$ to 0 if $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ is a multiplication-triple.
    - If $\mathsf{flag}_i^{(j)} = 0$, corresponding to every $P_j \in \mathcal{W}$, then set $\mathsf{flag}_i = 0$, else set $\mathsf{flag}_i = 1$.
- **Output Computation**: Each party $P_i \in \mathcal{P}$ does the following.
  - If $\mathsf{flag}_i = 0$ then output shares corresponding to t_s-shared triple $([a], [b], [c])$ *on behalf of* D, where $a = \mathsf{X}(\beta)$, $b = \mathsf{Y}(\beta)$ and $c = \mathsf{Z}(\beta)$ and where $[a]$, $[b]$ and $[c]$ are locally computed from $\{[\mathbf{x}^{(j)}]\}_{j=1,\ldots,t_s+1}$, $\{[\mathbf{y}^{(j)}]\}_{j=1,\ldots,t_s+1}$ and $\{[\mathbf{z}^{(j)}]\}_{j=1,\ldots,2t_s+1}$ respectively by using appropriate Lagrange's linear functions. Here β is a non-zero element from $\mathbb{F}$, distinct from $\alpha_1, \ldots, \alpha_{2t_s+1}$.
  - If $\mathsf{flag}_i = 1$ then output default-shares (namely all shares being 0) corresponding to t_s-shared triple $([0], [0], [0])$ *on behalf of* D.

Figure 8: A protocol for verifiably sharing a single multiplication triple.

We next prove the properties of the protocol of Π_{TripSh}.

Lemma 6.3. *Protocol Π_{TripSh} achieves the following properties.*
- *If D is honest, then the following hold:*
  - *t_s-Correctness: If the network is synchronous, then after time $T_{\mathsf{TripSh}} = T_{\mathsf{ACS}} + 4\Delta$, the honest parties output a t_s-shared multiplication-triple on the behalf of D.*
  - *t_a-Correctness: If the network is asynchronous, then almost-surely, the (honest) parties eventually output a t_s-shared multiplication-triple on the behalf of D.*
  - *t_s-Privacy: Irrespective of the network type, the view of the adversary remains independent of the output multiplication-triple, shared on the behalf of D.*
- *If D is corrupt, then either no honest party computes any output or depending upon the network type, the following hold*
  - *t_s-Strong Commitment: If the network is synchronous, then the (honest) parties eventually output a t_s-shared multiplication-triple on behalf of D. Moreover, if some honest party computes its output shares at time T, then by time $T+2\Delta$, all honest parties will compute*

their respective output shares.

- t_a*-Strong Commitment: The (honest) parties eventually output a* t_s*-shared multiplication-triple on the behalf of* D.

– *The protocol incurs a communication of* $\mathcal{O}(n^6 \log |\mathbb{F}|)$ *bits from the honest parties and invokes* $\mathcal{O}(n^2)$ *instances of* Π_{BA}.

Proof. We first consider an *honest* D and prove the corresponding properties. We first consider a *synchronous* network with up to t_s corruptions. At time T_{VSS}, the multiplication-triples $\{(x^{(j)}, y^{(j)}, z^{(j)})\}_{j=1,\ldots,2t_s+1}$ will be t_s-shared. This follows from the t_s*-correctness* property of Π_{VSS} in the *synchronous* network. Moreover, these triples will be random from the point of view of the adversary, which follows from the t_s*-privacy* property of Π_{VSS}. Since the instance of Π_{ACS} is invoked in parallel with the instance of Π_{VSS} invoked by D, at time T_{ACS}, all honest parties will have a common subset $\mathcal{W}$ from the instance of Π_{ACS}, with *every honest* P_j being present in the $\mathcal{W}$. This follows from the properties of Π_{ACS} in the *synchronous* network. At time $T_{\mathsf{ACS}} + \Delta$, the multiplication-triples shared by D will be transformed and parties will have t_s-shared multiplication-triples $\{(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})\}_{j=1,\ldots,2t_s+1}$ and there will exist t_s-degree polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $2t_s$-degree polynomial $\mathsf{Z}(\cdot)$ where $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds. This follows from the properties of $\Pi_{\mathsf{TripTrans}}$ in the *synchronous* network.

Next, corresponding to *every honest* $P_j \in \mathcal{W}$, the value $\mathfrak{z}^{(j)}$ will be the same as $\mathbf{x}^{(j)} \cdot \mathbf{y}^{(j)}$, which follows from the properties of Π_{Beaver} and the fact that the corresponding verification-triple $(u^{(j)}, v^{(j)}, w^{(j)})$ will be a multiplication-triple. Hence, $\gamma^{(j)} = \mathbf{z}^{(j)} - \mathfrak{z}^{(j)}$ will be 0 and so each *honest* P_i will set $\mathsf{flag}_i^{(j)}$ to 0, without suspecting and reconstructing the triple $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$. Moreover, in this case, no additional information about $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ is revealed, which follows from the properties of Π_{Beaver} and the fact that the verification-triple $(u^{(j)}, v^{(j)}, w^{(j)})$ remains random from the point of view of the adversary. On the other hand, if $P_j \in \mathcal{W}$ is *corrupt*, then $\gamma^{(j)}$ may not be 0. However, in this case each *honest* P_i will reset $\mathsf{flag}_i^{(j)}$ to 0 after reconstructing the corresponding suspected-triple $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$, since it will be a multiplication-triple. The process of computing $\mathfrak{z}^{(j)}$ and the difference $\gamma^{(j)}$ will take 2Δ time and additionally Δ time might be required to publicly reconstruct suspected-triples corresponding to *corrupt* $P_j \in \mathcal{W}$. Hence, at time $T_{\mathsf{ACS}} + 4\Delta$, each *honest* P_i sets $\mathsf{flag}_i = 1$ and hence, the honest parties output t_s-shared triple (a, b, c). Moreover, the triple will be a multiplication-triple, since (a, b, c) is the same as $(\mathsf{X}(\beta), \mathsf{Y}(\beta), \mathsf{Z}(\beta))$. Since at most t_s triples $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ may be publicly reconstructed corresponding to the *corrupt* parties $P_j \in \mathcal{W}$, it follows that adversary will learn at most t_s distinct points on the $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ polynomials. This further implies that $(\mathsf{X}(\beta), \mathsf{Y}(\beta), \mathsf{Z}(\beta))$ will be random from the point of view of the adversary, since $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ are t_s-degree polynomials and $\mathsf{Z}(\cdot)$ is a $2t_s$-degree polynomial. This completes the proof of the t_s*-correctness* in the *synchronous* network, as well as the proof of the t_s*-privacy* property.

If D is *honest* and the network is *asynchronous* with up to t_a corruptions, then the proof of the t_a*-correctness* property is similar to the above proof, except that now we now use the t_a*-correctness* property of Π_{VSS} and the properties of Π_{ACS}, Π_{Beaver} in the *asynchronous* network. Moreover, the privacy property holds since adversary now corrupt $t_a < t_s$ parties.

We next consider a *corrupt* D and prove the strong-commitment properties. We first consider a *synchronous* network with up to t_s corruptions. Note that irrespective of whether D shares any triples through instance of Π_{VSS} or not, all honest parties will output a set $\mathcal{W}$ at time T_{ACS} during the instance of Π_{ACS}, with *every honest* P_j being present in the $\mathcal{W}$. This follows from the properties of Π_{ACS} in the *synchronous* network. If no honest party computes any output in the protocol, then strong-commitment holds trivially. So consider the case when some honest party computes an output. This implies that at least one *honest* party, say P_h, must have computed an output during

the instance of Π_{VSS} invoked by D, as otherwise no honest party computes any output in the protocol. Let T be the time at which P_h has the output for the instance of Π_{VSS} invoked by D. Note that $T \geq T_{\mathsf{VSS}}$ and T could be greater than T_{ACS}, as a *corrupt* D may delay the start of the instances of Π_{VSS}. From the t_s-*strong commitment* property of Π_{VSS} in the *synchronous* network, it then follows that by time $T + 2\Delta$, *all* honest parties compute their output in the instance of Π_{VSS} invoked by D. Hence at time $T + 2\Delta$, there are $2t_s + 1$ triples which are t_s-shared by D.

If $T \leq T_{\mathsf{ACS}} - 2\Delta$, then at time T_{ACS}, all honest parties will have their respective shares corresponding to the t_s-shared triples of D, as well as the set $\mathcal{W}$ and the shares corresponding to the verification-triples, shared by the parties in $\mathcal{W}$. The instance of $\Pi_{\mathsf{TripTrans}}$ will produce its output at time $T_{\mathsf{ACS}} + \Delta$. The follow up instances of Π_{Beaver} to recompute the products will take Δ time, followed by Δ time for publicly reconstructing the difference values $\gamma^{(j)}$. Additionally, the parties may take Δ time to publicly reconstruct any suspected triples. Hence in this case, *all honest* parties will have their respective output shares at time $T_{\mathsf{ACS}} + 4\Delta$.

On the other hand, if $T > T_{\mathsf{ACS}} - 2\Delta$, then each honest party computes its output during the instance of $\Pi_{\mathsf{TripTrans}}$, either at time $T + \Delta$ or at time $T + 3\Delta$. Then, each honest party computes its output from the instances of Π_{Beaver}, either at time $T + 2\Delta$ or at time $T + 4\Delta$. This implies that the difference values $\gamma^{(j)}$ are available with the honest parties, either at time $T + 3\Delta$ or $T + 5\Delta$. Consequently, the suspected triples (if any) will be available with the honest parties, either at time $T + 4\Delta$ or $T + 6\Delta$. Hence each honest party computes its output share in the protocol either at time $T + 4\Delta$ or $T + 6\Delta$. Notice that in this case there might be a difference of at most 2Δ time within which the honest parties compute their output in the protocol, due to a possible difference of 2Δ time in getting the output in the instances of Π_{VSS} invoked by the *corrupt* D.

