Unconditionally Secure Asynchronous Multiparty Computation with Linear Communication Complexity

Ashish Choudhury* Martin Hirt† Arpita Patra‡

Abstract

We present two unconditionally secure asynchronous multiparty computation (AMPC) protocols among n parties with an amortized communication complexity of $\mathcal{O}(n)$ field elements per multiplication gate and which can tolerate a computationally unbounded active adversary corrupting $t < n/4$ parties. These are the first AMPC protocols with linear communication complexity per multiplication gate. Our first protocol is *statistically* secure in a completely asynchronous setting and improves on the previous best AMPC protocol in the same setting by a factor of $\Theta(n)$. Our second protocol is *perfectly* secure in a *hybrid* setting, where one round of communication is assumed to be synchronous and improves on the previous best AMPC protocol in the hybrid setting by a factor of $\Theta(n^2)$.

The central contribution common to both the protocols is a new, simple and communication efficient, albeit natural framework for the preprocessing (offline) phase that is used to generate sharings of random multiplication triples, to be used later for the circuit evaluation. The framework is built on two new components, both of which are instantiated robustly: the first component allows the parties to verifiably share random multiplication triples. The second component allows the parties to securely extract sharings of random multiplication triples from a set of sharings of multiplication triples, verifiably shared by individual parties. Our framework is simple and does not involve either of the existing somewhat complex, but popular techniques, namely *player elimination* and *dispute control*, used in the preprocessing phase of most of the existing protocols. The framework is of independent interest and can be adapted to other MPC scenarios to improve the overall round complexity.

*Department of Computer Science, University of Bristol, UK, Email: Ashish.Choudhary@bristol.ac.uk.

†Department of Computer Science, ETH Zurich, Switzerland, Email: hirt@inf.ethz.ch

‡Department of Computer Science, University of Bristol, UK, Email: arpitapatra10@gmail.com.

1 Introduction

Threshold unconditionally secure multiparty computation (MPC) enables a set of n mutually distrusting parties to jointly and securely compute a publicly known function f over some finite field $\mathbb{F}$. The distrust among the parties is modelled by a computationally unbounded centralized adversary Adv, who can *actively* corrupt any t out of the n parties. The first generic but inefficient unconditionally secure protocols were proposed in [8, 14, 32, 2]. In a general MPC protocol, the function f is usually expressed as an arithmetic circuit over $\mathbb{F}$ and then the protocol evaluates each gate in the circuit in a shared/distributed fashion. Usually, the number of multiplication gates is significantly larger than the other types of gates and the evaluation of a multiplication gate requires more communication than the other types of gate. The focus therefore is rightfully placed on measuring the communication complexity (namely the total number of field elements communicated) required to evaluate the multiplication gates.

In the recent past, several efficient unconditionally secure MPC protocols have been proposed [23, 22, 3, 18, 5, 10]. The state of the art unconditionally secure MPC protocols have *linear* (i.e. $\mathcal{O}(n)$ field elements) *amortized* communication complexity per multiplication gate for both the *perfect* setting [5] (i.e. error-free) as well as for the *statistical* setting [10] (where a negligible error is allowed). The amortized communication complexity is derived under the assumption that the circuit is large enough so that the terms that are independent of the circuit size can be ignored [10]. Moreover, these protocols have the *optimal resilience* of $t < n/3$ and $t < n/2$ respectively. The significance of linear communication complexity roots from the fact that the amortized communication done by *each* party for the evaluation of a multiplication gate is *independent* of n. This makes the protocol "scalable" in the sense that the communication done by an individual party does not grow with the number of parties in the system. In fact, even the best known unconditionally secure MPC protocol in the *passive* security setting [18] does not do better than amortized communication complexity of $\mathcal{O}(n)$ field elements per multiplication gate. We note that if one is willing to reduce the resilience t from the optimal resilience by a constant fraction, then by using techniques like packed secret sharing [21] and committee election [12], one can achieve additional efficiency, as shown in [17, 16]. However, the resultant protocols are quiet complex.

Our Motivation: The results discussed above are designed to work in the synchronous network setting, where the delay of every message in the network is bounded by a known constant. However, it is well-known that such networks do not model well the real-life networks like the Internet. Consequently, the asynchronous network model has been proposed [7], where there are no timing assumptions and the messages can be arbitrarily delayed. Protocols in the asynchronous model are much more involved than their synchronous counterparts due to the following general phenomena, which is impossible to avoid in a complete asynchronous setting: if a party does not receive an expected message, then it does not know whether the sender is corrupted (and did not send the message at all) or the message is just delayed in the network. Thus, at any "stage" of an asynchronous protocol, no party can afford to listen the communication from all the n parties, as the wait may turn out be endless and so the communication from t (potentially honest) parties has to be ignored; see [13] for an introduction to the asynchronous protocols. Perfectly secure (i.e. error-free) asynchronous MPC (AMPC) is possible if and only if $t < n/4$ [7], while statistically secure AMPC (involving negligible error) is possible if and only if $t < n/3$ [9].

Unlike the synchronous setting, unconditional AMPC protocols have received less attention, probably due to its complexity. Unconditionally secure AMPC protocols are presented in [7, 9, 34, 31, 4, 29, 28]. The best known (perfectly secure) AMPC protocol is reported in [28]. The protocol has resilience $t < n/4$ and amortized communication complexity of $\mathcal{O}(n^2)$ field elements per multiplication gate. Designing AMPC protocols with linear communication complexity per multiplication gate is the focus of this paper.

Our Results: We present two AMPC protocols with (amortized) communication complexity of $\mathcal{O}(n)$ field elements per multiplication gate and with resilience $t < n/4$. The first protocol is statistically secure and works in a completely asynchronous setting. Though non-optimally resilient, the protocol is the first AMPC protocol with linear communication complexity per multiplication gate. The protocol does not use tools like packed secret sharing and committee election, which are quiet complex to apply even in the synchronous setting [17, 16] and which are not known how to be applied in the asynchronous setting. Instead, we use simple techniques and follow

a simple, but new framework (more on this soon). Our second protocol trades the network model to gain perfect security with optimal resilience of $t < n/4$. The protocol is designed in a *hybrid* setting, that allows a single synchronous round at the beginning, followed by a fully asynchronous setting. The hybrid setting was exploited earlier in [24, 4, 6] to enforce "input provision", i.e. to consider the inputs of all the n parties for the computation, which is otherwise impossible in a complete asynchronous setting. The best known AMPC protocol in the hybrid setting was proposed in [4] with perfect security, resilience $t < n/4$ and communication complexity $\mathcal{O}(n^3)$ per multiplication gate. Thus, our protocol significantly improves over the hybrid model protocol of [4].

1.1 Our Results in More Details

We follow the well-known "offline-online" paradigm used in almost all the recent MPC protocols [3, 4, 18, 5, 10], where the offline (preprocessing) phase produces t-sharing of c_M random *multiplication triples* $\{(a^{(i)}, b^{(i)}, c^{(i)})\}_{i\in[c_M]}$ unknown[1] to Adv, where $c^{(i)} = a^{(i)}b^{(i)}$ and later the online phase uses the shared triples for the shared evaluation of the multiplication gates using Beaver's circuit randomization technique [2] (see Sec. 3.2). The efficiency of the protocols in this paradigm is thus reduced to the problem of efficient implementation of the offline phase. Our proposal for the offline phase takes a completely different approach compared to the existing ones and significantly outperforms them in terms of the efficiency and simplicity; the details follow.

The traditional way of generating the shared triples $\{(a^{(i)}, b^{(i)}, c^{(i)})\}_{i\in[c_M]}$ is the following: first the individual parties are asked to t-share random *pairs* of values on which a "randomness extraction" algorithm (such as the one based on Vandermonde matrix [18] or the one based on Hyper-invertible matrix [5] or the simplest of all, namely to add all the pairs) is applied to generate t-sharing of "truly" random pairs $\{a^{(i)}, b^{(i)}\}_{i\in[c_M]}$ and then known multiplication protocols are invoked to compute t-sharing of $\{c^{(i)}\}_{i\in[c_M]}$. Instead, we find it a more natural approach to ask the parties to "directly" share random multiplication triples and then "extract" shared random multiplication triples unknown to Adv from the triples shared by the individual parties. This leads to a communication efficient, simple and more natural "framework" to generate the triples. The framework is built with the following two components:

- The first component allows a party P_i to *robustly* and "verifiably" t-share $\Theta(n)$ random multiplication triples with a private communication of $\mathcal{O}(n^2)$ and thus requires linear (i.e. $\mathcal{O}(n)$) "overhead". The verifiability ensures that the shared triples are indeed multiplication triples. If P_i is *honest* then the shared triples remain private from Adv. Such shared triples generated by the individual parties are referred as "local" triples. Thus far, we are unaware of any existing protocol (even in the synchronous setting) meeting these requirements.

- The second component allows the parties to extract sharings of $\Theta(n)$ random multiplication triples unknown to Adv from a set of $3t + 1$ local shared multiplication triples with a private communication of $\mathcal{O}(n^2)$ (and thus requires $\mathcal{O}(n)$ "overhead"), given that at least $2t+1$ out of the $3t+1$ local sharings are generated by the honest parties (and hence are random and private). While it is known how to extract shared random values from a set of shared random and non-random values, our protocol is the first of its kind to extract shared random multiplication triples from a set of shared random and non-random multiplication triples[2].

We instantiate both the components robustly. Interestingly, we do not employ either the *player elimination* [23, 5] or the *dispute control* [3, 18, 10] technique, the two powerful but somewhat complex techniques, used in the preprocessing phase of most of the recent synchronous unconditional MPC protocols. We note that our framework can be adapted to any *honest majority* setting to improve the round complexity of the preprocessing phase. For example, when compiled in a completely synchronous network, our framework can lead to a very simple preprocessing phase which will reduce the overall round round complexity of the existing round efficient MPC protocols of [25, 26] in point-to-point networks; however, providing the exact details is out of scope of the current article.

For the first component, we present two protocols for verifiably sharing multiplication triples. The first protocol is a completely asynchronous protocol and probabilistically verifies the correctness of the shared triples, leading to

[1] A value v is d-shared if there exists a random polynomial $p(\cdot)$ of degree at most d with $v(0) = d$ and every honest party holds a distinct point on $v(\cdot)$. Here c_M denotes the number of multiplication gates in the circuit. By $[X]$, we refer to the set $\{1, \ldots, X\}$.

[2] It is not hard to note that the existing (information theoretic) techniques for extracting random values from a set of random and non-random values cannot be deployed to extract random multiplication triples from a set of random and non-random multiplication triples.

our statistical AMPC protocol. The second protocol verifies the shared triples in an error-free fashion in a hybrid setting, leading to our perfect AMPC protocol in the hybrid setting. For the second component, we present a triple extraction protocol. Central to all these protocols lie the following two simple and interesting protocols:

Verifiably Sharing $\Theta(n)$ Values with $\mathcal{O}(n^2)$ Communication (Section 4): Our triple sharing protocols use a *robust* secret sharing protocol, which allows a *dealer* D to "verifiably" t-share $\Theta(n)$ values with $\mathcal{O}(n^2)$ communication, implying linear "overhead". The protocol is obtained by modifying the perfectly secure *asynchronous verifiable secret sharing* (AVSS) protocol of [29, 28] that allows D to generate a single $2t$-sharing. To the best of our knowledge, we are unaware of any robust secret sharing protocol (with $t < n/4$) having linear overhead.