If the triples shared by D during the instances of Π_{VSS} are all *multiplication-triples*, then similar to the proof of the correctness property for an *honest* D, it follows that the honest parties will output a t_s-shared multiplication-triple on behalf of D. So consider the case when all the triples shared by D are *not* multiplication-triples. This implies that $\mathsf{Z}(\cdot) \neq \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$, where $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ are the t_s-degree polynomials and $\mathsf{Z}(\cdot)$ is the $2t_s$-degree polynomial, which are guaranteed to exist from the protocol $\Pi_{\mathsf{TripTrans}}$. Let P_j be an *honest* party, such that $\mathsf{Z}(\alpha_j) \neq \mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j)$. This further implies that the transformed triple $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ is *not* a multiplication-triple. Such a P_j is bound to exist. This is because there are at least $2t_s + 1$ honest parties P_j. And if $\mathsf{Z}(\alpha_j) = \mathsf{X}(\alpha_j) \cdot \mathsf{Y}(\alpha_j)$ holds corresponding to *every* honest P_j, then it implies that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds (due to the degrees of the respective polynomials), which is a contradiction.

We next show that each *honest* P_i will set $\mathsf{flag}_i^{(j)} = 1$ and hence $\mathsf{flag}_i = 1$. For this, we note that $P_j \in \mathcal{W}$. This follows from the properties of Π_{ACS} in the *synchronous* network, which guarantees that *all* honest parties (and not just P_j) will be present in $\mathcal{W}$. Since the verification-triple $(u^{(j)}, v^{(j)}, w^{(j)})$ shared by P_j will be a multiplication-triple, from the properties of Π_{Beaver} in the *synchronous* network, it follows that $\mathfrak{z}^{(j)} = \mathbf{x}^{(j)} \cdot \mathbf{y}^{(j)}$ holds. But since $\mathbf{z}^{(j)} \neq \mathbf{x}^{(j)} \cdot \mathbf{y}^{(j)}$, it follows that $\gamma^{(j)} = \mathbf{z}^{(j)} - \mathfrak{z}^{(j)} \neq 0$. Consequently, the parties will publicly reconstruct the suspected-triple $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ and find that it is not a multiplication-triple. Hence each *honest* P_i will set $\mathsf{flag}_i^{(j)} = 1$ and hence $\mathsf{flag}_i = 1$. So the parties output a default t_s-sharing of the multiplication-triple $(0, 0, 0)$ on behalf of D.

The proof for the t_a-*strong-commitment* property in the *asynchronous* network is similar to the above proof, except that we now use the t_a-*strong-Commitment* property of Π_{VSS} and the properties of Π_{ACS} and Π_{Beaver} in the *asynchronous* network. Moreover, there will be at least $n - t_s - t_a \geq 2t_s + 1$ honest parties in $\mathcal{W}$, who will lead the verification of at least $2t_s + 1$ distinct points on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, through their respective verification-triples.

The communication complexity follows from communication complexity of $\Pi_{\mathsf{ACS}}, \Pi_{\mathsf{VSS}}, \Pi_{\mathsf{TripTrans}}$

and Π_{Beaver}. □

We next discuss the modifications needed in the protocol Π_{TripSh} to handle L multiplication-triples.

Protocol Π_{TripSh} for Sharing L Multiplication-Triples: Protocol Π_{TripSh} can be easily generalized so that L multiplication-triples are shared on behalf of D. Namely, D now has to share $L \cdot (2t_s + 1)$ random multiplication-triples through Π_{VSS}, while each party P_j will need to select L verification-triples during the instance of Π_{ACS}. Moreover, there will be L instances of $\Pi_{\mathsf{TripTrans}}$ to transform D's shared triples, resulting in L triplets of shared polynomials $(\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot))$, each of which is independently verified by performing supervised verification. To avoid repetition, we do not provide the formal details. The modified Π_{TripSh} protocol will incur a communication of $\mathcal{O}(n^4 L \log|\mathbb{F}| + n^6 \log|\mathbb{F}|)$ bits from the honest parties and invokes $\mathcal{O}(n^2)$ instances of Π_{BA}.

6.4 best-of-both-worlds Triple-Extraction Protocol

Protocol Π_{TripExt} (Fig 9) takes as input a *publicly-known* subset $\mathcal{CS}$ of $2d+1$ parties, where $d \geq t_s$ and where it will be *ensured* that each party $P_j \in \mathcal{CS}$ has t_s-shared a multiplication-triple. It will also be ensured that if P_j is *honest*, then the multiplication-triple is random from the point of view of the adversary. The protocol outputs $d+1-t_s$ number of t_s-shared multiplication-triples, which will be random from the point of view of the adversary. The high level idea of the protocol is very simple. The parties first invoke an instance of $\Pi_{\mathsf{TripTrans}}$ to "transform" the input triples into a set of co-related triples. Since all the input triples are multiplication-triples, the output triples will also be multiplication-triples. Let $(\mathsf{X}(\cdot), \mathsf{Y}(\cdot), \mathsf{Z}(\cdot))$ be the triplet of shared polynomials which is guaranteed to exist after $\Pi_{\mathsf{TripTrans}}$. From the properties of $\Pi_{\mathsf{TripTrans}}$, it follows that adversary will know at most t_s distinct points on these polynomials and hence at least $d+1-t_s$ points on these polynomials are random for the adversary. Hence, the parties output $d+1-t_s$ "new" points on these polynomials (in a t_s-shared fashion), which are guaranteed to be random from the point of view of the adversary. This requires the parties to perform *only* local computation.

Protocol $\Pi_{\mathsf{TripExt}}(\mathcal{CS}, \{[x^{(j)}], [y^{(j)}], [z^{(j)}]\}_{P_j \in \mathcal{CS}})$

- **Transforming the Input Multiplication-Triples** — The parties jointly do the following:
  - Participate in an instance $\Pi_{\mathsf{TripTrans}}(d, \{[x^{(j)}], [y^{(j)}], [z^{(j)}]\}_{P_j \in \mathcal{CS}})$ of $\Pi_{\mathsf{TripTrans}}$.
    - Let $\{[\mathbf{x}^{(j)}], [\mathbf{y}^{(j)}], [\mathbf{z}^{(j)}]\}_{P_j \in \mathcal{CS}}$ be the shared multiplication-triples obtained from $\Pi_{\mathsf{TripTrans}}$. Moreover, let $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ be the d-degree polynomials and $\mathsf{Z}(\cdot)$ be the $2d$-degree polynomial where $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ and where $\mathsf{X}(\alpha_j) = \mathbf{x}^{(j)}$, $\mathsf{Y}(\alpha_j) = \mathbf{y}^{(j)}$ and $\mathsf{Z}(\alpha_j) = \mathbf{z}^{(j)}$ holds corresponding to every $P_j \in \mathcal{CS}$.
    - For $j = 1, \ldots, d+1-t_s$, locally compute $[\mathbf{a}^{(j)}]$, $[\mathbf{b}^{(j)}]$ and $[\mathbf{c}^{(j)}]$ from $\{[\mathbf{x}^{(j)}]\}_{j=1,\ldots,d+1}$, $[\{\mathbf{y}^{(j)}]\}_{j=1,\ldots,d+1}$ and $\{[\mathbf{z}^{(j)}]\}_{j=1,\ldots,2d+1}$ respectively by applying the corresponding Lagrange's linear function. Here $\mathbf{a}^{(j)} = \mathsf{X}(\beta_j)$, $\mathbf{b}^{(j)} = \mathsf{Y}(\beta_j)$ and $\mathbf{c}^{(j)} = \mathsf{Z}(\beta_j)$, where $\beta_1, \ldots, \beta_{d+1-t_s}$ are distinct, non-zero elements from $\mathbb{F}$, different from $\alpha_1, \ldots, \alpha_n$.
    - Output $\{[\mathbf{a}^{(j)}], [\mathbf{b}^{(j)}], [\mathbf{c}^{(j)}]\}_{j=1,\ldots,d+1-t_s}$.

Figure 9: Protocol for extracting $d+1-t_s$ random t_s-shared random multiplication-triples from a set of $2d+1$ t_s-shared multiplication triples, where $d \geq t_s$.

Lemma 6.4. *Let $\mathcal{CS}$ be a set of $2d+1$ parties where $d \geq t_s$, such that each party $P_j \in \mathcal{CS}$ has a multiplication-triple $(x^{(j)}, y^{(j)}, z^{(j)})$ which is t_s-shared. Moreover, if P_j is honest, then the multiplication-triple is random from the point of view of the adversary. Then protocol Π_{TripExt} achieves the following properties.*

– *t_s-Correctness: If the network is synchronous, then after time Δ, the parties output t_s-shared multiplication-triples $\{\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)}\}_{j=1,\ldots,d+1-t_s}$.*
– *t_a-Correctness: If the network is asynchronous, then the parties eventually output t_s-shared multiplication-triples $\{\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)}\}_{j=1,\ldots,d+1-t_s}$.*
– *t_s-Privacy: Irrespective of the network type, the triples $\{\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)}\}_{j=1,\ldots,d+1-t_s}$ will be random from the point of view of the adversary.*
– *The protocol incurs a communication of $\mathcal{O}(dn^2 \log |\mathbb{F}|)$ bits from the honest parties.*

Proof. If the network is *synchronous*, then from the properties of $\Pi_{\mathsf{TripTrans}}$ in the *synchronous* network, it follows that after time Δ, the honest parties have t_s-shared triples $\{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)}\}_{P_j \in \mathcal{CS}}$. Moreover, all these triples will be multiplication-triples, since all the input triples $\{x^{(j)}, y^{(j)}, z^{(j)}\}_{P_j \in \mathcal{CS}}$ are guaranteed to be multiplication-triples. This further implies that the condition $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds, where $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ are the d, d and $2d$-degree polynomials respectively, which are guaranteed to exist from $\Pi_{\mathsf{TripTrans}}$, such that $\mathsf{X}(\alpha_j) = \mathbf{x}^{(j)}$, $\mathsf{Y}(\alpha_j) = \mathbf{y}^{(j)}$ and $\mathsf{Z}(\alpha_j) = \mathbf{z}^{(j)}$ holds for every j, such that $P_j \in \mathcal{CS}$. It now follows that the honest parties output the t_s-shared triples $\{(\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)})\}_{j=1,\ldots,d+1-t_s}$ after time Δ, where $\mathsf{X}(\beta_j) = \mathbf{a}^{(j)}$, $\mathsf{Y}(\beta_j) = \mathbf{b}^{(j)}$ and $\mathsf{Z}(\beta_j) = \mathbf{c}^{(j)}$. Moreover, the triples will be multiplication-triples, because $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ holds.

If the network is *asynchronous*, then the proof of *t_a-correctness* property will be similar as above, except that we now depend upon the properties of $\Pi_{\mathsf{TripTrans}}$ in the *asynchronous* network.

For *privacy*, we note that there will be at most t_s *corrupt* parties in $\mathcal{CS}$ and hence adversary will know at most t_s multiplication-triples in the set $\{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)}\}_{P_j \in \mathcal{CS}}$, which follows from the properties of $\Pi_{\mathsf{TripTrans}}$. This implies that adversary will know at most t_s distinct points on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, leaving $d + 1 - t_s$ degrees of freedom on the these polynomials. This further implies that the multiplication-triples $\{(\mathsf{X}(\beta_j), \mathsf{Y}(\beta_j), \mathsf{Z}(\beta_j))\}_{j=1,\ldots,d+1-t_s}$, which are the same as $\{\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)}\}_{j=1,\ldots,d+1-t_s}$, will be random from the point of view of the adversary. Namely, there will be a one-to-one correspondence between the $d + 1 - t_s$ multiplication-triples in the set $\{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)}\}_{P_j \in \mathcal{CS}}$ which are *unknown* to the adversary and the output multiplication-triples $\{\mathbf{a}^{(j)}, \mathbf{b}^{(j)}, \mathbf{c}^{(j)}\}_{j=1,\ldots,d+1-t_s}$. Hence adversary's view will be consistent with every candidate value of the output $d + 1 - t_s$ multiplication-triples.

The communication complexity simply follows from the fact that the protocol requires one instance of $\Pi_{\mathsf{TripTrans}}$. □

6.5 The best-of-both-worlds Preprocessing Phase Protocol

We finally present our best-of-both-worlds preprocessing phase protocol $\Pi_{\mathsf{PreProcessing}}$, which generates c_M number of t_s-shared multiplication-triples, which will be random from the point of view of the adversary. The protocol is formally presented in Fig 10. In the protocol, each party acts as a dealer and invokes an instance of Π_{TripSh}, so that $\frac{c_M}{(\frac{n-t_s-1}{2}+1-t_s)}$ random multiplication-triples are shared on its behalf. As *corrupt* dealers may not invoke their instances of Π_{TripSh} (even in a *synchronous* network), the parties agree on a *common* subset $\mathcal{CS}$ of $n - t_s$ parties, who have shared multiplication-triples, by executing instances of Π_{BA} (similar to the protocol Π_{ACS}). The multiplication-triples shared on the behalf of up to t_s *corrupt* triple-providers in $\mathcal{CS}$ will be known to adversary, while the multiplication-triples shared on the behalf of the *honest* triple-providers in $\mathcal{CS}$ will be random for the adversary. Since the exact identity of the *honest* triple-providers in $\mathcal{CS}$ will not be known, the parties execute $\frac{c_M}{(\frac{n-t_s-1}{2}+1-t_s)}$ instances of Π_{TripExt} to securely extract c_M shared multiplication-triples, which will be random for the adversary. In the protocol, for simplicity and without loss of generality, we assume that $n - t_s$ is of the form $2d + 1$.

Protocol $\Pi_{\mathsf{PreProcessing}}$

Let $L \stackrel{def}{=} \frac{c_M}{(\frac{n-t_s-1}{2}+1-t_s)}$.

- **Phase 1 — Sharing Random Multiplication-Triples**: Each $P_i \in \mathcal{P}$ does the following.
  - Act as a dealer D and invoke an instance $\Pi^{(i)}_{\mathsf{TripSh}}$ of Π_{TripSh}, so that L random multiplication-triples are shared on P_i's behalf.
  - For $j = 1, \ldots, n$, participate in the instance $\Pi^{(j)}_{\mathsf{TripSh}}$ invoked by P_j and wait for time T_{TripSh}.
  - Initialize a set $\mathcal{C}_i = \emptyset$ and include P_j in $\mathcal{C}_i$, if an output is obtained from $\Pi^{(j)}_{\mathsf{TripSh}}$.
- **Phase II — Agreement on a Common Subset of Triple-Providers**: Each $P_i \in \mathcal{P}$ does the following.
  - For $j = 1, \ldots, n$, participate in an instance of $\Pi^{(j)}_{\mathsf{BA}}$ of Π_{BA} with input 1, if $P_j \in \mathcal{C}_i$.
  - Once $n - t_s$ instances of Π_{BA} have produced an output 1, then participate with input 0 in all the Π_{BA} instances $\Pi^{(j)}_{\mathsf{BA}}$, such that $P_j \notin \mathcal{C}_i$.
  - Once a binary output is computed in all the n instances of Π_{BA}, set $\mathcal{CS}$ to be the set of *first* $n - t_s$ parties P_j, such that 1 is computed as the output in the instance $\Pi^{(j)}_{\mathsf{BA}}$.
- **Phase III — Extracting Random Multiplication-Triples**: The parties do the following.
  - For every $P_j \in \mathcal{CS}$, let $\{[x^{(j,\ell)}], [y^{(j,\ell)}], [z^{(j,\ell)}]\}_{\ell=1,\ldots,L}$ be the t_s-shared multiplication-triples, shared on P_j's behalf, during the instance $\Pi^{(j)}_{\mathsf{TripSh}}$.
  - For $\ell = 1, \ldots, L$, the parties participate in an instance $\Pi_{\mathsf{TripExt}}(\mathcal{CS}, \{[x^{(j,\ell)}], [y^{(j,\ell)}], [z^{(j,\ell)}]\}_{P_j \in \mathcal{CS}})$ of Π_{TripExt} and compute the output $\{[\mathbf{a}^{(j,\ell)}], [\mathbf{b}^{(j,\ell)}], [\mathbf{c}^{(j,\ell)}]\}_{j=1,\ldots,\frac{n-t_s-1}{2}+1-t_s}$.
  - Output the shared triples $\{[\mathbf{a}^{(j,\ell)}], [\mathbf{b}^{(j,\ell)}], [\mathbf{c}^{(j,\ell)}]\}_{j=1,\ldots,\frac{n-t_s-1}{2}+1-t_s, \ell=1,\ldots,L}$.

Figure 10: The best-of-both-worlds preprocessing phase protocol for generating shared random multiplication-triples.

Theorem 6.5. *Protocol $\Pi_{\mathsf{PreProcessing}}$ achieves the following properties.*

- *In a synchronous network, by time $T_{\mathsf{TripGen}} = T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}} + \Delta$, the honest parties output a t_s-sharing of c_M multiplication-triples.*
- *In an asynchronous network, almost-surely, the honest parties eventually output a t_s-sharing of c_M multiplication-triples.*
- *Irrespective of the network type, the view of the adversary remains independent of the output multiplication-triples.*
- *The protocol incurs a communication of $\mathcal{O}(\frac{n^5}{\frac{t_a}{2}+1} c_M \log|\mathbb{F}| + n^7 \log|\mathbb{F}|)$ bits from the honest parties and invokes $\mathcal{O}(n^3)$ instances of Π_{BA}.*

Proof. Let the network be *synchronous* with up to t_s corruptions. Let $\mathcal{H}$ be the set of parties, where $|\mathcal{H}| \geq n - t_s$. Corresponding to each $P_j \in \mathcal{H}$, at time T_{TripSh}, L multiplication-triples $\{x^{(j,\ell)}, y^{(j,\ell)}, z^{(j,\ell)}\}_{\ell=1,\ldots,L}$ will be t_s-shared on the behalf of P_j during the instance $\Pi^{(j)}_{\mathsf{TripSh}}$, which follows from the *t_s-correctness* of Π_{TripSh} in the *synchronous* network. Consequently, the set $\mathcal{C}_i$ will be of size at least $n - t_s$ for every honest P_i. After time T_{TripSh}, corresponding to each $P_j \in \mathcal{H}$, each honest P_i participates with input 1 in the instance $\Pi^{(j)}_{\mathsf{BA}}$. It then follows from the *t_s-validity* and *t_s-guaranteed liveness* of Π_{BA} in the *synchronous* network that corresponding to every $P_j \in \mathcal{H}$, every honest P_i computes the output 1 during the instance $\Pi^{(j)}_{\mathsf{BA}}$, at time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$. Consequently, after time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$, every honest party will start participating in the remaining Π_{BA} instances for which no input has been provided yet (if there are any). And from the *t_s-guaranteed liveness* and *t_s-consistency* of Π_{BA} in the *synchronous* network, all honest parties will compute a common output in these Π_{BA} instances, at time $T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}}$. Consequently, by time $T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}}$, every honest party has a common $\mathcal{CS}$ of size $n - t_s$.

Consider an *arbitrary* party $P_j \in \mathcal{CS}$. If P_j is *honest*, then as shown above, L multiplication-triples will be shared on behalf of P_j at time T_{TripSh} during the instance $\Pi^{(j)}_{\mathsf{TripSh}}$. Next consider a *corrupt* $P_j \in \mathcal{CS}$. Since $P_j \in \mathcal{CS}$, it follows that the honest parties computed the output 1 during the instance $\Pi^{(j)}_{\mathsf{BA}}$. This implies that at least one *honest* P_i must have computed some output during the instance $\Pi^{(j)}_{\mathsf{TripSh}}$, within time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$ (implying that $P_j \in \mathcal{C}_i$) and participated with input 1 in the instance $\Pi^{(j)}_{\mathsf{BA}}$. This is because if P_j *does not* belong to the $\mathcal{C}_i$ set of *any* honest P_i at time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$, then it implies that *all* honest parties participate with input 0 in the instance $\Pi^{(j)}_{\mathsf{BA}}$ after time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$. And then from the t_s-*validity* of Π_{BA} in the *synchronous* network, every honest party would compute the output 0 in the instance $\Pi^{(j)}_{\mathsf{BA}}$ and hence P_j will *not* be present in $\mathcal{CS}$, which is a contradiction. Now if P_i has computed some output in $\Pi^{(j)}_{\mathsf{TripSh}}$ by time $T_{\mathsf{TripSh}} + T_{\mathsf{BA}}$, then from the t_s-*strong-commitment* of Π_{TripSh} in the *synchronous* network, it follows that there exist L multiplication-triples, say $\{(x^{(j,\ell)}, y^{(j,\ell)}, z^{(j,\ell)})\}_{\ell=1,\ldots,L}$, which will be t_s-shared among the parties on behalf of P_j by time $(T_{\mathsf{TripSh}} + T_{\mathsf{BA}} + 2\Delta) < (T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}})$; the latter follows because $2\Delta < T_{\mathsf{BA}}$.