Transforming Independent Triples to Co-related Triples (Section 5): A common protocol that is used in the *error-free triple sharing* protocol and also in the *triple extraction* protocol is the following: the protocol takes as input $3t+1$ t-shared triples, say $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i\in[3t+1]}$ and outputs $3t+1$ t-shared triples, say $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i\in[3t+1]}$ such that the following holds: **(a)** there exists polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ of degree at most $\frac{3t}{2}, \frac{3t}{2}$ and $3t$ respectively, where $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds for $3t+1$ distinct α_is; **(b)** the output triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a *multiplication triple* if and only if the input triple $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication triple, which implies that the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true if and only if all the input triples are multiplication triples; and **(c)** if Adv knows at most t' input triples and if $t' \leq \frac{3t}{2}$, then it learns at most t' output triples and thus t' distinct values on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, implying $\frac{3t}{2}+1-t'$ "degree of freedom" in the polynomials. The protocol for the "transformation" from $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i\in[3t+1]}$ to $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i\in[3t+1]}$ is inherited from the batch verification protocol of [10], where the *only* goal was to *probabilistically* check whether a set of input triples are multiplication triples. We use the protocol in *two* different contexts by "post-processing" the output of the transformation. Namely, in our error-free triple sharing protocol, we use the transformation protocol to check in an *error-free manner* whether a set of triples shared by a dealer are multiplication triples and if so, then extract a set of multiplication triples, such that if the dealer is *honest*, then the extracted triples are random and unknown to Adv. On the other hand, in our triple extraction protocol, we use the transformation protocol on a set of local multiplication triples shared by individual parties and extract random multiplication triples unknown to Adv. We next elaborate a little more on our triple sharing protocols and the triple extraction protocol.

Verifiably Sharing Multiplication Triples (Section 6.1 and 8): Our first triple sharing protocol resorts to the so called "sacrificing trick" [15, 19] that deploys two triples and sacrifices the privacy of the second triple (if it was private before) to probabilistically confirm if the first triple is a multiplication triple by leaking nothing more than the conviction that it is a multiplication triple. Specifically, if a dealer D has shared two independent triples (x, y, z) and (f, g, h) and claims them to be multiplication triples, then the relation $r(z - xy) \stackrel{?}{=} (h - fg)$ must hold for any random challenge r selected from $\mathbb{F}$, except with probability at most $\frac{2}{|\mathbb{F}|}$. The relation can be checked publicly by making $\rho = rx - f$ and $\sigma = y - g$ public, followed by making $\gamma = rz - h - \sigma f - \rho g - \rho\sigma$ public. While the random r that is to be chosen once D shares the triples, ensures verifiability against a *corrupted* D with very high probability, the publicly revealed values leak no further information about (x, y, z) when D is *honest* due to the guarantee that (f, g, h) is random and unknown to Adv.

Our second triple sharing protocol first asks D to share $3t+1$ random multiplication triples $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i\in[3t+1]}$ and then applies the transformation protocol to compute set of shared triples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i\in[3t+1]}$. We then propose a technique to verify the relation $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ in an error-free fashion with the help of the first synchronous round in a way that leaks at most t values (at the same evaluation points) on the three polynomials; this implies $\frac{t}{2}$ "degree of freedom" in the polynomials for an *honest* D. If the verification passes, then $\frac{t}{2}$ output shared multiplication triples are computed as $\{\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \mathbf{c}^{(i)}\}_{i\in[\frac{t}{2}]}$ where $\mathbf{a}^{(i)} = \mathsf{X}(\beta_i), \mathbf{b}^{(i)} = \mathsf{Y}(\beta_i)$ and $\mathbf{c}^{(i)} = \mathsf{Z}(\beta_i)$ for $\{\beta_i\}_{i\in[\frac{t}{2}]}$, which are distinct from $\{\alpha_i\}_{i\in[n]}$. It is important to extract the output triples this way after the verification instead of directly outputting the triples shared by D, since we need to guarantee that all the output triples are unknown to Adv when D is honest and the verification process leaks some of the triples shared

by D. The verification of the relation $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ in an error-free way using the single synchronous round while leaking at most t values on each polynomial is the core idea in this protocol[3]. Briefly, the relation can be verified while leaking at most t points on the polynomials if each party P_i can verify if $\mathsf{Z}(\alpha_i) \stackrel{?}{=} \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$. However in a fully asynchronous setting, it is impossible to wait for the response of all the n parties. Here the synchronous round comes to our rescue. We ask each party to "non-verifiably" share a random multiplication triple in the first synchronous round[4] and later each such shared triple is used to perform the verification on behalf of the corresponding party in an asynchronous fashion.

Extracting Random Multiplication Triples (Section 6.2): Let $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i\in[3t+1]}$ be the set of $3t+1$ *local* t-shared multiplication triples, such that at least $2t+1$ of them are shared by the *honest* parties. From these shared triples, we compute the set of shared multiplication triples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i\in[3t+1]}$ using the transformation protocol. Since the input triples are guaranteed to be multiplication triples, the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds. Furthermore, since Adv may know at most t input triples, t output triples are leaked to Adv leaving $\frac{t}{2}$ "degree of freedom" in these polynomials. It allows us to output $\frac{t}{2}$ shared random multiplication triples unknown to Adv as $\{\mathbf{a}^{(i)}, \mathbf{b}^{(i)}, \mathbf{c}^{(i)}\}_{i\in[\frac{t}{2}]}$ where $\mathbf{a}^{(i)} = \mathsf{X}(\beta_i), \mathbf{b}^{(i)} = \mathsf{Y}(\beta_i)$ and $\mathbf{c}^{(i)} = \mathsf{Z}(\beta_i)$ for $\{\beta_i\}_{i\in[\frac{t}{2}]}$ distinct from $\{\alpha_i\}_{i\in[n]}$.

2 Model, Definitions and Notations

Model: We consider a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of n parties connected by pair-wise secure (private and authentic) channels, where $n = 4t+1$. The distrust in the network is modelled by a computationally unbounded centralized adaptive adversary Adv who can actively corrupt any t parties and can force them to deviate in any arbitrary manner during the execution of a protocol. The communication channels are asynchronous, having arbitrary, but finite delay (i.e. the messages will reach to their destination eventually). The order of the message delivery is decided by a *scheduler* and to model the worst case scenario, we assume that the scheduler is under the control of Adv. However, the scheduler can only schedule the messages exchanged between the honest parties (who are not under the control of Adv), without having any access to the "contents" of these messages. As in [7], we consider a computation (protocol execution) in the asynchronous model as a sequence of *atomic steps*, where a single party is *active* in each such step. A party is activated by receiving a message after which it performs an internal computation and then possibly sends messages on its outgoing channels. The order of the atomic steps are controlled by the scheduler. At the beginning of the computation, each party will be in a special *start* state. A party is said to terminate/complete the computation if it reaches a *halt* state, after which it does not perform any further computation. A protocol execution is said to be complete if all the honest parties terminate the protocol.

We assume that the function f to be computed is specified as a publicly known arithmetic circuit C over a finite field $\mathbb{F}$, where $|\mathbb{F}| > 2n$ and $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$ are publicly known distinct elements from $\mathbb{F}$. For our *statistical* AMPC protocol, we additionally require that $|\mathbb{F}| \geq 2t \cdot 2^{\kappa}$, for a given error parameter κ, to bound the error probability by $2^{-\kappa}$. The circuit C consists of input, addition (linear), multiplication, random and output gates. We denote the number of gates of each type by c_I, c_L, c_M, c_R and c_O respectively. We assume that each party P_i has c_i inputs for the computation and so $c_1 + \ldots + c_n = c_I$. Without loss of generality, we assume that f is expressed as $f : \mathbb{F}^{c_I} \rightarrow \mathbb{F}^n$, where $f(\vec{x}_1, \ldots, \vec{x}_n) = (y_1, \ldots, y_n)$, such that $\vec{x}_i \in \mathbb{F}^{c_i}$ denotes the vector of inputs of party P_i and P_i is supposed to receive the output $y_i \in \mathbb{F}$.

Definitions: A "property based" definition of secure AMPC in the information theoretic setting was followed in [9], which in essence is "equivalent" to the more standard definition of secure AMPC in the "real-world/ideal-world" paradigm [7]. All the papers on AMPC since then follow the style of definition used in [9]. As our main goal is to provide efficient AMPC protocols, we keep the formalities to a bare minimum and instead use the

[3]Probabilistically $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ can be verified by checking $\mathsf{Z}(\alpha) \stackrel{?}{=} \mathsf{X}(\alpha)\mathsf{Y}(\alpha)$ for a random α [10]; we aim for error-freeness.

[4]By "non-verifiably", we mean neither the correctness of the t-sharing nor the fact that the shared triple is a multiplication triple is guaranteed when the party that generates the sharing is corrupted.

property based definition to prove the security. However, using standard techniques, our protocols can be proved secure according to the (simulation based) real world/ideal-world definition of [7].

Definition 2.1 (Secure AMPC [9])**.** *Let $f : \mathbb{F}^{c_I} \rightarrow \mathbb{F}^n$ be a publicly known function and party P_i has input $\vec{x}_i \in \mathbb{F}^{c_i}$. In any asynchronous multiparty computation, each party P_i first commits to its input. Let $\vec{x'}_i$ be the values committed by P_i. If P_i is honest then $\vec{x'}_i = \vec{x}_i$. Due to the asynchronicity, the parties cannot wait for all the n parties to commit their inputs and so the parties agree on a subset* Com *of size $n - t$ of committed inputs. Finally, the parties compute an* estimation *of f as $(y_1, \ldots, y_n) = f(\vec{\mathsf{x}}_1, \ldots, \vec{\mathsf{x}}_n)$ and P_i receives the output y_i, where $\vec{\mathsf{x}}_i = \vec{x'}_i$ if $P_i \in$ Com, otherwise $\vec{\mathsf{x}}_i = 0^{c_i}$. An asynchronous protocol Π among the n parties* securely computes *f if it satisfies the following conditions for every possible* Adv *and every possible scheduler, on all possible inputs:*

(1) TERMINATION*: All the honest parties terminate Π* almost surely*, i.e. with probability*[5] 1. **(2)** CORRECTNESS*: Every honest party P_i correctly obtains his output y_i after completing Π, irrespective of the behavior of the corrupted parties.* **(3)** PRIVACY*: The adversary obtains no additional information (in the information theoretic sense), other than what is inferred from the inputs and the outputs of the corrupted parties.*

Definition 2.2 (Statistical and Perfect AMPC [7, 9])**.** *A statistical AMPC protocol satisfies the termination and correctness with probability at least $(1 - 2^{-\kappa})$, for a given error parameter κ (no compromise in the privacy property is made). A perfect AMPC protocol satisfies the termination and correctness with probability* 1 .

Definition 2.3 (d-sharing [3, 4, 18, 5])**.** *A value $s \in \mathbb{F}$ is said to be d-*shared *among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$ if every (honest) party $P_i \in \overline{\mathcal{P}}$ is holding a* share *s_i of s, such that there exists a polynomial $p(\cdot)$ of degree at most d, where $p(0) = s$ and $p(\alpha_i) = s_i$*[6] *holds for every (honest) $P_i \in \overline{\mathcal{P}}$. The vector of shares $\{s_i\}_{P_i \in \overline{\mathcal{P}}}$, corresponding to the (honest) parties in $\overline{\mathcal{P}}$ is called a d-sharing of s and denoted by $[s]_d^{\overline{\mathcal{P}}}$. A set of shares is called d-*consistent *if the shares lie on a polynomial of degree at most d. A vector $\vec{S} = (s^{(1)}, \ldots, s^{(\ell)})$ of ℓ values is said to be d-shared among a set of parties $\overline{\mathcal{P}}$ if each $s^l \in \vec{S}$ is d-shared among the parties in $\overline{\mathcal{P}}$.*

We write $[s]_d$ (ignoring the superscript) to mean that s is d-shared among *all* the (honest) parties in $\mathcal{P}$. Notice that d-sharings are *linear* in the sense that given $[a]_d$ and $[b]_d$, then $[a + b]_d = [a]_d + [b]_d$ and $[c \cdot a]_d = c \cdot [a]_d$, for a publicly known constant c. In general, given ℓ sharings $([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d)$ and a publicly known linear function $g : \mathbb{F}^\ell \rightarrow \mathbb{F}^m$, where $g(x^{(1)}, \ldots, x^{(\ell)}) = (y^{(1)}, \ldots, y^{(m)})$, then $g([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d) = ([y^{(1)}]_d, \ldots, [y^{(m)}]_d)$. By saying that the parties compute (locally) $([y^{(1)}]_d, \ldots, [y^{(m)}]_d) = g([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d)$, we mean that every party P_i (locally) applies the function g to its shares of $x^{(1)}, \ldots, x^{(\ell)}$. That is, party P_i computes $(y_i^{(1)}, \ldots, y_i^{(m)}) = g(x_i^{(1)}, \ldots, x_i^{(\ell)})$, where $y_i^{(l)}$ and $x_i^{(l)}$ denotes the ith share of $y^{(l)}$ and $x^{(l)}$ respectively.

Definition 2.4 (Random Multiplication Triples)**.** *A triple (x, y, z) is called a multiplication triple, if $z = xy$ holds. A multiplication triple (x, y, z) is called random if x, y and z are uniformly random subject to $z = xy$.*

Notation: By $[X]$ and $[X, Y]$ for $Y \geq X$, we denote the sets $\{1, \ldots, X\}$ and $\{X, X + 1, \ldots, Y\}$, respectively.

3 Existing Building Blocks

3.1 Private and Public Reconstruction of d-shared Values

Private reconstruction of a d-shared value: Let $[v]_d^{\overline{\mathcal{P}}}$ be a d-sharing of v, shared through a polynomial $p(\cdot)$ of degree at most d, where $d < |\overline{\mathcal{P}}| - 2t$. The goal is to make some party $P_R \in \mathcal{P}$ to *privately* reconstruct v. The well-know algorithm called *online error correction* (OEC) [7, 13] allows P_R to reconstruct the polynomial $p(\cdot)$ and thus v, as $p(0) = v$. The intuition behind OEC is that every party in $\overline{\mathcal{P}}$ sends its share of v to P_R, who waits to

[5] All probabilities are taken over all possible random coins of the honest parties. Asynchronous Byzantine agreement (ABA) is a special case of secure computation. From [20], it follows that any (probabilistic) ABA protocol must have some non-terminating runs. Thus, the best we can hope for is that a secure computation protocol terminates with probability 1 (see [7]).

[6] We often say that the share s_i lie on the polynomial $p(\cdot)$.

receive $d+t+1$ d-consistent shares from the parties in $\overline{\mathcal{P}}$. We call the protocol as $\mathsf{OEC}(P_R, d, [v]_d^{\overline{\mathcal{P}}})$. The protocol requires a private communication of $\mathcal{O}(n)$ elements from $\mathbb{F}$. If P_R is *honest* then no additional information about v is leaked to Adv. The protocol and its properties can be found in Appendix A.1.

Batch public reconstruction of several d-shared values: Protocol BatRecPubl takes $(t+1)(=\Theta(n))$ d-shared values $[u^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [u^{(t+1)}]_d^{\overline{\mathcal{P}}}$, where $d < |\overline{\mathcal{P}}| - 2t$ and allows *all* the parties in $\mathcal{P}$ to robustly reconstruct $u^{(1)}, \ldots, u^{(t+1)}$, such that the protocol requires a private communication of $\mathcal{O}(n^2)$. Protocol BatRecPubl is based on the following idea of [18]: we first "expand" the $t+1$ sharings to n sharings, say $[v^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [v^{(n)}]_d^{\overline{\mathcal{P}}}$, by applying a linear function. Specifically, by interpreting $u^{(1)}, \ldots, u^{(t+1)}$ as the coefficients of a polynomial $u(\cdot)$ of degree at most t and by letting $v^{(1)}, \ldots, v^{(n)}$ to be the evaluations of $u(\cdot)$ at n publicly known distinct points, each $[v^{(i)}]_d$ can be computed as linear combination of the given $t+1$ sharings. Once the parties compute $[v^{(1)}]_d, \ldots, [v^{(n)}]_d$, then $v^{(i)}$ is privately reconstructed towards the party P_i using an instance of OEC. Each P_i then sends $v^{(i)}$ to every other party in $\mathcal{P}$. Finally, each party applies OEC on the received $v^{(i)}$s to reconstruct the polynomial $u(\cdot)$ and outputs $u^{(1)}, \ldots, u^{(t+1)}$. We call the protocol as $\mathsf{BatRecPubl}(\mathcal{P}, d, [u^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [u^{(t+1)}]_d^{\overline{\mathcal{P}}})$. Every honest party will eventually terminate the protocol. The protocol and its properties can be found in Appendix A.2.

3.2 Batch Multiplication of ℓ Pairs of t-shared Values using Beaver's Technique

Beaver's circuit randomization method [2] is a well known method for securely computing $[xy]_t$ from $[x]_t$ and $[y]_t$, at the expense of two *public reconstructions*, using a *pre-computed* t-shared random multiplication triple (from the pre-processing phase), say $([a]_t, [b]_t, [c]_t)$. For this, the parties first (locally) compute $[e]_t$ and $[d]_t$, where $[e]_t = [x]_t - [a]_t = [x-a]_t$ and $[d]_t = [y]_t - [b]_t = [y-b]_t$, followed by the public reconstruction of $e = (x-a)$ and $d = (y-b)$. Since the relation $xy = ((x-a)+a)((y-b)+b) = de + eb + da + c$ holds, the parties can locally compute $[xy]_t = de + e[b]_t + d[a]_t + [c]_t$, once d and e are publicly known. The above computation leaks no information about x and y if a and b are random and unknown to Adv. For the sake of efficiency, we will apply the Beaver's trick on a batch of ℓ pairs of t-shared values simultaneously so that we can use BatRecPubl to publicly reconstruct the 2ℓ values (note that two reconstructions are required for each pair) at once efficiently. Looking ahead, we require ℓ to be at least $t+1$ and this will result the public reconstruction to cost a private communication of $\mathcal{O}(\lceil \frac{2\ell}{t+1} \rceil \cdot n^2) = \mathcal{O}(n\ell)$. We call the protocol as $\mathsf{BatchBeaver}(\{([x^{(i)}]_t, [y^{(i)}]_t, [a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t)\}_{i\in[\ell]})$ and state its properties in Theorem 3.1. The protocol and the proof of Theorem 3.1 can be found in Appendix A.3.

Theorem 3.1. *Let $\{([x^{(i)}]_t, [y^{(i)}]_t)\}_{i\in[\ell]}$ be a batch of ℓ pairs of t-sharing and $\{([a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t)\}_{i\in[\ell]}$ be the t-sharing of ℓ random multiplication triples unknown to* Adv*, where $\ell \geq t+1 = \Omega(n)$. Then for every possible* Adv *and for every possible scheduler, protocol* BatchBeaver *achieves:*

(1) TERMINATION*: All the honest parties eventually terminate.* **(2)** CORRECTNESS*: The protocol outputs $\{[x^{(i)}\ y^{(i)}]_t\}_{i\in[\ell]}$.* **(3)** PRIVACY*: The view of* Adv *is distributed independently of the $x^{(i)}$s and $y^{(i)}$s.* **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n\ell)$ elements from $\mathbb{F}$.*

3.3 Agreement on a Common Set (ACS)

Protocol ACS [7, 9] is a well known asynchronous protocol. It allows the (honest) parties to agree on a *common* set Com of $(n-t)$ parties, who have correctly shared values, which may be input to the computation, a multiplication triple, a random value, etc. The idea behind ACS is to execute n instances of an ABA protocol [13], one on the behalf of each party to decide if it should be included in Com. The protocol is invoked constant number of times in our AMPC protocols and is presented in Appendix A.4. The communication complexity of ACS is $\mathcal{O}(\text{poly}(n))$.

3.4 Asynchronous Broadcast

Bracha [11] implemented the asynchronous broadcast protocol A-Cast with $3t+1$ parties, which allows a sender Sen $\in \mathcal{P}$ to send some message m identically to all the n parties. If Sen is *honest* then all the honest parties

eventually terminate with output m. If Sen is *corrupted* and some *honest* party terminates with output m', then every other honest party eventually terminates with output m'. The protocol requires a private communication of $\mathcal{O}(n^2|m|)$ to broadcast a message m of size $|m|$ [4, 29, 28]. We say that P_i receives m from the broadcast of P_j if P_i terminates the instance of A-Cast where P_j is Sen with m as the output; see Appendix A.5 for the protocol.

3.5 Generating a Random Value

The following well known protocol called Rand() allows the parties to generate a uniformly randomly value $r \in \mathbb{F}$: every party $P_i \in \mathcal{P}$ t-shares (using any AVSS scheme; in fact the protocol presented in Section 4 can be used for this purpose) a random value $r^{(i)}$. The parties then execute an instance of ACS and decide a set of $n - t$ parties Com who have correctly shared the values. Finally, the parties set $[r]_t = \sum_{P_i \in \mathsf{Com}} [r^{(i)}]_t$ and publicly reconstruct the value r. Since the shared values of at least $n - 2t$ honest parties in Com are unknown to Adv and random (due to AVSS), r is indeed random. We note that the communication complexity of the protocol is polynomial in n.

4 Generating Batch of t-shared Values

We present a protocol called Sh, which allows a dealer $\mathsf{D} \in \mathcal{P}$ to t-share ℓ values $\vec{S} = (s^{(1)}, \ldots, s^{(\ell)})$ in a "verifiable" manner, where $\ell \geq t + 1$. The "verifiability" ensures that irrespective of the status of D, if the honest parties terminate the protocol then the output sharings are t-sharing. The protocol requires a private communication of $\mathcal{O}(n\ell)$ and a broadcast communication of $\mathcal{O}(n^2)$. We first explain the protocol assuming that $\vec{S}$ has $t+1$ secrets and later discuss how to extend it to share more than $t + 1$ values together.

The starting point of our protocol Sh is the sharing protocol of the perfectly secure AVSS scheme of [28, 29]. The AVSS protocol of [28, 29] enables D to $2t$-share a single secret s. The $2t$-sharing is achieved via a univariate polynomial $F(x, 0)$ of degree at most $2t$, where $F(x, y)$ is a random bi-variate polynomial of degree at most $2t$ in x and at most t in y (note the difference in degrees), such that $F(0, 0) = s$. Initially, D is asked to pick $F(x, y)$ and hand over the ith row polynomial $f_i(x)$ of degree at most $2t$ and the ith column polynomial $g_i(y)$ of degree at most t to the party P_i, where $f_i(x) \stackrel{def}{=} F(x, \alpha_i)$ and $g_i(y) \stackrel{def}{=} F(\alpha_i, y)$. If the sharing protocol terminates, then it is ensured that there exists a bi-variate polynomial $F'(x, y)$ of degree at most $2t$ in x and at most t in y, such that every *honest* party P_j holds a column polynomial $g'_j(y)$ of degree at most t, where $g'_j(y) = F'(\alpha_j, y)$. This makes the secret $s' \stackrel{def}{=} F'(0, 0)$ to be $2t$-shared through the polynomial $f'_0(x)$ of degree at most $2t$ where $f'_0(x) \stackrel{def}{=} F'(x, 0)$ and every *honest* party P_j holds its share s'_j of s', with $s'_j = f'_0(\alpha_j) = F'(\alpha_j, 0) = g'_j(0)$. For an *honest* D, $F'(x, y) = F(x, y)$ will hold and thus s will be $2t$-shared though the polynomial $f_0(x) \stackrel{def}{=} F(x, 0)$.