From the above discussion, it follows that there will be L multiplication-triples, which will be t_s-shared on behalf of *each* $P_j \in \mathcal{CS}$ by time $T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}}$. Hence each instance of Π_{TripExt} will output $\frac{n-t_s-1}{2} + 1 - t_s$ number of t_s-shared multiplication-triples by time $T_{\mathsf{TripGen}} = T_{\mathsf{TripSh}} + 2T_{\mathsf{BA}} + \Delta$. This follows from the t_s-*correctness* property of Π_{TripExt} in the *synchronous* network by substituting $|\mathcal{CS}| = n - t_s$ and $d = \frac{n-t_s-1}{2}$ in Lemma 6.4. Since there are $L = \frac{c_M}{(\frac{n-t_s-1}{2}+1-t_s)}$ instances of Π_{TripExt}, it follows that at time T_{TripGen}, the parties have $L \cdot \left(\frac{n-t_s-1}{2} + 1 - t_s\right) = c_M$ number of t_s-shared multiplication-triples. This completes the proof of the t_s-*correctness* property in the *synchronous* network.

The proof of the t_a-*correctness* property in the *asynchronous* network is similar as above, except that we now use the t_a-*correctness* and t_a-*strong-commitment* properties of Π_{TripSh} in the *asynchronous* network, the t_a-*correctness* property of Π_{TripExt} in the *asynchronous* network and the properties of Π_{BA} in the *asynchronous* network.

From the t_s-*privacy* property of Π_{TripSh}, it follows that the multiplication-triples which are t_s-shared on behalf of the *honest* parties $P_j \in \mathcal{CS}$ will be random from the point of view of the adversary under the presence of up to t_s corrupt parties, irrespective of the network type. It then follows from the t_s-*privacy* property of Π_{TripExt} that the t_s-shared multiplication-triples generated from each instance of Π_{TripExt} will be random from the point of view of the adversary. This proves the t_s-*privacy* property.

The communication complexity follows from the communication complexity of Π_{TripSh} and Π_{TripExt}, and from the fact that $\frac{n-t_s-1}{2} + 1 - t_s \geq \frac{t_a}{2} + 1$. □

7 The best-of-both-worlds Circuit-Evaluation Protocol

The best-of-both-worlds protocol Π_{CirEval} for evaluating cir has four phases. In the first phase, the parties generate t_s-sharing of c_M random multiplication-triples through $\Pi_{\mathsf{PreProcessing}}$. The parties also invoke an instance of Π_{ACS} to generate t_s-sharing of their respective inputs for f and agree on a *common* subset $\mathcal{CS}$ of *at least* $n - t_s$ parties, whose inputs for f are t_s-shared, while the remaining inputs are set to 0. In a *synchronous* network, all *honest* parties will be in $\mathcal{CS}$, thus ensuring that the inputs of *all* honest parties are considered for the circuit-evaluation. In the second phase, each gate is evaluated in a t_s-shared fashion after which the parties *publicly* reconstruct the secret-shared output in the third phase. The fourth phase is the *termination phase*, where the

parties check whether "sufficiently many" parties have obtained the same output, in which case the parties "safely" take that output and terminate the protocol (and all the underlying sub-protocols). Protocol Π_{CirEval} is formally presented in Fig 11.

Protocol Π_{CirEval}

- **Preprocessing and Input-Sharing** — The parties do the following:
  - Each $P_i \in \mathcal{P}$ on having the input $x^{(i)}$ for f, selects a random t_s-degree polynomial $f_{x^{(i)}}(\cdot)$ where $f_{x^{(i)}}(0) = x^{(i)}$ and participates in an instance of Π_{ACS} with input $f_{x^{(i)}}(\cdot)$. Let $\mathcal{CS}$ be the common subset of parties, computed as an output during the instance of Π_{ACS}, where $|\mathcal{CS}| \geq n - t_s$. Corresponding to every $P_j \notin \mathcal{CS}$, set $x^{(j)} = 0$ and set $[x^{(j)}]$ to a default t_s-sharing of 0.
  - In parallel, participate in an instance of $\Pi_{\mathsf{PreProcessing}}$. Let $\{[\mathbf{a}^{(j)}], [\mathbf{b}^{(j)}], [\mathbf{c}^{(j)}]\}_{j=1,\ldots,c_M}$ be the t_s-shared multiplication-triples, computed as an output during the instance of $\Pi_{\mathsf{PreProcessing}}$.
- **Circuit Evaluation** — Let $G_1, \ldots, G_m$ be a publicly-known topological ordering of the gates of cir. For $k = 1, \ldots, m$, the parties do the following for gate G_k:
  - *If G_k is an addition gate*: the parties locally compute $[w] = [u] + [v]$, where u and v are gate-inputs and w is the gate-output.
  - *If G_k is a multiplication-with-a-constant gate with constant c*: the parties locally compute $[v] = c \cdot [u]$, where u is the gate-input and v is the gate-output.
  - *If G_k is an addition-with-a-constant gate with constant c*: the parties locally compute $[v] = c + [u]$, where u is the gate-input. and v is the gate-output.
  - *If G_k is a multiplication gate*: Let G_k be the ℓ^{th} multiplication gate in cir where $\ell \in \{1, \ldots, c_M\}$ and let $([a^{(\ell)}], [b^{(\ell)}], [c^{(\ell)}])$ be the ℓ^{th} shared multiplication-triple, generated from $\Pi_{\mathsf{PreProcessing}}$. Moreover, let $[u]$ and $[v]$ be the shared gate-inputs of G_k. Then the parties participate in an instance $\Pi_{\mathsf{Beaver}}(([u], [v]), ([a^{(\ell)}], [b^{(\ell)}], [c^{(\ell)}]))$ of Π_{Beaver} and compute the output $[w]$.
- **Output Computation** — Let $[y]$ be the t_s-shared circuit-output. The parties exchange their respective shares of y and apply the $\mathsf{OEC}(t_s, t_s, \mathcal{P})$ procedure on the received shares to reconstruct y.
- **Termination**: Each P_i does the following.
  - If y has been computed during output computation phase, then send $(\texttt{ready}, y)$ message to all the parties.
  - If the message $(\texttt{ready}, y)$ is received from at least $t_s + 1$ distinct parties, then send $(\texttt{ready}, y)$ message to all the parties, if not sent earlier.
  - If the message $(\texttt{ready}, y)$ is received from at least $2t_s + 1$ distinct parties, then output y and terminate all the sub-protocols.

Figure 11: A best-of-both-worlds perfectly-secure protocol for securely evaluating the arithmetic circuit cir.

We now prove the properties of the protocol Π_{CirEval}.

Theorem 7.1. *Let $t_a < t_s$, such that $3t_s + t_a < n$. Moreover, let $f : \mathbb{F}^n \to \mathbb{F}$ be a function represented by an arithmetic circuit cir over $\mathbb{F}$ consisting of c_M number of multiplication gates, and whose multiplicative depth is D_M. Moreover, let party P_i has input $x^{(i)}$ for f. Then, Π_{CirEval} achieves the following.*

- *In a synchronous network, all honest parties output $y = f(x^{(1)}, \ldots, x^{(n)})$ at time $(120n + D_M + 6k - 20) \cdot \Delta$, where $x^{(j)} = 0$ for every $P_j \notin \mathcal{CS}$, such that $|\mathcal{CS}| \geq n - t_s$ and every honest party $P_j \in \mathcal{P}$ is present in $\mathcal{CS}$. Here k is the constant from Lemma 3.3, as determined by the underlying (existing) perfectly-secure ABA protocol Π_{ABA}.*
- *In an asynchronous network, almost-surely, the honest parties eventually output $y = f(x^{(1)}, \ldots, x^{(n)})$ where $x^{(j)} = 0$ for every $P_j \notin \mathcal{CS}$ and where $|\mathcal{CS}| \geq n - t_s$.*
- *Irrespective of the network type, the view of the adversary will be independent of the inputs of the honest parties in $\mathcal{CS}$.*

– *The protocol incurs a communication of* $\mathcal{O}(\frac{n^5}{\frac{t_a}{2}+1} c_M \log|\mathbb{F}| + n^7 \log|\mathbb{F}|)$ *bits from the honest parties and invokes* $\mathcal{O}(n^3)$ *instances of* Π_{BA}.

Proof. Consider a *synchronous* network with up to t_s corruptions. From the properties of $\Pi_{\mathsf{PreProcessing}}$ in the *synchronous* network, at time T_{TripGen}, the (honest) parties output c_M number of t_s-shared multiplication-triples, during the instance of $\Pi_{\mathsf{PreProcessing}}$. From the t_s-*correctness* property of Π_{ACS} in the *synchronous* network, at time T_{ACS}, the (honest) parties output a common subset of parties $\mathcal{CS}$ during the instance of Π_{ACS}, where *all honest* parties will be present in $\mathcal{CS}$ and where $|\mathcal{CS}| \geq n - t_s$. Moreover, corresponding to *every* $P_j \in \mathcal{CS}$, there will be some $x^{(j)}$ available with P_j (which will be the same as P_j's input for f for an *honest* P_j), such that $x^{(j)}$ will be t_s-shared. As $\mathcal{CS}$ will be known *publicly*, the parties take a default t_s-sharing of 0 on the behalf of the parties P_j outside $\mathcal{CS}$ by considering $x^{(j)} = 0$. Since $T_{\mathsf{ACS}} < T_{\mathsf{TripGen}}$, it follows that at time T_{TripGen}, the parties will hold t_s-sharing of c_M multiplication-triples and t_s-sharing of $x^{(1)}, \ldots, x^{(n)}$.