To achieve our goal by using the sharing protocol of [29, 28], we first use the fact that the adversary's view in the sharing protocol leaves $(t+1)(2t+1) - t(2t+1) - t = (t+1)$ "degree of freedom" in the polynomial $F(x, y)$ when D is *honest*. Informally this is because, Adv receives $t(2t+1)+t$ distinct points on $F(x, y)$ through the t row and column polynomials of the corrupted parties, but $(t+1)(2t+1)$ distinct points are required to completely define $F(x, y)$. While the authors of [29, 28] used the $t + 1$ degree of freedom to create a *single* $2t$-sharing by embedding a single secret in $F(x, y)$, we use it to create t-sharing of $t + 1$ different secrets by embedding $t + 1$ secrets in $F(x, y)$. Namely, given $t + 1$ secrets $\vec{S} = (s^{(1)}, \ldots, s^{(t+1)})$, the dealer D in our protocol fixes $F(\beta_l, 0) = s^{(l)}$ for $l \in [t + 1]$, where $F(x, y)$ is otherwise a random polynomial of degree at most $2t$ in x and at most t in y. At the end, the goal is that the secret $s^{(l)}$ is t-shared among the parties through the polynomial $F(\beta_l, y)$ of degree at most t, which we denote by $g_{\beta_l}(y)$. As depicted in Fig. 1 (in blue color), an *honest* party P_i can compute his shares of all the secrets in $\vec{S}$ by local computation, once it gets the polynomial $f_i(x) = F(x, \alpha_i)$. This follows from the fact that for $l \in [t + 1]$ the ith share $s_i^{(l)}$ of the secret $s^{(l)}$ satisfies $s_i^{(l)} = g_{\beta_l}(\alpha_i) = f_i(\beta_l)$.

To enable P_i to get $f_i(x)$, we recall that the sharing protocol of [29, 28] ensures that every *honest* P_j holds $g'_j(y)$ such that there exists a bi-variate polynomial $F'(x, y)$ of degree at most $2t$ in x and at most t in y such that $F'(\alpha_j, y) = g'_j(y)$ and furthermore for an honest D, $F'(x, y) = F(x, y)$ holds. Since by holding $g'_j(y)$, P_j already holds a point on every $f'_i(x)$ (namely $g'_j(\alpha_i)$ is the same as $f'_i(\alpha_j)$, where $f'_i(x) = F'(x, \alpha_i)$), we can ensure the

(private) reconstruction of the polynomial $f'_i(x)$ by P_i by asking every party P_j to send its point on $f'_i(x)$ to P_i. Since $f'_i(x)$ has degree at most $2t$ and there are $4t+1$ parties, OEC enables P_i to compute $f'_i(x)$ from the received points. This completes our discussion on t-sharing of $t+1$ secrets verifiably. We note that for a *corrupted* D, the values $\vec{S'} = (F'(\beta_1, 0), \ldots, F'(\beta_{t+1}, 0))$ will be t-shared and in case of an honest D, $\vec{S'} = \vec{S}$ will hold.

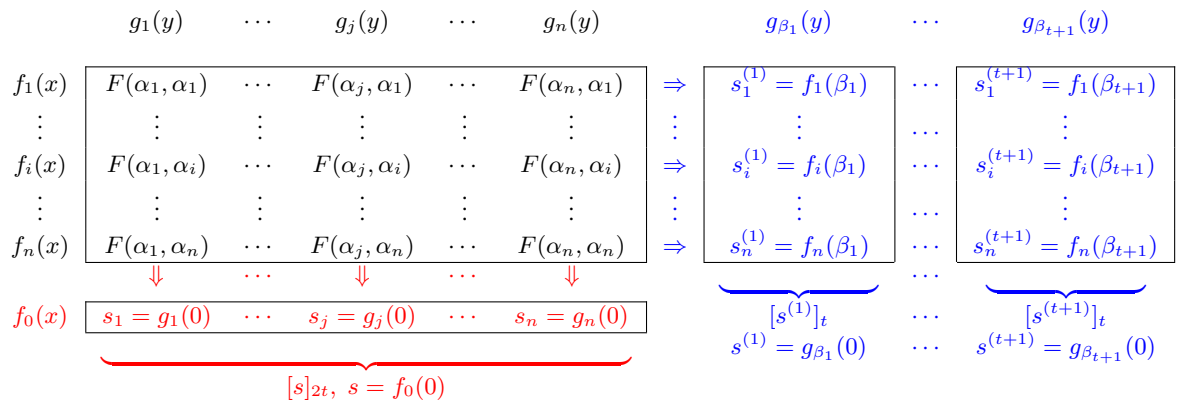

Figure 1: Pictorial representation of the values distributed by (an honest) D in the AVSS scheme of [29, 28] and protocol Sh. The polynomials $f_1(x), \ldots, f_n(x), g_1(y), \ldots, g_n(y)$ computed from the bi-variate polynomial $F(x, y)$ of degree at most $2t$ and t in x and y are distributed in both the protocols. In the AVSS protocol, s is $2t$-shared through the row polynomial $f_0(x)$ (shown in red color), while in Sh, $t+1$ values $s^{(1)}, \ldots, s^{(t+1)}$ are t-shared through the $t+1$ column polynomials $g_{\beta_1}(y), \ldots, g_{\beta_{t+1}}(y)$ (shown in blue color).

Finally, we would like to comment that our idea of embedding several secrets in a single *bi-variate* polynomial is different from the notion of *packed secret sharing* [21] where k secrets are embedded in a single *univariate* polynomial of degree at most t such that each party receives a single share, namely a distinct point on the polynomial. In the later, a single share is the share for k secrets and the robust reconstruction works when the adversary controls at most $t-k+1$ instead of t parties. Protocol Sh, on the other hand, ensures that *each* secret in $\vec{S}$ is *independently* t-shared and thus the robust reconstruction of each secret works even when the adversary corrupts t parties.

To avoid repetition of details, we present the protocol Sh in Appendix B next to the description of the AVSS scheme of [29, 28]. The properties of Sh, stated in Theorem 4.1, mostly follow from the AVSS scheme of [29, 28] and the discussion above; see Appendix B for more details.

Theorem 4.1. *Let $\vec{S} = (s^{(1)}, \ldots, s^{(t+1)})$ be a vector of $t+1$ values, which a dealer* D $\in \mathcal{P}$ *wants to t-share among the parties in $\mathcal{P}$. Then for every possible* Adv *and scheduler, the protocol* Sh *achieves:*

(1) TERMINATION*: If* D *is honest, then every honest party will eventually terminate* Sh*. Moreover, even if* D *is corrupted and some honest party terminates* Sh*, then all the honest parties will eventually terminate* Sh*.* **(2)** CORRECTNESS*: Once an honest party terminates* Sh*, there exists a vector $\vec{S'} = (s'^{(1)}, \ldots, s'^{(t+1)})$ of $t+1$ values, such that $\vec{S'}$ will be eventually t-shared among the parties in $\mathcal{P}$. Moreover, if* D *is honest, then $\vec{S'} = \vec{S}$.* **(3)** PRIVACY*: If* D *is honest, then the information received by* Adv *during the protocol* Sh *is distributed independently of the shared secrets in $\vec{S}$.* **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$ and a broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$.*

4.1 Sharing more than $t+1$ Values without Blowingup the Broadcast Communication

On having ℓ values for $\ell > t+1$, D can divide them into groups of $t+1$ and execute an instance of Sh for each group. This will require a private communication of $\mathcal{O}(\lceil \frac{\ell}{(t+1)} \rceil \cdot n^2) = \mathcal{O}(n\ell)$ field elements, since $(t+1) = \Theta(n)$. The broadcast communication can be kept $\mathcal{O}(n^2)$ (*independent* of ℓ) by executing all instances of Sh (each handling $t+1$ secrets) in parallel and by asking each party to broadcast only once for all the instances, after confirming the veracity of the "pre-condition" for the broadcast for *all* the instances of Sh. The sharing protocol of the AVSS scheme of [29, 28] describes the same idea to keep the broadcast communication independent of ℓ when D $2t$-shares ℓ secrets; for details see Appendix B. In the rest of the paper, we will say that a party t-shares ℓ values, where $\ell \geq t+1$ using an instance of Sh to mean the above.

5 Transforming Independent Shared Triples to Co-related Shared Triples

We present a protocol TripTrans which takes as input a set of $(3t+1)$ *independent* t-shared triples $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[3t+1]}$ and outputs a set of $(3t+1)$ "co-related" t-shared triples $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t)\}_{i\in[3t+1]}$, such that the following holds: **(a)** There exist polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ of degree at most $\frac{3t}{2}$, $\frac{3t}{2}$ and $3t$ respectively, such that $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$, for $i \in [3t+1]$. **(b)** The ith output triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a multiplication triple iff the ith input triple $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication triple. This further implies that $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true iff all the $3t+1$ input triples are multiplication triples. **(c)** If Adv knows t' input triples and if $t' \leq \frac{3t}{2}$, then Adv learns t' distinct values of $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, implying at least $\frac{3t}{2} + 1 - t'$ "degree of freedom" on $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$. If $t' > \frac{3t}{2}$, then Adv will completely know $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$.

The protocol is inherited from the protocol for the batch verification of the multiplication triples proposed in [10]. The idea is as follows: we assume $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ to be "defined" by the first and second component of the *first* $\frac{3t}{2}+1$ input triples, compute $\frac{3t}{2}$ "new" points on the $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ polynomials and compute the product of the $\frac{3t}{2}$ new points using Beaver's technique making use of the *remaining* $\frac{3t}{2}$ input triples. The $\mathsf{Z}(\cdot)$ is then defined by the $\frac{3t}{2}$ computed products and the third component of the first $\frac{3t}{2}+1$ input triples. In a more detail, we define the polynomial $\mathsf{X}(\cdot)$ (of degree at most $\frac{3t}{2}$) by setting $\mathsf{X}(\alpha_i) = x^{(i)}$ for $i \in [\frac{3t}{2}+1]$ and get $[\mathbf{x}^{(i)}]_t = [\mathsf{X}(\alpha_i)]_t = [x^{(i)}]_t$ for $i \in [\frac{3t}{2}+1]$. Following the same logic, we define $\mathsf{Y}(\alpha_i) = y^{(i)}$ for $i \in [\frac{3t}{2}+1]$ and get $[\mathbf{y}^{(i)}]_t = [\mathsf{Y}(\alpha_i)]_t = [y^{(i)}]_t$ for $i \in [\frac{3t}{2}+1]$. Moreover, we set $\mathsf{Z}(\alpha_i) = z^{(i)}$ for $i \in [\frac{3t}{2}+1]$ and get $[\mathbf{z}^{(i)}]_t = [\mathsf{Z}(\alpha_i)]_t = [z^{(i)}]_t$ for $i \in [\frac{3t}{2}+1]$.

Now for $i \in [\frac{3t}{2}+2, 3t+1]$, we compute $[\mathbf{x}^{(i)}]_t = [\mathsf{X}(\alpha_i)]_t$ and $[\mathbf{y}^{(i)}]_t = [\mathsf{Y}(\alpha_i)]_t$ which requires only local computation on the t-sharings $\left\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t)\right\}_{i\in[\frac{3t}{2}+1]}$, as it is a linear function. For $i \in [\frac{3t}{2}+2, 3t+1]$, fixing $\mathbf{z}^{(i)}$ to be the same as $z^{(i)}$ will, however, violate the requirement that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds if all the input triples are multiplication triples; this is because $\mathbf{x}^{(i)} = \mathsf{X}(\alpha_i) \neq x^{(i)}$ and $\mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)} \neq y^{(i)}$ and thus $z^{(i)} = x^{(i)}y^{(i)} \neq \mathbf{x}^{(i)}\mathbf{y}^{(i)}$. Here we resort to the Beaver's technique to find $[\mathbf{z}^{(i)}]_t = [\mathbf{x}^{(i)}\mathbf{y}^{(i)}]_t$ from $[\mathbf{x}^{(i)}]_t$ and $[\mathbf{y}^{(i)}]_t$, using the t-shared triples $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[\frac{3t}{2}+2,3t+1]}$. We note that the triples $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[\frac{3t}{2}+2,3t+1]}$ are never touched before for any computation and used only for the Beaver's technique.

It is easy to see that $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a multiplication triple if and only if $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication triple. For $i \in [\frac{3t}{2}+1]$, this is trivially true, as for such an i, $([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t) = ([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)$. For $i \in [\frac{3t}{2}+2, 3t+1]$, it follows from the correctness of the Beaver's technique and the fact that $([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)$ is used to compute $[\mathbf{z}^{(i)}]_t$ from $[\mathbf{x}^{(i)}]_t$ and $[\mathbf{y}^{(i)}]_t$ and so $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}\mathbf{y}^{(i)}$ if and only if $z^{(i)} = x^{(i)}y^{(i)}$.

For privacy, we see that if Adv knows the ith input triple then the ith output triple will be known to Adv: for $i \in [\frac{3t}{2}+1]$ the statement is trivially true, while for $i \in [\frac{3t}{2}+2, 3t+1]$, the statement follows because Adv will know the ith input triple $(x^{(i)}, y^{(i)}, z^{(i)})$, which is used to compute $[\mathbf{z}^{(i)}]_t$ from $[\mathbf{x}^{(i)}]_t$ and $[\mathbf{y}^{(i)}]_t$. Since $(\mathbf{x}^{(i)} - x^{(i)})$ and $(\mathbf{y}^{(i)} - y^{(i)})$ are disclosed during the computation of $\mathbf{z}^{(i)}$, Adv will learn $\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$ and trivially $\mathbf{z}^{(i)}$. Thus if Adv knows t' input triples where $t' \leq \frac{3t}{2}$ then Adv will learn t' output triples and hence t' values of the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, leaving $\frac{3t}{2}+1-t'$ degree of freedom in these polynomials. The protocol is presented in Fig. 2; its properties are stated in Theorem 5.1 and the proof can be found in Appendix C.

Figure 2: Protocol for transforming a set of independent shared triples to a set of co-related shared triples.

Protocol $\mathsf{TripTrans}(\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[3t+1]})$

1. For each $i \in [\frac{3t}{2}+1]$, the parties locally set $[\mathbf{x}^{(i)}]_t = [x^{(i)}]_t$, $[\mathbf{y}^{(i)}]_t = [y^{(i)}]_t$ and $[\mathbf{z}^{(i)}]_t = [z^{(i)}]_t$.
2. Let the points $\{(\alpha_i, \mathbf{x}^{(i)})\}_{i\in[\frac{3t}{2}+1]}$ and the points $\{(\alpha_i, \mathbf{y}^{(i)})\}_{i\in[\frac{3t}{2}+1]}$ define the polynomial $\mathsf{X}(\cdot)$ and $\mathsf{Y}(\cdot)$ respectively of degree at most $\frac{3t}{2}$. The parties locally compute $[\mathbf{x}^{(i)}]_t = [\mathsf{X}(\alpha_i)]_t$ and $[\mathbf{y}^{(i)}]_t = [\mathsf{Y}(\alpha_i)]_t$, for each[a] $i \in [\frac{3t}{2}+2, 3t+1]$.
3. The parties execute $\mathsf{BatchBeaver}(\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[\frac{3t}{2}+2,3t+1]})$ to compute $\frac{3t}{2}$ sharings $\{[\mathbf{z}^{(i)}]_t\}_{i\in[\frac{3t}{2}+2,3t+1]}$. The parties output $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t)\}_{i\in[3t+1]}$ and terminate.

[a] Computing a new point on a polynomial of degree d is a linear function of $d+1$ given unique points on the same polynomial.

Theorem 5.1. *For every possible* Adv *and every possible scheduler, protocol* TripTrans *achieves:*

(1) TERMINATION*: All the honest parties eventually terminate the protocol.* **(2)** CORRECTNESS*:* **(a)** *There exist polynomials* $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ *and* $\mathsf{Z}(\cdot)$ *of degree* $\frac{3t}{2}, \frac{3t}{2}$ *and* $3t$ *respectively, 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* $i \in [3t+1]$. **(b)** $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ *holds iff all the input triples are multiplication triples.* **(3)** PRIVACY*: If* Adv *knows* $t' \leq \frac{3t}{2}$ *input triples then* Adv *knows* t' *values on* $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ *and* $\mathsf{Z}(\cdot)$. **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of* $\mathcal{O}(n^2)$ *elements from* $\mathbb{F}$.

6 The Asynchronous Preprocessing Phase

We present a completely asynchronous protocol called PreProc, which generates t-sharing of $c_M + c_R$ random multiplication triples, unknown to Adv, with a private communication of $\mathcal{O}((c_M + c_R)n)$ and broadcast communication of $\mathcal{O}(n^3)$. All the existing protocols follow the following common strategy to create the t-sharing of random multiplication triples: first every party t-shares *pairs* of random values (possibly coupled with player elimination/dispute control technique). Then a "randomness extraction" algorithm is applied on the shared pairs to extract sharings of truly random pairs, unknown to Adv. Finally, known multiplication protocols are invoked to securely compute t-sharing of the product of the random pairs. We propose a completely different, albeit seemingly more natural approach. Briefly, the preprocessing phase takes t-sharing of "local" random multiplication triples shared by individual parties acting as dealers such that the local triples are random and private when shared by an honest dealer and then "extract" random multiplication triples from the shared multiplication triples. On a high level, the preprocessing phase protocol is built on *two* sub-protocols: the first subprotocol allows a dealer $\mathsf{D} \in \mathcal{P}$ to t-share random multiplication triples in a "verifiable" fashion. The "verifiability" ensures that irrespective of the status of D, if the honest parties terminate the subprotocol then the output sharings are of multiplication triples. The second subprotocol allows to securely extract random multiplication triples unknown to Adv from a set of local multiplication triples, shared by individual parties acting as a dealer. We next elaborate on the two subprotocols.

6.1 Verifiably Sharing Multiplication Triples

We present an asynchronous protocol called TripleSh which resorts to a probabilistic approach to ensure the verifiability of the shared triples except with a negligible error probability for a *corrupted* D.

The High Level Idea of TripleSh**:** Protocol TripleSh is based on the well known technique, called the "sacrificing trick" [19]. The trick deploys two triples and sacrifices the privacy of the second triple (if it was private before) to probabilistically confirm if the first triple is a multiplication triple by leaking no further information except the conviction that it is a multiplication triple. More specifically, suppose D has t-shared a pair of triples, say (x, y, z) and (f, g, h) and has claimed that they are multiplication triples. If the claim of D is false then except with probability at most $\frac{2}{|\mathbb{F}|}$, the relation $r(z - xy) = (h - fg)$ will be false as well, for any *random challenge* $r \in \mathbb{F}$ unknown to D while t-sharing the triples (x, y, z) and (f, g, h). The probability comes from the fact that for every non-multiplication triple (x, y, z), there exists a *single non-zero* r for which the relation $r(z - xy) = (h - fg)$ will be true (we note that for $r = 0$, the relation will be true irrespective of (x, y, z)). To check the condition $r(z - xy) \stackrel{?}{=} (h - fg)$, the parties (locally) compute the t-sharing of ρ and σ, where $\rho = rx - f$, $\sigma = y - g$ and publicly reconstruct ρ and σ. The parties then locally compute the t-sharing of γ, where $\gamma = rz - h - \sigma f - \rho g - \sigma\rho$ and publicly reconstruct γ. By noting that γ is the same as $rz - rxy - h + fg$ which in turn is the same as $r(z - xy) - (h - fg)$, it can be concluded that the triple (x, y, z) is a multiplication triple if $\gamma = 0$. The above technique probabilistically confers the conviction if the triple (x, y, z) is a multiplication triple. The privacy of (x, y, z) for an *honest* D is maintained at the expense of sacrificing the privacy of (f, g, h) while making ρ, σ and γ public (to carry out the verification). Protocol TripleSh is presented in Fig. 3, where the above idea is applied on a batch of ℓ pairs of triples, where $\ell \geq t + 1$. A single r is used for the verification of all the ℓ pairs. Theorem 6.1 states the properties of TripleSh. The proof can be found in Appendix D.1

Figure 3: A probabilistic protocol for t-sharing ℓ random multiplication triples where $\ell \geq t+1$.

Protocol TripleSh(D)

1. D selects two sets of ℓ random multiplication triples $\vec{S} = ((x^{(1)}, y^{(1)}, z^{(1)}), \ldots, (x^{(\ell)}, y^{(\ell)}, z^{(\ell)}))$ and $\vec{S}_{sac} = ((f^{(1)}, g^{(1)}, h^{(1)}), \ldots, (f^{(\ell)}, g^{(\ell)}, h^{(\ell)}))$. D invokes an instance of the sharing protocol Sh to t-share all the secrets in $\vec{S}$ and $\vec{S}_{sac}$ together[a] and the parties participate in the instance of Sh.
2. After terminating the instance of Sh initiated by D, the parties execute an instance of Rand to generate a random r.
3. For each $i \in [\ell]$, the parties locally compute $[\rho^{(i)}]_t$ and $[\sigma^{(i)}]_t$ and then execute $\lceil \frac{2\ell}{t+1} \rceil$ instances of BatRecPubl[b] to reconstruct $\{\rho^{(i)}, \sigma^{(i)}\}_{i \in [\ell]}$, where $[\rho^{(i)}]_t = r[x^{(i)}]_t - [f^{(i)}]_t$ and $[\sigma^{(i)}]_t = [y^{(i)}]_t - [g^{(i)}]_t$.
4. For each $i \in [\ell]$, the parties locally compute $[\gamma^{(i)}]_t$ and then execute $\lceil \frac{\ell}{t+1} \rceil$ instances of BatRecPubl to reconstruct $\{\gamma^{(i)}\}_{i \in [\ell]}$, where $[\gamma^{(i)}]_t = r[z^{(i)}]_t - [h^{(i)}]_t - \sigma^{(i)}[f^{(i)}]_t - \rho^{(i)}[g^{(i)}]_t - \rho^{(i)}\sigma^{(i)}$.
5. Upon terminating the instances of BatRecPubl, parties (locally) check $\gamma^{(i)} \stackrel{?}{=} 0$ for each $i \in [\ell]$, output $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i \in [\ell]}$ if the condition holds and terminate. If the condition does not hold then the parties output default t-sharing of ℓ publicly known multiplication triples and terminate.

[a] See section 4.1 for the description of the protocol Sh sharing ℓ secrets together for any $\ell \geq t+1$.

[b] BatRecPubl allows to publicly reconstruct $t+1$ values and so $2\ell/(t+1)$ instances are required to reconstruct 2ℓ values.

Theorem 6.1. *For every possible* Adv *and for every possible scheduler, protocol* TripleSh *achieves:*

(1) TERMINATION*: If* D *is honest then all the honest parties eventually terminate the protocol. If* D *is corrupted and some honest party terminates, then all the honest parties eventually terminate.* **(2)** CORRECTNESS*: If* D *is honest then ℓ multiplication triples will be t-shared, where $\ell \geq t+1$. If* D *is corrupted and some honest party terminates then ℓ triples will be t-shared; moreover except with probability at most $\frac{2}{|\mathbb{F}|}$, the triples will be multiplication triples.* **(3)** PRIVACY*: If* D *is honest, then the view of* Adv *in the protocol is distributed independently of the output multiplication triples.* **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n\ell)$ elements and a broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$, plus one invocation to* Rand.

6.2 Extraction of Random Multiplication Triples from Local Multiplication Triples

Let Com $\subset \mathcal{P}$ be a set of $3t+1$ parties, known to all the parties in $\mathcal{P}$, such that every party in Com has t-shared ℓ multiplication triples among the parties in $\mathcal{P}$, where the the triples shared by the honest parties are random and unknown to Adv. We present a protocol called TripExt that "extracts" $\frac{t\ell}{2} = \Theta(n\ell)$ *random* t-shared multiplication triples unknown to Adv from these $(3t+1)\ell$ "local" t-shared multiplication triples with a private communication of $\mathcal{O}(n^2\ell)$. Without loss of generality, we assume Com to consist of the *first* $3t+1$ parties.