The circuit-evaluation will take $D_M \cdot \Delta$ time. This follows from the fact that linear gates are evaluated locally, while all the *independent* multiplication gates can be evaluated in parallel by running the corresponding instances of Π_{Beaver} in *parallel*, where each such instance requires Δ time. From the properties of Π_{Beaver} in the *synchronous* network, the multiplication-gates will be evaluated correctly and hence, during the output-computation phase, the parties will hold a t_s-sharing of y, where $y = f(x^{(1)}, \ldots, x^{(n)})$. From the properties of OEC, it will take Δ time for every party to reconstruct y. Hence, during the termination phase, *all* honest parties will send a `ready` message for y. Since there are at least $2t_s + 1$ honest parties, every honest party will then terminate with output y at time $T_{\mathsf{TripGen}} + (D_M + 2) \cdot \Delta$. By substituting the values of $T_{\mathsf{TripGen}}, T_{\mathsf{TripSh}}, T_{\mathsf{ACS}}, T_{\mathsf{VSS}}, T_{\mathsf{WPS}}, T_{\mathsf{BC}}, T_{\mathsf{BA}}, T_{\mathsf{SBA}}$ and T_{ABA} and by noting that all instances of Π_{BC} in Π_{CirEval} are invoked with $t = t_s$, we get that the parties terminate the protocol at time $(120n + D_M + 6k - 20) \cdot \Delta$, where k is the constant from Lemma 3.3, as determined by the underlying (existing) perfectly-secure ABA protocol Π_{ABA}.

The proof of the properties in an *asynchronous* network is similar as above, except that we now use the security properties of the building blocks $\Pi_{\mathsf{PreProcessing}}, \Pi_{\mathsf{ACS}}, \Pi_{\mathsf{Beaver}}$ and Π_{RecPriv} in the *asynchronous* network. During the termination phase, at most t_a *corrupt* parties can send `ready` messages for $y' \neq y$ and there will be at least $2t_s + 1$ honest parties, who eventually send `ready` messages for y. Moreover, if some honest party P_h terminates with output y, then every honest party eventually terminates the protocol with output y. This is because P_h must have received `ready` messages for y from at least $t_s + 1$ *honest* parties before termination, which are eventually delivered to *every* honest party. Consequently, irrespective of which stage of the protocol an honest party is in, every honest party (including P_h) eventually sends a `ready` message for y which are eventually delivered. As there are at least $2t_s + 1$ honest parties, this implies that every honest party eventually terminates with output y.

From the t_s-*privacy* property of Π_{ACS}, corresponding to every *honest* $P_j \in \mathcal{CS}$, the input $x^{(j)}$ will be random from the point of view of the adversary. Moreover, from the properties of $\Pi_{\mathsf{PreProcessing}}$, the multiplication-triples generated through $\Pi_{\mathsf{PreProcessing}}$ will be random from the point of view of the adversary. During the evaluation of linear gates, no interaction happens among the parties and hence, no additional information about the inputs of the honest parties is revealed. The same is true during the evaluation of multiplication-gates as well, which follows from the properties of Π_{Beaver}.

The communication complexity of the protocol follows from the communication complexity of $\Pi_{\mathsf{PreProcessing}}, \Pi_{\mathsf{ACS}}$ and Π_{Beaver}. □

8 Conclusion and Open Problems

In this work, we presented the *first* best-of-both-worlds perfectly-secure MPC protocol, which remains secure both in a synchronous as well as an asynchronous network. To design the protocol, we presented a best-of-both-worlds perfectly-secure VSS protocol and a best-of-both-worlds perfectly-secure BA protocol. Our work leaves the following interesting open problems.

- We could not prove whether the condition $3t_s + t_a < n$ is also necessary for any best-of-both-worlds perfectly-secure MPC protocol and conjecture that it is indeed the case.
- Our main focus in this work is on the *existence* of best-of-both-worlds perfectly-secure MPC protocols. Improving the efficiency of the protocol is left open for future work.

Acknowledgements: We would like to sincerely thank the anonymous reviewers of PODC 20222 for their excellent reviews on the preliminary version of this article, which got published as an extended abstract.

References

[1] Lecture 10: Consensus. https://www.mpi-inf.mpg.de/fileadmin/inf/d1/teaching/summer19/tkds/Lec10.pdf, 2019.

[2] I. Abraham, G. Asharov, and A. Yanai. Efficient Perfectly Secure Computation with Optimal Resilience. In *TCC*, volume 13043 of *Lecture Notes in Computer Science*, pages 66–96. Springer, 2021.

[3] I. Abraham, D. Dolev, and J. Y. Halpern. An Almost-Surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In *PODC*, pages 405–414. ACM, 2008.

[4] I. Abraham, D. Dolev, and G. Stern. Revisiting Asynchronous Fault Tolerant Computation with Optimal Resilience. In *PODC*, pages 139–148. ACM, 2020.

[5] A. Appan, A. Chandramouli, and A. Choudhury. Perfectly-Secure Synchronous MPC with Asynchronous Fallback Guarantees. In *PODC*, pages 92–102. ACM, 2022.

[6] G. Asharov and Y. Lindell. A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation. *J. Cryptology*, 30(1):58–151, 2017.

[7] L. Bangalore, A. Choudhury, and A. Patra. The Power of Shunning: Efficient Asynchronous Byzantine Agreement Revisited. *J. ACM*, 67(3):14:1–14:59, 2020.

[8] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.

[9] Z. Beerliová-Trubíniová and M. Hirt. Efficient Multi-party Computation with Dispute Control. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer, 2006.

[10] Z. Beerliová-Trubíniová and M. Hirt. Simple and Efficient Perfectly-Secure Asynchronous MPC. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2007.

[11] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-Secure MPC with Linear Communication Complexity. In *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2008.

[12] M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *PODC*, pages 27–30. ACM, 1983.

[13] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous Secure Computation. In *STOC*, pages 52–61. ACM, 1993.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*, pages 1–10. ACM, 1988.

[15] E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-Linear Unconditionally-Secure Multiparty Computation with a Dishonest Minority. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680. Springer, 2012.

[16] P. Berman, J. A. Garay, and K. J. Perry. Bit Optimal Distributed Consensus. In *Computer Science Research*, pages 313–322. Springer, 1992.

[17] E. Blum, J. Katz, and J. Loss. Synchronous Consensus with Optimal Asynchronous Fallback Guarantees. In *TCC*, volume 11891 of *Lecture Notes in Computer Science*, pages 131–150. Springer, 2019.

[18] E. Blum, J. Katz, and J. Loss. Tardigrade: An Atomic Broadcast Protocol for Arbitrary Network Conditions. In *ASIACRYPT*, volume 13091 of *Lecture Notes in Computer Science*, pages 547–572. Springer, 2021.

[19] E. Blum, C. L. Zhang, and J. Loss. Always Have a Backup Plan: Fully Secure Synchronous MPC with Asynchronous Fallback. In *CRYPTO*, volume 12171 of *Lecture Notes in Computer Science*, pages 707–731. Springer, 2020.

[20] G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC*, pages 154–162. ACM, 1984.

[21] R. Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.

[22] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC*, pages 42–51. ACM, 1993.

[23] A. Chandramouli, A. Choudhury, and A. Patra. A Survey on Perfectly-Secure Verifiable Secret-Sharing. *IACR Cryptol. ePrint Arch.*, page 445, 2021.

[24] D Chaum, C. Crépeau, and I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *STOC*, pages 11–19. ACM, 1988.

[25] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *FOCS*, pages 383–395. IEEE Computer Society, 1985.

[26] A. Choudhury and A. Patra. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Trans. Information Theory*, 63(1):428–468, 2017.

[27] R. Cramer and I. Damgård. *Multiparty Computation, an Introduction. Contemporary Cryptography.* Birkhåuser Basel, 2005.

[28] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *EUROCRYPT*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1999.

[29] I. Damgård and J. B. Nielsen. Scalable and Unconditionally Secure Multiparty Computation. In *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.

[30] G. Deligios, M. Hirt, and C. Liu-Zhang. Round-Efficient Byzantine Agreement and Multiparty Computation with Asynchronous Fallback. In *TCC*, volume 13042 of *Lecture Notes in Computer Science*, pages 623–653. Springer, 2021.

[31] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly Secure Message Transmission. *J. ACM*, 40(1):17–47, 1993.

[32] P. Feldman and S. Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[33] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.

[34] M. Fitzi, J. A. Garay, S. Gollakota, C. Pandu Rangan, and K. Srinathan. Round-Optimal and Efficient Verifiable Secret Sharing. In *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2006.

[35] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin. The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In *STOC*, pages 580–589. ACM, 2001.

[36] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fast-Track Multiparty Computations with Applications to Threshold Cryptography. In *PODC*, pages 101–111. ACM, 1998.

[37] D. Ghinea, C. Liu-Zhang, and R. Wattenhofer. Optimal Synchronous Approximate Agreement with Asynchronous Fallback. In *PODC*, pages 70–80. ACM, 2022.

[38] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press, 2004.

[39] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *STOC*, pages 218–229. ACM, 1987.

[40] V. Goyal, Y. Liu, and Y. Song. Communication-Efficient Unconditional MPC with Guaranteed Output Delivery. In *CRYPTO*, volume 11693 of *Lecture Notes in Computer Science*, pages 85–114. Springer, 2019.

[41] J. Katz, C. Y. Koo, and R. Kumaresan. Improving the Round Complexity of VSS in Point-to-point Networks. *Inf. Comput.*, 207(8):889–899, 2009.

[42] R. J. McEliece and D. V. Sarwate. On Sharing Secrets and Reed-Solomon Codes. *Commun. ACM*, 24(9):583–584, 1981.

[43] A. Mostéfaoui, H. Moumen, and M. Raynal. Signature-Free Asynchronous Binary Byzantine Consensus with t < n/3, O(n2) Messages, and O(1) Expected Time. *J. ACM*, 62(4):31:1–31:21, 2015.

[44] A. Patra, A. Choudhury, and C. Pandu Rangan. Efficient Asynchronous Verifiable Secret Sharing and Multiparty Computation. *J. Cryptology*, 28(1):49–109, 2015.

[45] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, 1980.

[46] M. O. Rabin. Randomized Byzantine Generals. In *FOCS*, pages 403–409. IEEE Computer Society, 1983.

[47] T. Rabin and M. Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *STOC*, pages 73–85. ACM, 1989.

[48] A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.

[49] A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS*, pages 160–164. IEEE Computer Society, 1982.

A Properties of the Existing (Asynchronous) Primitives

In this section we discuss the existing asynchronous primitives in detail.

A.1 Online Error-Correction (OEC)

The OEC procedure uses a Reed-Solomon (RS) error-correcting procedure $\mathsf{RSDec}(d, r, \mathcal{W})$, that takes as input a set $\mathcal{W}$ of distinct points on a d-degree polynomial and tries to output a d-degree polynomial, by correcting at most r *incorrect* points in $\mathcal{W}$. Coding theory [42] says that RS-Dec can correct up to r errors in $\mathcal{W}$ and correctly interpolate back the original polynomial if and only if $|\mathcal{W}| \geq d + 2r + 1$ holds. There are several efficient implementations of RSDec (for example, the algorithm of Berlekamp-Welch).