The high level idea is as follows: the input triples from the parties in Com are perceived as ℓ batches of $3t+1$ triples where the lth batch contains the lth local triple from each party in Com. Then the transformation protocol TripTrans is executed on the lth batch to obtain a new set of $3t+1$ triples and the three associated polynomials of degree $\frac{3t}{2}$, $\frac{3t}{2}$ and $3t$, namely $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$. Since, each input triple is guaranteed to be a multiplication triple, the multiplicative relation holds among the polynomials, i.e. $\mathsf{Z}^l(\cdot) = \mathsf{X}^l(\cdot)\mathsf{Y}^l(\cdot)$. Since Adv gets to know at most t input triples in the lth batch, the transformation ensures that Adv gets to know at most t points on each of the three polynomials, leaving $\frac{t}{2}$ degree of freedom on each polynomial. The random output multiplication triples for the lth batch, unknown to Adv, are then extracted as $\left\{([\mathsf{X}^l(\beta_i)]_t, [\mathsf{Y}^l(\beta_i)]_t, [\mathsf{Z}^l(\beta_i)]_t)\right\}_{i \in [\frac{t}{2}]}$ without requiring any interaction. Protocol TripExt is presented in Fig. 4. The properties of protocol TripExt are stated in Theorem 6.2 and the proof is provided in Appendix D.2.

Theorem 6.2. *Let* Com *be a set of $3t+1$ parties known to all the parties in $\mathcal{P}$, such that each party party $P_i \in$ Com has t-shared ℓ random multiplication triples $\{(x^{(i,l)}, y^{(i,l)}, z^{(i,l)})\}_{l \in [\ell]}$. Then for every possible* Adv *and for every possible scheduler, protocol* TripExt *achieves:*

(1) TERMINATION*: All the honest parties eventually terminate the protocol.* **(2)** CORRECTNESS*: Each triple in $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{l \in [\ell],\, i \in [\frac{t}{2}]}$ is a multiplication triple and will be t-shared.* **(3)** PRIVACY*: The view of* Adv *in*

Figure 4: Protocol for extracting $\frac{t\ell}{2}$ random multiplication triples from a set of $(3t+1)\ell$ local multiplication triples.

Protocol $\mathsf{TripExt}(\mathsf{Com}, \{([x^{(i,l)}]_i, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{l\in[\ell],\ P_i\in\mathsf{Com}})$

Without loss of generality, assume $\mathsf{Com} = \{P_1, \ldots, P_{3t+1}\}$. For each $l \in [\ell]$, the parties do the following in parallel:

1. The parties execute the protocol $\mathsf{TripTrans}(\{([x^{(i,l)}]_t, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{i\in[3t+1]})$ and compute the sharings $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{i\in[3t+1]}$, such that the points $\{(\alpha_i, \mathbf{x}^{(i,l)})\}_{i\in[3t+1]}, \{(\alpha_i, \mathbf{y}^{(i,l)})\}_{i\in[3t+1]}$ and $\{(\alpha_i, \mathbf{z}^{(i,l)})\}_{i\in[3t+1]}$ lie on the polynomial $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$ respectively of degree at most $\frac{3t}{2}$, $\frac{3t}{2}$ and $3t$.
2. The parties locally compute $[\mathbf{a}^{(i,l)}]_t = [\mathsf{X}^l(\beta_i)]_t$, $[\mathbf{b}^{(i,l)}]_t = [\mathsf{Y}^l(\beta_i)]_t$ and $[\mathbf{c}^{(i,l)}]_t = [\mathsf{Z}^l(\beta_i)]_t$ for $i \in [\frac{t}{2}]$ and terminate.

the protocol is distributed independently of the multiplication triples in $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{l\in[\ell],\ i\in[\frac{t}{2}]}$. **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of* $\mathcal{O}(n^2\ell)$ *elements from* $\mathbb{F}$.

6.3 Preprocessing Phase using Triple Sharing and Triple Extraction

Our asynchronous protocol PreProc for the preprocessing phase consists of the following three steps: **(1)** Every party in $\mathcal{P}$ acts as a dealer and t-share $\ell = \frac{2(c_M+c_R)}{t}$ random multiplication triples using an instance of the protocol TripleSh. **(2)** The parties then execute an instance of ACS to decide on a common set Com of $3t+1$ parties who have correctly shared multiplication triples in their respective instances of TripleSh. **(3)** The parties then execute the random triple extraction protocol TripExt on the triples shared by the parties in Com to extract $\frac{t\ell}{2} = c_M + c_R$ random multiplication triples. The properties of the protocol PreProc are stated in Theorem 6.3. The details of the protocol PreProc and the proof of Theorem 6.3 are provided in Appendix D.3.

Theorem 6.3. *For every possible* Adv *and every possible scheduler, protocol* PreProc *achieves:*

(1) TERMINATION*: All the honest parties terminate the protocol with probability* 1. **(2)** CORRECTNESS*: The output triples will be* t*-shared among the parties. Moreover, except with error probability at most* $\frac{2t}{\mathbb{F}}$*, the triples are multiplication triples.* **(3)** PRIVACY*: The view of* Adv *in the protocol is independent of the output multiplication triples.* **(4)** COMMUNICATION COMPLEXITY*: The protocol incurs private communication of* $\mathcal{O}((c_M + c_R)n)$ *and broadcast of* $\mathcal{O}(n^3)$ *elements from* $\mathbb{F}$ *plus one invocation to* ACS *and* Rand.

7 The AMPC Protocol

Given the preprocessing phase that generates t-sharing of $c_M + c_R$ random multiplication triples, an AMPC protocol is quiet straight forward to achieve by implementing the rest of the two phases: input phase and computation phase. In the sequel we discuss the high level idea and refer to Appendix E for the details.

In the *input phase* implemented by the protocol Input, the parties provide their input(s) for the computation. Recall that party P_i has c_i inputs and $c_1 + \ldots + c_n = c_I$, where c_I is the number of input gates in the circuit. Assuming $c_i \geq t+1$, party P_i executes an instance of Sh to t-share his c_i inputs. Due to asynchronicity, the parties cannot wait to terminate more than $n-t$ instances of Sh. So the parties execute an instance of ACS to agree on a set of $n-t$ parties whose inputs will be taken into computation. For the remaining t parties, default t-sharing of 0 is taken as the input to the computation.

The next phase is the *computation phase*, implemented by the protocol CircEval, where the circuit is securely "evaluated" on a gate by gate basis. This phase employs the standard technique of circuit evaluation in the information theoretic setting using preprocessed multiplication triples [3, 18, 4, 5]. The protocol (securely) implements the following *invariant* for each gate of the circuit: given the t-sharing of the input(s) of a gate, the protocol allows the parties to compute the t-sharing of the output of the gate. A gate is said to be evaluated if the t-sharing of the output of the gate is computed. This is achieved as follows for various gates: the linearity of the t-sharing ensures that the linear gates can be evaluated locally. For a multiplication gate, the parties associate a multiplication triple

from the set of preprocessed multiplication triples and then evaluate the gate by applying the Beaver's circuit randomization technique. Similar to [5] for the sake of efficiency, we evaluate $t+1$ multiplication gates at once by using the protocol BatchBeaver, assuming that in general the circuit is well-spread and we will have sufficiently many " independent" multiplication gates to evaluate in parallel. Here two gates are called independent of each other if the input of one gate does not depend on the output of the other gate. For every random gate, the parties associate a multiplication triple from the set of preprocessed multiplication triples and the first component of the triple is considered as the outcome of the random gate. Once all the gates are evaluated, the t-sharing of each output gate is privately reconstructed towards the party who is supposed to receive that output.

Finally, the AMPC protocol AsynAMPC is a sequence of three protocols: protocol PreProc, Input and CircEval, where a party terminates after completing all the three protocols. We avoid giving the complete formal (simulation based) proofs for CircEval, since it is well known in the literature (see [3, 18, 4, 5]) that the protocol CircEval described above securely computes the function f. The properties of the protocol AsynAMPC are stated in Theorem 7.1. For more details on Input, CircEval and the (informal) proof of Theorem 7.1, see Appendix E.

Theorem 7.1. *Let $f : \mathbb{F}^{c_I} \rightarrow \mathbb{F}^n$ be a publicly known function expressed as an arithmetic circuit C over a finite field $\mathbb{F}$, where $|\mathbb{F}| \geq \max\{2t \cdot 2^{\kappa}, 2n\}$ for a given error parameter κ, consisting of c_I, c_O, c_M and c_R number of input, output, multiplication and random gates respectively. Then for every possible* Adv *and for every possible scheduler, protocol* AsynAMPC *is a statistical AMPC protocol to securely compute f. The protocol requires a private communication of $\mathcal{O}((c_I + c_O + c_M + c_R)n)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$, along with two invocations to* ACS *and one invocation to* Rand.

8 Error-free AMPC Protocol in the Hybrid Model

An inherent drawback of a completely asynchronous setting is that the inputs of up to t potentially honest parties may get ignored. Noting this undesirability for many real-world applications, [4, 6] introduced a "partial synchronous" or hybrid setting wherein the very *first* communication round is a synchronous round. It was shown in [4] how to enforce "input provision" from all the n parties using the synchronous round, at the expense of generating $c_I(t+1)$ additional random t-sharings in the preprocessing stage. Interestingly in this paper, we further utilize the first synchronous round to present an *error-free* protocol called HybTripleSh for t-sharing multiplication triples. In contrast to the protocol TripleSh (see Section 6.1) designed for the same purpose in the fully asynchronous network, HybTripleSh ensures error-free verifiability of the shared triples even for a *corrupted* D. We dispose the high level idea in the following and defer the details to Appendix F.

High Level Idea of HybTripleSh**:** We explain the high level idea assuming that D wants to t-share $\frac{t}{2}$ multiplication triples. In HybTripleSh, the same idea is used parallely ℓ times, where $\ell \geq t+1$, resulting in t-sharing of $\frac{t\ell}{2}$ multiplication triples. To generate t-sharing of $\frac{t}{2}$ multiplication triples, D t-shares $3t+1$ triples, say $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[3t+1]}$. From these sharings a new set of sharings of $3t+1$ triples, say $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t)\}_{i\in[3t+1]}$, are generated using the transformation protocol TripTrans (see Section 5), so that there exists polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ of degree at most $\frac{3t}{2}, \frac{3t}{2}$ and $3t$ respectively, where $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds for $i \in [3t+1]$. The transformation guarantees that if the input triples are multiplication triples then the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds. Here our goal is to verify if $\mathsf{Z}(\cdot) \stackrel{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ in an error-free manner, by leaking at most t values on the three polynomials during the verification. Since this leaves at least $\frac{t}{2}$ degree of freedom on each of the polynomials for an *honest* D, $\frac{t}{2}$ output shared multiplication triples can be computed (if the verification is successful) as $\{[\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t\}_{i\in[\frac{t}{2}]}$ where $\mathbf{a}^{(i)} = \mathsf{X}(\beta_i), \mathbf{b}^{(i)} = \mathsf{Y}(\beta_i)$ and $\mathbf{c}^{(i)} = \mathsf{Z}(\beta_i)$.

We continue with the discussion on the verification. If *each* party P_i can confirm that $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is a multiplication triple (i.e. $\mathsf{Z}(\alpha_i) = \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$ is true) on holding $\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i)$, then it can be concluded that the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true. This follows from the fact that the confirmation comes from at least $3t+1$ *honest* parties and the degree of the polynomials $\mathsf{X}(\cdot)$, $\mathsf{Y}(\cdot)$ is at most $\frac{3t}{2}$ and the degree of $\mathsf{Z}(\cdot)$ is at most $3t$. This further ensures that only t values on each polynomial are leaked to the adversary through the t corrupted parties.

However, due to asynchronicity, we cannot wait for the confirmation from all the parties in $\mathcal{P}$, as the wait may turn out to be endless. Fortunately, the synchronous round at the beginning rescues us from the deadlock.

In the synchronous round, every party P_i is asked to "non-verifiably" t-share a dummy multiplication triple, say $(f^{(i)}, g^{(i)}, h^{(i)})$ which will be used later to verify if $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is a multiplication triple without further participation of P_i. Here by non-verifiably we mean that neither the correctness of the t-sharing nor the fact that the shared triple is a multiplication triple is guaranteed if P_i is *corrupted*. The synchronous round ensures that a triple sharing is done non-verifiably on behalf of *every* party, since even if a corrupted party does not send the shares of the dummy triples to some party by the end of the round, the receiver can take some default value to complete the sharing. This leads to two important cases: each honest party's triple are "good", since they will be correctly t-shared and the triple will be a random multiplication triple. For a corrupted party, none of the above guarantees are provided. Assuming that $(f^{(i)}, g^{(i)}, h^{(i)})$ is a *good* triple, it can be easily verified if $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is a multiplication triple by computing the sharing of $\mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$ from the sharing of $\mathsf{X}(\alpha_i)$ and $\mathsf{Y}(\alpha_i)$ by using the Beaver's technique with the shared dummy triple $(f^{(i)}, g^{(i)}, h^{(i)})$, followed by publicly checking if the difference sharing of $\mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$ and $\mathsf{Z}(\alpha_i)$ is the sharing of 0. If the checking passes then $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ can be concluded as a multiplication triple. Further if P_i is *honest* then the verification leaks nothing more about $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ due to the dummy multiplication triple $(f^{(i)}, g^{(i)}, h^{(i)})$. If the checking fails, then the sharing of $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ are publicly reconstructed for its public verification. Note that in such a case, either P_i or D must be *corrupted* and thus the privacy of the triple is lost already. However, if $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is proven to be a non-multiplication triple then D is definitely corrupted in which case the protocol can be halted after outputting $\frac{t}{2}$ default sharing of multiplication triples.

When the shared triple $(f^{(i)}, g^{(i)}, h^{(i)})$ is *not* a good triple due to the reason that it is a non-multiplication triple (but t-shared correctly), the checking of the corresponding multiplication triple $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ might fail leading to its public reconstruction and verification. But in this case P_i is surely corrupted and thus loosing the privacy of the triple does not matter. Furthermore, the public verification of the multiplication triple will be successful for an *honest* D, implying that an honest D can *not* be disqualified. The case when the shared triple $(f^{(i)}, g^{(i)}, h^{(i)})$ is *not* a good triple due to the reason that it is not t-shared correctly is more intricate to handle. The problem might be during the reconstructions of the values while applying the Beaver's technique, as they may not be t-shared. We solve this problem by using a "variant" of OEC that concludes the reconstructed value upon receiving shares from *any* $3t + 1$ parties without further waiting. This however, might cause different parties to reconstruct different values. Therefore an ABA protocol is run to reach agreement and then continue with the agreed value. We note that this does not alter the properties claimed before for the case when $(f^{(i)}, g^{(i)}, h^{(i)})$ is a good triple and correctly t-shared. The reason is that the parties will implicitly have a pre-agreement on the correct reconstructed value prior to the execution of ABA and by the property of ABA the pre-agreement is maintained.

Now combining the protocol $\mathsf{HybTripleSh}$ with the the triple extraction protocol $\mathsf{TripExt}$, we get an error-free protocol $\mathsf{HybPrePro}$ for the preprocessing phase, which can generate t-sharing of random multiplication triples, without any error. In the protocol $\mathsf{HybPrePro}$, instead of generating $c_M + c_R$ random multiplication triples, we will generate $c_M + c_R + c_I(t + 1)$ random multiplication triples. Then by using the additional $c_I(t + 1)$ sharings, we enforce "input provision" from all the n parties by using the method of [4]. Note that now the preprocessing phase protocol and the input phase protocol start parallely, so that both can use the first synchronous round. While the preprocessing phase protocol uses it for the error-free generation of the triples, the input phase protocol uses it to get the inputs from all the n parties. As the values shared by the parties in these two protocols are independent, this will not cause any problem. Finally, the protocol $\mathsf{CircEval}$ is used for the computation phase. Thus we get a perfectly secure AMPC protocol called $\mathsf{HybridAMPC}$, whose properties are stated in Theorem 8.1.

Theorem 8.1. *Let $f : \mathbb{F}^{c_I} \rightarrow \mathbb{F}^n$ be a publicly known function expressed as an arithmetic circuit C over a finite field $\mathbb{F}$, where $|\mathbb{F}| > 2n$, consisting of c_I, c_O, c_M and c_R number of input, output, multiplication and random gates respectively. Given that the first communication round is a synchronous round, protocol* $\mathsf{HybridAMPC}$ *is a perfectly secure protocol to securely compute f, for every possible* Adv *and for every possible scheduler. The protocol requires a private communication of $\mathcal{O}((c_O + c_M + c_R)n + c_I n^2)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$, along with two invocations to* ACS *and n^2 invocations to ABA.*

References

[1] I. Abraham, D. Dolev, and J. Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine agreement with optimal resilience. In R. A. Bazzi and B. Patt-Shamir, editors, *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 405–414. ACM, 2008.

[2] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Verlag, 1991.

[3] Z. Beerliová-Trubíniová and M. Hirt. Efficient multi-party computation with dispute control. In S. Halevi and T. Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 305–328. Springer Verlag, 2006.

[4] Z. Beerliová-Trubíniová and M. Hirt. Simple and efficient perfectly-secure asynchronous MPC. In K. Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 376–392. Springer Verlag, 2007.

[5] Z. Beerliová-Trubíniová and M. Hirt. Perfectly-secure MPC with linear communication complexity. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.

[6] Z. Beerliová-Trubíniová, M. Hirt, and J. B. Nielsen. On the theoretical gap between synchronous and asynchronous mpc protocols. In A. W. Richa and R. Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 211–218. ACM, 2010.

[7] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 52–61. ACM, 1993.

[8] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In J. Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[9] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In J. H. Anderson, D. Peleg, and E. Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 183–192. ACM, 1994.

[10] E. Ben-Sasson, S. Fehr, and R. Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 663–680. Springer, 2012.

[11] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In T. Kameda, J. Misra, J. Peters, and N. Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984.

[12] G. Bracha. An O(log n) expected rounds randomized Byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987.

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

[14] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

[15] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In J. Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 1999.

[16] I. Damgård, Y. Ishai, and M. Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[17] I. Damgård, Y. Ishai, M. Krøigaard, J. B. Nielsen, and A. Smith. Scalable multiparty computation with nearly optimal work and resilience. In D. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 241–261. Springer, 2008.

[18] I. Damgård and J. B. Nielsen. Scalable and unconditionally secure multiparty computation. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer Verlag, 2007.

[19] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[20] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[21] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 699–710. ACM, 1992.

[22] M. Hirt and U. M. Maurer. Robustness for free in unconditional multi-party computation. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2001.

[23] M. Hirt, U. M. Maurer, and B. Przydatek. Efficient secure multi-party computation. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 143–161. Springer, 2000.

[24] M. Hirt, J. B. Nielsen, and B. Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In R. Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 322–340. Springer, 2005.

[25] J. Katz and C. Y. Koo. Round-efficient secure computation in point-to-point networks. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 311–328. Springer, 2007.

[26] C. Y. Koo. *Studies on Fault-Tolerant Broadcast and Secure Computation*. PhD thesis, University of Maryland, 2007.

[27] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North-Holland Publishing Company, 1978.

[28] A. Patra, A. Choudhary, and C. Pandu Rangan. Communication efficient perfectly secure VSS and MPC in asynchronous networks with optimal resilience. In D. J. Bernstein and T. Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 184–202. Springer Verlag, 2010. Full version available at [29].

[29] A. Patra, A. Choudhury, and C. Pandu Rangan. Efficient asynchronous verifiable secret sharing and multiparty computation. Cryptology ePrint Archive, Report 2010/007, 2010. A preliminary version appeared as [28].

[30] A. Patra and C. Pandu Rangan. Brief announcement: communication efficient asynchronous Byzantine agreement. In A. W. Richa and R. Guerraoui, editors, *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010*, pages 243–244. ACM, 2010.

[31] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Trading players for efficiency in unconditional multiparty computation. In S. Cimato, C. Galdi, and G. Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2002.

[32] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In D. S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 73–85. ACM, 1989.

[33] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[34] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In B. K. Roy and E. Okamoto, editors, *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings*, volume 1977 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2000.

A Known Protocols

A.1 Private Reconstruction of a d-shared Value using OEC [7, 29]

Let v be a value which is d-shared among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$, where $d < |\overline{\mathcal{P}}| - 2t$. The goal is to make some party, say P_R, to reconstruct the value v robustly. In the synchronous setting, this can be achieved by asking every

party in $\overline{\mathcal{P}}$ to send its share of v to P_R, who can apply the Reed-Solomon (RS) error correction [27] algorithm on the received shares to reconstruct the value. Given the condition that $d < (|\overline{\mathcal{P}}| - 2t)$, the reconstruction will be robust, even if at most t corrupted parties in $\overline{\mathcal{P}}$ send incorrect or no shares. In an asynchronous setting, achieving the same requires a bit of trick as explained in the OEC protocol of [13].

The intuition behind the OEC is that P_R keeps waiting till it receives $d + t + 1$ shares which are d-consistent. This step requires applying the RS error correction algorithm repeatedly. Let $\mathsf{RSDec}(d, r, \mathcal{I})$ denote an RS error correction procedure, which takes as input a set $\mathcal{I}$ of shares (contains possibly some incorrect shares) of a d-shared value (that we would like to reconstruct) and outputs a polynomial of degree at most d, by correcting at most r errors (incorrect shares) in $\mathcal{I}$. Coding theory [27] says that RSDec can correct upto r errors in $\mathcal{I}$ and correctly output the original polynomial provided that $|\mathcal{I}| \geq d + 2r + 1$. There are several efficient implementations of RSDec (for example, the Berlekamp-Welch algorithm [27]). Once P_R receives $d + t + 1$ consistent shares that lie on a unique polynomial $\overline{p}(\cdot)$ (returned by RSDec) of degree at most d, then P_R outputs $\overline{p}(0)$ as v. The correctness follows from the fact that at least $d + 1$ shares out of the $d + t + 1$ consistent shares are from the honest parties in $\overline{\mathcal{P}}$ and those $d + 1$ shares uniquely define the original polynomial $p(\cdot)$ implying that $\overline{p}(\cdot) = p(\cdot)$. Note that the corrupted parties in $\overline{\mathcal{P}}$ may send incorrect shares to P_R or may not send any value. But the shares of at least $|\overline{\mathcal{P}}| - t \geq (d+t+1)$ honest parties in the set $\overline{\mathcal{P}}$ are d-consistent and they will eventually reach to P_R. As mentioned in [7, 29], the above procedure is nothing but applying the RS error correction algorithm in an "online" fashion.

The above steps are formally described in the protocol OEC presented in Fig. 5. The current description is inspired from [13] (skipping several other formal details).

Figure 5: Protocol OEC.

$\mathsf{OEC}(P_R, d, [v]_d^{\overline{\mathcal{P}}})$

1. The following code is executed by every party $P_i \in \overline{\mathcal{P}}$:
   - Send the share v_i of v to the party P_R and terminate.
2. For $r = 0, \ldots, t$, party P_R executes the following code during the iteration r:
   (a) Let $\mathcal{W}$ denote the set of parties in $\overline{\mathcal{P}}$ from whom P_R has received the shares and $\mathcal{I}_r$ denote the shares received from the parties in $\mathcal{W}$, when $\mathcal{W}$ contains exactly $d + t + 1 + r$ parties.
   (b) Wait until $|\mathcal{W}| \geq d + t + 1 + r$. Execute $\mathsf{RSDec}(d, r, \mathcal{I}_r)$. If no polynomial of degree at most d is obtained at the end of RSDec, then skip the next step and proceed to the next iteration. Otherwise, let $\overline{p}_r(\cdot)$ be the polynomial of degree at most d obtained at the end of RSDec.
   (c) If for at least $d+t+1$ shares $v_i \in \mathcal{I}_r$ it holds that $\overline{p}_r(\alpha_i) = v_i$, then output $\overline{p}_r(\cdot)$ and $\overline{p}_r(0)$ and terminate. Otherwise, proceed to the next iteration.

The properties of the protocol OEC are stated in Theorem A.1.