Suppose $\mathcal{P}' \subseteq \mathcal{P}$ contains at most t corrupt parties and let there exist some d-degree polynomial $q(\cdot)$, with every (honest) $P_i \in \mathcal{P}'$ having a point $q(\alpha_i)$. The goal is to make some *designated* party P_R reconstruct $q(\cdot)$. For this, each $P_i \in \mathcal{P}'$ sends $q(\alpha_i)$ to P_R, who then applies the OEC procedure OEC as described in Fig 12.

Protocol $\mathsf{OEC}(d, t, \mathcal{P}')$

Setting: There exists a subset of parties $\mathcal{P}'$ containing at most t corrupt parties, with each $P_i \in \mathcal{P}'$ having a point $q(\alpha_i)$ on some d-degree polynomial $q(\cdot)$. Every (honest) party in $\mathcal{P}'$ is supposed to send its respective point to P_R, who is designated to reconstruct $q(\cdot)$.

- **Output Computation** — For $r = 0, \ldots, t$, party P_R does the following in iteration r:
  - Let $\mathcal{W}$ denote the set of parties in $\mathcal{P}'$ from whom P_R has received the points and let $\mathcal{I}_r$ denote the points received from the parties in $\mathcal{W}$, when $\mathcal{W}$ contains exactly $d + t + 1 + r$ parties.
  - Wait until $\mathcal{W} \geq d + t + 1 + r$. Execute $\mathsf{RSDec}(d, r, \mathcal{I}_r)$ to get a d-degree polynomial, say $q_r(\cdot)$. If no polynomial is obtained, then skip the next step and proceed to the next iteration.
  - If for at least $d + t + 1$ values $v_i \in \mathcal{I}_r$ it holds that $q_r(\alpha_i) = v_i$, then output $q_r(\cdot)$. Otherwise, proceed to the next iteration.

Figure 12: The online error-correction procedure.

Lemma A.1 ([21]). *Let $\mathcal{P}' \subseteq \mathcal{P}$ contain at most t corrupt parties and let there exist some d-degree polynomial $q(\cdot)$, with every (honest) $P_i \in \mathcal{P}'$ having a point $q(\alpha_i)$. Then the OEC protocol prescribed in Fig 12 achieves the following for an honest P_R in the presence of up to t corruptions.*

- *If $d < (|\mathcal{P}'| - 2t)$, then in a synchronous network, it takes at most Δ time for P_R to output $q(\cdot)$. And in an asynchronous network, P_R eventually outputs $q(\cdot)$.*
- *If P_R obtains any output, then irrespective of the network type, the output polynomial is the same as $q(\cdot)$.*
- *The protocol incurs a communication of $\mathcal{O}(n \log |\mathbb{F}|)$ bits from the honest parties.*

Proof. The *communication complexity* follows from the fact that each party send its point to P_R. We next show that if P_R outputs a d-degree polynomial, say $q_r(\cdot)$, during the iteration number r, then $q_r(\cdot)$ is the same as $q(\cdot)$, *irrespective* of the network type. However, this easily follows from the fact that $q_r(\cdot)$ is consistent with $d + t + 1$ values from $\mathcal{I}_r$, out of which at least $d + 1$ values belong to the *honest* parties and thus, they lie on the polynomial $q(\cdot)$ as well. Furthermore, two *different* d-degree polynomials can have at most d distinct points in common.

We next prove the first property, assuming an *asynchronous* network. We first argue that an *honest P_R eventually* obtains some output, provided $d < (|\mathcal{P}'| - 2t)$. Let adversary control $\hat{r}$ parties in $\mathcal{P}'$, where $\hat{r} \leq t$. Assume that $\hat{r}_1$ corrupt parties send incorrect points to P_R and the remaining $\hat{r}_2 = \hat{r} - \hat{r}_1$ corrupt parties do not send anything at all. Then, consider iteration number $t - \hat{r}_2$. Since $\hat{r}_2$ parties never send any value, P_R will receive at least $d+t+1+t-\hat{r}_2$ distinct points on $q(\cdot)$, of which $\hat{r}_1$ could be corrupted. Since $|\mathcal{I}_{d+t+1+t-\hat{r}_2}| \geq d + 2\hat{r}_1 + 1$ holds, the algorithm RSDec will correct $\hat{r}_1$ errors and will return the polynomial $q(\cdot)$ during the iteration number $t - \hat{r}_2$. Therefore P_R will obtain an output, latest after $(t - \hat{r}_2)$ iterations.

The proof of the first property in the *synchronous* network is the same as above. In this case, it should be noted that the points of all honest parties reach P_R within Δ time. □

A.2 Bracha's Acast Protocol

Bracha's Acast protocol [20] tolerating $t < n/3$ corruptions is presented in Fig 13.

Protocol Π_{ACast}

1. If $P_i = \mathsf{S}$, then on input m, send $(\mathtt{init}, \mathsf{S}, m)$ to all the parties.
2. Upon receiving the message $(\mathtt{init}, \mathsf{S}, m)$ from S, send $(\mathtt{echo}, \mathsf{S}, m)$ to all the parties. Do not execute this step, more than once.
3. Upon receiving $(\mathtt{echo}, \mathsf{S}, m^\star)$ from $n - t$ parties, send $(\mathtt{ready}, \mathsf{S}, m^\star)$ to all the parties.
4. Upon receiving $(\mathtt{ready}, \mathsf{S}, m^\star)$ from $t + 1$ parties, send $(\mathtt{ready}, \mathsf{S}, m^\star)$ to all the parties.
5. Upon receiving $(\mathtt{ready}, \mathsf{S}, m^\star)$ from $n - t$ parties, output $m^\star$.

Figure 13: Bracha's Acast protocol. The above code is executed by every $P_i \in \mathcal{P}$ including the sender S.

We now prove the properties of the protocol Π_{ACast}.

Lemma 2.4. *Bracha's Acast protocol Π_{ACast} achieves the following in the presence of up to $t < n/3$ corruptions, where S has an input $m \in \{0,1\}^\ell$ for the protocol.*

- *Asynchronous Network:*
  - *(a) t-Liveness: If S is honest, then all honest parties eventually obtain some output.*
  - *(b) t-Validity: If S is honest, then every honest party with an output, outputs m.*
  - *(c) t-Consistency: If S is corrupt and some honest party outputs $m^\star$, then every honest party eventually outputs $m^\star$.*
- *Synchronous Network:*

- *(a) t-Liveness: If* S *is honest, then all honest parties obtain an output within time* 3Δ.
- *(b) t-Validity: If* S *is honest, then every honest party with an output, outputs* m.
- *(c) t-Consistency: If* S *is corrupt and some honest party outputs* $m^\star$ *at time* T*, then every honest* P_i *outputs* $m^\star$ *by the end of time* $T + 2\Delta$.

– *Irrespective of the network type,* $\mathcal{O}(n^2\ell)$ *bits are communicated by the honest parties.*

Proof. We first prove the properties assuming an *asynchronous* network with up to t corruptions. We start with the *validity* and *liveness* properties, for which we consider an *honest* S. We show that all honest parties eventually output m. This is because all honest parties complete steps $2-5$ in the protocol, even if the corrupt parties do not send their messages. This is because there are at least $n-t$ honest parties, whose messages are eventually selected for delivery. Moreover, the adversary may send at most t `echo` messages for m', where $m' \neq m$, on behalf of corrupt parties. Similarly, the adversary may send at most t `ready` messages for m', where $m' \neq m$, on behalf of corrupt parties. Consequently, no honest party ever generates a `ready` message for m', neither in step 3, nor in step 4. This is because $n - t > t$, as $t < n/3$.

For *consistency*, we consider a *corrupt* S and let P_h be an *honest* party, who outputs $m^\star$. We next show that all honest parties eventually outputs $m^\star$. Since P_h outputs $m^\star$, it implies that it receives $n-t$ `ready` messages for $m^\star$ during step 5 of the protocol. Let $\mathcal{H}$ be the set of *honest* parties whose `ready` messages are received by P_h during step 5. It is easy to see that $|\mathcal{H}| \geq t+1$. The `ready` messages of the parties in $\mathcal{H}$ are eventually delivered to every honest party and hence each honest party (including P_h) eventually executes step 4 and sends a `ready` message for $m^\star$. As there are at least $n-t$ honest parties, it follows that eventually $n-t$ `ready` messages for $m^\star$ are delivered to every honest party (irrespective of whether adversary sends all the required messages). This guarantees that all honest parties eventually obtain some output. To complete the proof, we show that this output is $m^\star$.

On contrary, let $P_{h'}$ be another honest party, different from P_h, who outputs $m^{\star\star} \neq m^\star$. This implies that $P_{h'}$ received `ready` messages for $m^{\star\star}$ from at least $t+1$ *honest* parties during step 5 of the protocol. Now from the protocol steps, it follow that an honest party generates a `ready` message for some potential m, only if it either receives $n-t$ `echo` messages for the m during step 3 or $t+1$ `ready` messages for m (one of which has to come from an honest party) during step 4. So all in all, in order that $n-t$ `ready` messages are eventually generated for some potential m during step 5, it must be the case that some honest party has to receive $n-t$ `echo` messages for m during step 2 and generate a `ready` message for m. Now since P_h receives $n-t$ `ready` messages for $m^\star$, some honest party must have received $n-t$ `echo` messages for $m^\star$, at most t of which could come from the corrupt parties. Similarly, since $P_{h'}$ receives $n-t$ `ready` messages for $m^{\star\star}$, some honest party must have received $n-t$ `echo` messages for $m^{\star\star}$. However, since $n-t > 2t$, it follows that in order that $n-t$ `echo` messages are produced for both $m^\star$ as well as $m^{\star\star}$, it must be the case that some honest party must have generated an `echo` message, both for $m^\star$, as well as $m^{\star\star}$ during step 2, which is impossible. This is because an honest party executes step 2 at most once and hence generates an `echo` message at most once.

The proofs of the properties in the *synchronous* network closely follow the proofs of the properties in the *asynchronous* network. If S is *honest*, then it will send the `init` message for m to all the parties, which will be delivered within time Δ. Consequently, every *honest* party will send an `echo` message for m to all the parties, which will be delivered within time 2Δ. Hence every *honest* party will send a `ready` message for m to all the parties, which will be delivered within time 3Δ. As there are at least $n-t$ honest parties, every honest party will receive `ready` messages for m from at least $n-t$ parties within time 3Δ and output m.

If S is *corrupt* and some honest party P_h outputs $m^\star$ at time T, then it implies that P_h has

received `ready` messages for $m^\star$ during step 5 of the protocol at time T from a set $\mathcal{H}$ of at least $t+1$ honest parties. These ready messages are guaranteed to be received by every other honest party within time $T+\Delta$. Consequently, every *honest* party who has not yet executed step 4 will do so and will send a `ready` message for $m^\star$ at time $T+\Delta$. Consequently, by the end of time $T+\Delta$, every honest party would have sent a `ready` message for $m^\star$ to every other honest party, which will be delivered within time $T+2\Delta$. Hence, every honest party will output $m^\star$ latest at time $T+2\Delta$.

The communication complexity (both in a synchronous as well as asynchronous network) simply follows from the fact that every party may need to send an `echo` and `ready` message for m to every other party. □

B An Overview of the Existing ABA Protocols [3, 7]

In this section, we give a very high level overview of the existing *t-perfectly-secure* ABA protocols of [3, 7]. Both these protocols are perfectly-secure and can tolerate up to $t < n/3$ corruptions. The protocols follow the standard framework of Rabin and Ben-Or [46, 12], which uses two building-blocks to get a BA protocol. The first building-block is a *voting* protocol (often called *gradecast* or *graded consensus* in the literature) and which is a deterministic protocol. The second building-block is a *coin-flipping* protocol which is a randomized protocol. In the sequel, we review these building blocks and discuss how they are "combined" to get an ABA protocol. While presenting these building-blocks, unless it is explicitly stated, we assume an *asynchronous* network. Also, for simplicity, we present these building-blocks *without* specifying any *termination* criteria and hence, the parties may keep on running these building-blocks (as well as the ABA protocol) even after obtaining an output.[12]

B.1 The Voting Protocol

Informally, the voting protocol does "whatever can be done deterministically" to reach agreement. In a voting protocol, every party has a single bit as input. The protocol tries to find out whether there is a detectable majority for some value among the inputs of the parties. In the protocol, each party's output can have *five* different forms:

- For $\sigma \in \{0,1\}$, the output $(\sigma, 2)$ stands for "overwhelming majority for σ";
- For $\sigma \in \{0,1\}$, the output $(\sigma, 1)$ stands for "distinct majority for σ";
- The output $(\Lambda, 0)$ stands for "non-distinct majority".

The protocol code of the voting protocol taken from [21] is presented in Fig 14.

Protocol Π_{Vote}

- On having the input x_i, Acast $(\texttt{input}, P_i, x_i)$.
- Create a dynamic set $\mathcal{X}_i$ which is initialized to $\emptyset$. Add (P_j, x_j) to $\mathcal{X}_i$ if $(\texttt{input}, P_j, x_j)$ is received from the Acast of P_j.
- Wait until $|\mathcal{X}_i| = n - t$. Then assign $X_i = \mathcal{X}_i$, set a_i to the majority bit among $\{x_j \mid (P_j, x_j) \in X_i\}$. Acast $(\texttt{vote}, P_i, X_i, a_i)$.
- Create a dynamic set $\mathcal{Y}_i$, which is initialized to $\emptyset$. Add (P_j, X_j, a_j) to $\mathcal{Y}_i$ if $(\texttt{vote}, P_j, X_j, a_j)$ is received from the Acast of P_j, $X_j \subseteq \mathcal{X}_i$, and a_j is the majority bit of X_j.
- Wait until $|\mathcal{Y}_i| = n - t$. Then assign $Y_i = \mathcal{Y}_i$, set b_i to the majority bit among $\{a_j \mid (P_j, X_j, a_j) \in Y_i\}$ and Acast $(\texttt{re-vote}, P_i, Y_i, b_i)$.
- Create a set Z_i, which is initialized to $\emptyset$. Add (P_j, Y_j, b_j) to Z_i if $(\texttt{re-vote}, P_j, Y_j, b_j)$ is received

[12] Recall that we do not put any termination criteria for any of our sub-protocols, as the termination of the MPC protocol will automatically ensure that all the underlying sub-protocols also get terminated.

from the Acast of P_j, $Y_j \subseteq \mathcal{Y}_i$, and b_j is the majority bit of Y_j.
- Wait until $|Z_i| = n - t$. Then compute the output as follows.
  - If all the parties $P_j \in Y_i$ have the same `vote` $a_j = \sigma$, then output $(\sigma, 2)$.
  - Else if all the parties $P_j \in Z_i$ have the same `re-vote` $b_j = \sigma$, then output $(\sigma, 1)$.
  - Else output $(\Lambda, 0)$.

Figure 14: The vote protocol. The above code is executed by every $P_i \in \mathcal{P}$.

The properties of the voting protocol are stated in Lemma B.1. While these properties hold in an *asynchronous* network, it automatically implies that they hold even for a *synchronous* network. We refer the readers to [21, 7] for the proof of these properties.

Lemma B.1 ([21, 7])**.** *Protocol Π_{Vote} achieves the following properties, both in the synchronous as well as asynchronous network, if the adversary corrupts up to $t < n/3$ parties, where all the parties participate with an input bit.*
- *If each honest party has the same input σ, then each honest party outputs $(\sigma, 2)$;*
- *If some honest party outputs $(\sigma, 2)$, then every other honest party outputs either $(\sigma, 2)$ or $(\sigma, 1)$;*
- *If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$ then each honest party outputs either $(\sigma, 1)$ or $(\Lambda, 0)$.*
- *The protocol incurs a communication of $\mathcal{O}(n^3)$ bits from the honest parties.*

An additional property which protocol Π_{Vote} achieves in a *synchronous* network is that all *honest* parties will have their output by the end of time 9Δ. Intuitively, this is because the protocol involves three different "phases" of Acast, each of which will produce an output within time 3Δ for *honest* sender parties in a *synchronous* network. Moreover, from Lemma B.1, this output will be $(\sigma, 2)$, if all the *honest* parties have the same input σ. We will require this property later while claiming the properties of the resultant ABA protocol in a *synchronous* network. Hence, we prove this property.

Lemma B.2. *If the network is synchronous and if the adversary corrupts up to $t < n/3$ parties, then in protocol Π_{Vote}, all honest parties obtain their output within time 9Δ. Moreover, the output will be $(\sigma, 2)$, if all the honest parties have the same input σ.*

Proof. Consider an arbitrary *honest* P_j. Party P_j will Acast its input x_j and from the *t-liveness* and *t-validity* properties of Acast in the *synchronous* network, every honest party will receive the output x_j, from the corresponding Acast instance within time 3Δ. As there are at least $n-t$ *honest* parties, it implies that every *honest* P_i will obtain a set X_i of size $n - t$ within time 3Δ. Hence each *honest* P_i will Acast a $(\texttt{vote}, P_i, X_i, a_i)$ message latest at time 3Δ and every honest party receives this message from the corresponding Acast instance within time 6Δ. We also note that if there is a *corrupt* P_j such that (P_j, x_j) is included by an *honest* P_i in its set X_i when P_i Acasts X_i, then from the *t-consistency* property of Acast in the *synchronous* network, every *honest* party P_k will include (P_j, x_j) in its set $\mathcal{X}_k$, latest by time 5Δ. This further implies that upon receiving the message $(\texttt{vote}, P_i, X_i, a_i)$ from the Acast of any *honest* P_i, all *honest* parties P_k will be able to verify this message and include (P_i, X_i, a_i) in their respective $\mathcal{Y}_k$ sets within time 6Δ.

As there are at least $n - t$ honest parties P_i whose `vote` messages are received and verified by all honest parties P_k within time 6Δ, it follows that every honest party Acasts a `re-vote` message, latest at time 6Δ, which is received by every honest party within time 9Δ. Moreover, as argued for the case of `vote` messages, every *honest* party will be able to verify these `re-vote` messages and include in their respective Z_i within time 9Δ. Since there are at least $n - t$ honest parties, it

follows that the Z_i sets of every honest party will attain the size of $n-t$ within time 9Δ and hence every honest party will obtain an output, latest at time 9Δ.

If all the honest parties have the same input σ, then there will be at most t *corrupt* parties who may Acast $1-\sigma$. Hence *every* party (both honest as well as corrupt) will send a `vote` message only for σ.[13] Consequently, every honest party will output $(\sigma, 2)$. □

B.2 Coin-Flipping Protocol

The *coin-flipping* protocol denoted by Π_{CoinFlip} (also called as the *common-coin* protocol) is an n-party (asynchronous) protocol, where the parties have local random inputs and the protocol outputs a bit for all the parties. The protocol achieves the following properties in an *asynchronous* (and hence *synchronous*) network in the presence of any $t < n/3$ corruptions.

- In an *asynchronous* network, all honest parties eventually obtain an output, while in a *synchronous* network, the honest parties obtain an output within some fixed time $c \cdot \Delta$, where c is a *publicly-known* constant.
- One of the following holds:
  - If no party deviates from the protocol, then with probability *at least* p, the output bits of all the honest parties are same. The probability p where $p < 1$ is often called as the *success-probability* of the protocol and is a parameter of the protocol.
  - Else, all honest parties will have the same output bit with probability *less than* p. But in this case, the protocol allows some *honest* party(ies) to *locally* identify and shun a (subset) of corrupt party(ies) from any future communication. Namely, the protocol *locally* outputs ordered pairs of the form (P_i, P_j), where P_i is some *honest* party and P_j is some *corrupt* party, such that P_i identifies P_j as a corrupt party and does not consider any communication from P_j for the rest of the protocol execution. Such pairs are called as *local-conflicts*. We stress that the local-conflicts are identified only *locally*. For instance, if an *honest* P_i has shunned a *corrupt* P_j during an instance of the coin-flipping protocol, then it is *not* necessary that every other *honest* party P_k also shuns P_j during the same instance, as P_j may decide to behave "honestly" towards P_k.
    The coin-flipping protocol of [3] guarantees that at least one *new* local-conflict is identified if, during an instance of the coin-flipping protocol, the parties obtain the same output bit with probability less than p. On the other hand, the coin-flipping protocol of [7] guarantees that $\Theta(n)$ number of *new* local-conflicts are identified, if the parties obtain the same output bit with probability less than p.

Protocol Π_{CoinFlip} is designed using a *weaker* variant of perfectly-secure AVSS called *shunning* AVSS (SAVSS), introduced in [3]. The SAVSS primitive is weaker than AVSS in the following aspects:

- It is *not* guaranteed that *every* honest party obtains a point on D's sharing-polynomial (and hence a share of D's secret), even if D is *honest*;
- If D is *corrupt*, then it may not participate with a t-degree polynomial and hence, the underlying shared value could be $\perp$, which is different from every element of $\mathbb{F}$;
- Irrespective of D, depending upon the behaviour of the corrupt parties, the honest parties later may either reconstruct the same secret as shared by D or an all-together different value. However, in the latter case, the protocol ensures that at least one *new* local-conflict is identified.

In [3], a perfectly-secure SAVSS protocol is designed with $t < n/3$. By executing n^2 instances of this protocol in *parallel* using the framework of [32, 21], a coin-flipping protocol is presented in [3],

[13] If a *corrupt* party P_j sends a `vote` message for $1-\sigma$, then it will never be accepted and no honest party P_i will ever include $(P_j, X_j, 1-\sigma)$ in its $\mathcal{Y}_i$ set. This is because $1-\sigma$ will not be the majority among the inputs of the honest parties in X_j.

where the success-probability p is $\frac{1}{4}$. The protocol incurs a communication of $\mathcal{O}(\text{poly}(n)\log|\mathbb{F}|)$ bits from the honest parties.

The coin-flipping protocol of [7] also uses the same framework of [32, 21], but substitutes the SAVSS of [3] with a "better" and more efficient SAVSS with $t < n/3$. Their SAVSS ensures that $\Theta(n)$ number of *new* local-conflicts are identified, if the value reconstructed by the parties is *different* from the one shared by D. The success-probability p remains $\frac{1}{4}$ and the communication complexity of the protocol is $\mathcal{O}(\text{poly}(n)\log|\mathbb{F}|)$ bits.

B.3 Vote + Coin-Flipping ⇒ ABA

We now show how to "combine" protocols Π_{Vote} and Π_{CoinFlip} to get the protocol Π_{ABA} (see Fig 15). The current description of Π_{ABA} is taken from [17]. The protocol consists of several iterations, where each iteration consists of two instances of Π_{Vote} protocol and one instance of Π_{CoinFlip}, which are carefully "stitched" together.

In the first instance of Π_{Vote}, the parties participate with their "current input", which is initialized to their respective bits for ABA in the first iteration. Then, independent of the output received from the instance of Π_{Vote}, the parties participate in an instance of Π_{CoinFlip}. Next, the parties decide their respective inputs for the second instance of Π_{Vote} protocol, based on the output they received from the first instance. If a party has received the *highest* grade (namely 2) during the first instance of Π_{Vote}, then the party *continues* with the bit received from that Π_{Vote} instance for the second Π_{Vote} instance. Otherwise, the party *switches* to the output received from Π_{CoinFlip}. The output from the second instance of Π_{Vote} is then set as the modified input for the next iteration, if it is obtained with a grade higher than 0. Otherwise, the output of Π_{CoinFlip} is taken as the modified input for the next iteration.

If during any iteration a party obtains the highest grade from the second instance of Π_{Vote}, then it indicates this publicly by sending a `ready` message to every party, along with the bit received. The `ready` message is an indication for the others about the "readiness" of the sender party to consider the corresponding bit as the output. Finally, once a party receives this readiness indication for a common bit b from at least $2t+1$ parties, then that bit is taken as the output. To ensure that every other party also outputs the same bit, a party upon receiving the `ready` message for a common bit from at least $t+1$ honest parties, itself sends a `ready` message for the same bit (if it has not done so earlier).

The idea behind the protocol is the following. In the protocol there can be two cases. The *first* case is when all the honest parties start with the *same* input bit, say b. Then, they will obtain the output b from all the instances of Π_{Vote} protocol in all the iterations and the outputs from Π_{CoinFlip} will be never considered. Consequently, each honest party will eventually send a `ready` message for b. Moreover, there can be at most t corrupt parties who may send a `ready` message for $1-b$ and hence no honest party ever sends a `ready` message for $1-b$. Hence, each honest party eventually outputs b.

The *second* case is when the honest parties start the protocol with *different* input bits. In this case, the protocol tries to take the help of Π_{CoinFlip} to ensure that all honest parties reach an iteration with a common input bit for that iteration. Once such an iteration is reached, this *second* case gets "transformed" to the *first* case and hence all honest parties will eventually output that common bit. In more detail, in each iteration k, it will be ensured that either every honest party have the same input bit for the second instance of Π_{Vote} with probability at least $p \cdot \frac{1}{2}$ or else certain number of new local-conflicts are identified.[14] This is because the input for second instance

[14]The number of local-conflicts identified will depend upon the Π_{CoinFlip} protocol: while the Π_{CoinFlip} protocol of [3] will ensure that at least 1 new local-conflict is identified, the Π_{CoinFlip} protocol of [7] will ensure that $\Theta(n)$ number of

of Π_{Vote} is either the output bit of the first instance of Π_{Vote} or the output of Π_{CoinFlip}, both of which are *independent* of each other. Hence if the output of Π_{CoinFlip} is same for all the parties with probability p, then with probability $p \cdot \frac{1}{2}$, this bit will be the same as output bit from the first instance of Π_{Vote}. If in any iteration k, it is *guaranteed* that all honest parties have the same inputs for the second instance of Π_{Vote}, then the parties will obtain a common output and with highest grade from the second instance of Π_{Vote}. And then from the next iteration onward, all parties will stick to that common bit and eventually output that common bit.

One can show that it requires $\mathcal{O}(\text{poly}(n))$ number of iterations in *expectation* before a "good" iteration is reached, where an iteration is considered good, if it is *guaranteed* that all honest parties have the same input for the second instance of Π_{Vote}. Intuitively, this is because there can be $\mathcal{O}(\text{poly}(n))$ number of "bad" iterations in which the honest parties may have different outputs from the corresponding instances of Π_{CoinFlip}. This follows from the fact that the corrupt parties may deviate from the protocol instructions during the instances of Π_{CoinFlip}. There can be at most $t(n-t)$ local-conflicts which may occur (t potentially corrupt parties getting in conflict with $n-t$ honest parties) *overall* during various "failed" instances of Π_{CoinFlip} (where a failed instance means that different honest parties obtain different outputs) and only after all these local-conflicts are identified, the parties may start witnessing "clean" instances of Π_{CoinFlip} where all honest parties shun communication from all corrupt parties and where it is ensured that all honest parties obtain the same output bit with probability p. Now depending upon the number of new local-conflicts which are revealed from a single failed instance of Π_{CoinFlip}, the parties may witness $\mathcal{O}(\text{poly}(n))$ number of bad iterations.[15] Now, once all the bad iterations are over and all potential local-conflicts are identified, in each subsequent iteration, all honest parties will then have the same output from Π_{CoinFlip} (and hence, same input for the second instance of Π_{Vote}) with probability at least $\frac{p}{2}$. Consequently, if p is a *constant*, then it will take $\Theta(1)$ expected number of such iterations before the parties reach a good iteration where it is guaranteed that all honest parties have the same inputs for the second instance of Π_{Vote}.[16]

Protocol Π_{ABA}

Input: Party P_i has the bit b_i as input for the ABA protocol.

- **Initialization**: Set $b = b_i$, $\mathsf{committed} = \mathsf{false}$ and $k = 1$. Then do the following.
  1. Participate in an instance of Π_{Vote} protocol with input b.
  2. Once an output (b, g) is received from the instance of Π_{Vote}, participate in an instance of Π_{CoinFlip}. Let Coin_k denote the output received from Π_{CoinFlip}.
  3. If $g < 2$, then set $b = \mathsf{Coin}_k$.
  4. Participate in an instance of Π_{Vote} protocol with input b and let (b', g') be the output received. If $g' > 0$, then set $b = b'$.
  5. If $g' = 2$ and $\mathsf{committed} = \mathsf{false}$, then set $\mathsf{committed} = \mathsf{true}$ and send $(\texttt{ready}, b)$ to all the parties.
  6. Set $k = k + 1$ and repeat from 1.
- **Output Computation**:
  - If $(\texttt{ready}, b)$ is received from at least $t + 1$ parties, then send $(\texttt{ready}, b)$ to all the parties.
  - If $(\texttt{ready}, b)$ is received from at least $2t + 1$ parties, then output b.

Figure 15: The ABA protocol from Π_{Vote} and Π_{CoinFlip}. The above code is executed by every $P_i \in \mathcal{P}$.

new local-conflicts are identified.

[15] Since each failed instance of the Π_{CoinFlip} protocol of [3] may reveal only 1 new local-conflict, the number of bad iterations could be $\mathcal{O}(n^2)$. On the other hand, each failed instance of the Π_{CoinFlip} protocol of [7] reveals $\Theta(n)$ new local-conflicts and hence there can be $\mathcal{O}(n)$ number of bad iterations.

[16] One can show that if one sets $p = \frac{1}{4}$ as done in [22, 3, 7], then it takes expected 16 iterations *after* all the local-conflicts are identified to reach a good iteration.

Lemma 3.3 now follows easily from the above discussion. Let $c \cdot \Delta$ be the time within which the protocol Π_{CoinFlip} generates output for the honest parties in a *synchronous* network, where c is some publicly-known constant. Note that c is determined by the underlying SAVSS protocol and is different for the SAVSS protocols of [3] and [7]. If all honest parties have the same input b in a *synchronous* network, then at the end of the first iteration itself, every party will send a `ready` message for b to every other party. Consequently, in this case, all honest parties will obtain their output within time $(c+18+1) \cdot \Delta$. This is because each instance of Π_{Vote} during the first iteration will take at most 9Δ time to produce output, while the instance of Π_{CoinFlip} will take at most $c \cdot \Delta$ time. Additionally, Δ time will be taken by each party to send a `ready` message for b to every other party. Consequently, T_{ABA} will be $(c + 19) \cdot \Delta$.