Theorem A.1. *Given $[v]_d^{\overline{\mathcal{P}}}$ for $\overline{\mathcal{P}} \subseteq \mathcal{P}$ and $d < |\overline{\mathcal{P}}| - 2t$, let the sharing is done using polynomial $p(\cdot)$ of degree at most d. Then for every possible Adv and for every possible scheduler, protocol OEC achieves:*

(1) TERMINATION*: Every honest party in $\overline{\mathcal{P}}$ will eventually terminate the protocol. Moreover, if P_R is honest then P_R will also eventually terminate the protocol.* **(2)** CORRECTNESS*: Party P_R will output $p(\cdot)$ and v.* **(3)** PRIVACY*: If P_R is honest then Adv obtains no additional information about v.* **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n)$ elements from $\mathbb{F}$.*

PROOF: The TERMINATION property is argued as follows. The honest parties in $\overline{\mathcal{P}}$ will terminate the protocol trivially after sending their shares of v to P_R. We now argue that (an honest) P_R will terminate the protocol as well. Let Adv corrupts $\hat{r}$ parties in $\overline{\mathcal{P}}$, where $\hat{r} \leq t$. Further assume $\hat{r}_1$ corrupted parties send wrong values and $\hat{r}_2$ corrupted parties send nothing ever, subject to $\hat{r}_1 + \hat{r}_2 = \hat{r}$. Consider the $(t - \hat{r}_2)$th iteration; since $\hat{r}_2$ parties in $\overline{\mathcal{P}}$ never send any value, P_R will receive $d + t + 1 + t - \hat{r}_2$ distinct values on the polynomial $p(\cdot)$, of which $\hat{r}_1$ are corrupted. Since $|\mathcal{I}_{t-\hat{r}_2}| = d + t + 1 + t - \hat{r}_2 \geq d + 2\hat{r}_1 + 1$, the algorithm RSDec will correct $\hat{r}_1$ errors and will return $\overline{p}_{t-\hat{r}_2}(\cdot) = p(\cdot)$ during the $(t - \hat{r}_2)$th iteration. Therefore the protocol will terminate at the latest after $(t - \hat{r}_2)$th iteration.

To argue CORRECTNESS, assume that the protocol terminates during the rth iteration and P_R outputs $\bar{p}_r(0)$ such that the polynomial $\bar{p}_r(\cdot)$ is consistent with $d+t+1$ shares from $\mathcal{I}_r$. To prove the correctness, we now show that $\bar{p}_r(\cdot) = p(\cdot)$. However, the equality follows from the fact that at least $d+1$ shares in $\mathcal{I}_r$ belong to the honest parties and thus they lie on $p(\cdot)$ as well. In other words, these $d+1$ shares are the common points of the two polynomials which are of degree at most d.

The PRIVACY is argued as follows. It is easy to see that if P_R is honest, then Adv gets no additional information about v. Since the (honest) parties in $\overline{\mathcal{P}}$ privately send their shares to P_R, no additional information about v or the values of $p(\cdot)$ is revealed to Adv during OEC.

In the protocol, each party in $\overline{\mathcal{P}}$ privately sends its share to P_R, causing a private communication of $\mathcal{O}(n)$ elements from $\mathbb{F}$. □

A.2 Batch Public Reconstruction of Several d-shared Values

Let $u^{(1)}, \ldots, u^{(t+1)}$ be d-shared among a set of parties $\overline{\mathcal{P}} \subseteq \mathcal{P}$, where $d < |\overline{\mathcal{P}}| - 2t$; i.e., $[u^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [u^{(t+1)}]_d^{\overline{\mathcal{P}}}$ is available. The goal is to make *every* party in $\mathcal{P}$ to reconstruct $u^{(1)}, \ldots, u^{(t+1)}$, such that the protocol requires a private communication of $\mathcal{O}(n^2)$ (and no broadcast communication). Protocol BatRecPubl uses the idea of [18] to solve this problem and is presented in Fig. 6.

Figure 6: Protocol BatRecPubl for the public reconstruction of $t+1$ d-shared Values.

BatRecPubl$(\mathcal{P}, d, [u^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [u^{(t+1)}]_d^{\overline{\mathcal{P}}})$

Let $u(x) = \sum_{i=1}^{t+1} u^{(i)} x^{i-1}$ be the polynomial of degree at most t and $v^{(i)} = u(\alpha_i)$ for $i \in [n]$. Then we have $[u(0)]_t^{\mathcal{P}} = (v^{(1)}, \ldots, v^{(n)})$.

1. For each $j \in [n]$, the parties in $\overline{\mathcal{P}}$ (locally) compute $[v^{(j)}]_d^{\overline{\mathcal{P}}}$ as:
$$[v^{(j)}]_d^{\overline{\mathcal{P}}} = [u^{(1)}]_d^{\overline{\mathcal{P}}} + \alpha_j \cdot [u^{(2)}]_d^{\overline{\mathcal{P}}} + \alpha_j^2 \cdot [u^{(3)}]_d^{\overline{\mathcal{P}}} + \ldots + \alpha_j^t \cdot [u^{(t+1)}]_d^{\overline{\mathcal{P}}}.$$
2. For each $i \in [n]$, party P_i and the parties in $\overline{\mathcal{P}}$ execute OEC$(P_i, d, [v^{(i)}]_d^{\overline{\mathcal{P}}})$ to enable P_i to privately reconstruct $v^{(i)}$.
3. For each $i \in [n]$, party P_i executes the following code:
   (a) Execute an instance OEC$(P_i, t, [u(0)]_t^{\mathcal{P}})$ to reconstruct the polynomial $u(x)$. For each $j \in [n]$, participate in the instance OEC$(P_j, t, [u(0)]_t^{\mathcal{P}})$ to enable the party P_j to reconstruct the polynomial $u(x)$.
   (b) On reconstructing the polynomial $u(x)$ at the end of the instance OEC$(P_i, t, [u(0)]_t^{\mathcal{P}})$, output the $t+1$ coefficients of $u(x)$ and terminate.

The properties of the protocol BatRecPubl are stated in Theorem A.2.

Theorem A.2. *Given $[u^{(1)}]_d^{\overline{\mathcal{P}}}, \ldots, [u^{(t+1)}]_d^{\overline{\mathcal{P}}}$ for $\overline{\mathcal{P}} \subseteq \mathcal{P}$ and $d < |\overline{\mathcal{P}}| - 2t$, for every possible* Adv *and for every possible scheduler, protocol* BatRecPubl *achieves:*

(1) TERMINATION*: Every honest party in $\mathcal{P}$ will eventually complete the protocol.* **(2)** CORRECTNESS*: Every honest party in $\mathcal{P}$ will output $u^{(1)}, \ldots, u^{(t+1)}$.* **(3)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$.*

PROOF: The TERMINATION property of the protocol follows from the termination property of OEC (Theorem A.1). For CORRECTNESS, we note that the correctness property of OEC (Theorem A.1) implies that each honest party $P_i \in \mathcal{P}$ will eventually reconstruct $v^{(i)}$ at the end of the instance OEC$(P_i, d, [v^{(i)}]_d^{\overline{\mathcal{P}}})$. Moreover, $(v^{(1)}, \ldots, v^{(n)}) = [u(0)]_t^{\mathcal{P}}$; so from the correctness property of OEC, each honest party P_i will output the polynomial $u(x)$ at the end of OEC$(P_i, t, [u(0)]_t^{\mathcal{P}})$ and hence will output $u^{(1)}, \ldots, u^{(t+1)}$, which are the coefficients of $u(x)$. The COMMUNICATION COMPLEXITY follows from the communication complexity of OEC (Theorem A.1) and the fact that $2n$ instances of OEC are executed. □

Unlike OEC that enables the reconstruction of the polynomial used for the given input sharing, protocol BatRecPubl

allows to reconstruct only the d-shared values and not the individual polynomials that are used for the sharings. We further note that the above mentioned property of OEC is instrumental in the correctness of BatRecPubl, since the entire polynomial $u(x)$ is required to be reconstructed in BatRecPubl in order to output the coefficients of $u(x)$.

A.3 Batch Multiplication of ℓ Pairs of t-shared Values using Beaver's Technique

Protocol BatchBeaver is presented in Fig. 7.

Figure 7: Protocol to perform batch multiplication of ℓ pairs of t-shared values where $\ell \geq t+1$.

Protocol BatchBeaver$(\{([x^{(i)}]_t, [y^{(i)}]_t, [a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t)\}_{i\in[\ell]})$

1. For each $i \in [\ell]$, the parties in $\mathcal{P}$ locally compute $[e^{(i)}]_t$ and $[d^{(i)}]_t$, where $[e^{(i)}]_t = [x^{(i)}]_t - [a^{(i)}]_t = [x^{(i)} - a^{(i)}]_t$ and $[d^{(i)}]_t = [y^{(i)}]_t - [b^{(i)}]_t = [y^{(i)} - b^{(i)}]_t$.
2. The parties execute $\lceil \frac{2\ell}{t+1} \rceil$ instances of BatRecPubl and publicly reconstruct[a] $\{d^{(i)}, e^{(i)}\}_{i\in[\ell]}$. If ℓ is not an exact multiple of $t+1$, some default sharings are added to make ℓ exact multiple of $t+1$.
3. For each $i \in [\ell]$, the parties in $\mathcal{P}$ locally compute $[x^{(i)}y^{(i)}]_t = d^{(i)}e^{(i)} + e^{(i)}[b^{(i)}]_t + d^{(i)}[a^{(i)}]_t + [c^{(i)}]_t$ and terminate.

[a] A single instance of BatRecPubl can reconstruct $t+1$ values. To reconstruct 2ℓ values, $\lceil \frac{2\ell}{t+1} \rceil$ instances of BatRecPubl are required.

We now prove Theorem 3.1 (for the theorem statement see Sec. 3).

Proof of Theorem 3.1: The TERMINATION property follows from the termination property of BatRecPubl. The CORRECTNESS follows from the fact that for each $i \in [\ell]$, we have $x^{(i)}y^{(i)} = ((x^{(i)} - a^{(i)}) + a^{(i)})((y^{(i)} - b^{(i)}) + b^{(i)}) = d^{(i)}e^{(i)} + e^{(i)}b^{(i)} + d^{(i)}a^{(i)} + c^{(i)}$, where $e^{(i)} = x^{(i)} - a^{(i)}$ and $d^{(i)} = y^{(i)} - b^{(i)}$. The PRIVACY is argued as follows: the only step where the parties communicate is during the reconstruction of $d^{(i)}$s and $e^{(i)}$s. Now $e^{(i)} = x^{(i)} - a^{(i)}$ and the fact that $a^{(i)}$ is random and unknown to Adv implies that even after learning $e^{(i)}$, the value $x^{(i)}$ remains as secure as it was before from the view point of Adv. Similarly, as $b^{(i)}$ is random, even after learning $d^{(i)}$, the value $y^{(i)}$ remains as secure as before from the view point of Adv. This proves the privacy property. The COMMUNICATION COMPLEXITY follows from the communication complexity of BatRecPubl and the fact that $2\lceil \frac{\ell}{t+1} \rceil$ instances of BatRecPubl are executed in total and $(t+1) = \Theta(n)$. □

A.4 Agreement on a Common Set (ACS)

Protocol ACS is a well known protocol to allow the parties to agree on a common set of $n-t$ parties, say Com, where each party in Com satisfies a "property" Q, where Q has the following characteristics:

1. Every *honest* party will satisfy Q eventually, while a corrupted party may or may not choose to satisfy Q.
2. If some *honest* party $P_i \in \mathcal{P}$ finds some (probably corrupted) party $P_j \in \mathcal{P}$ to satisfy Q, then every other honest party in $\mathcal{P}$ will eventually find P_j to satisfy Q.

Protocol ACS is presented in Fig. 8.
The properties of the protocol ACS are stated in Theorem A.3.

Theorem A.3 ([7])**.** *Let Q be a property satisfying the above described conditions. Then for every possible* Adv *and for every possible scheduler, protocol* ACS *achieves:*
(1) TERMINATION*: All the honest parties terminate the protocol almost surely (i.e. with probability* 1*).* **(2)** CORRECTNESS*: Every honest party will output a set of $n-t$ parties* Com *such that every party in* Com *satisfies the property Q eventually.* **(3)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(poly(n))$.*

Figure 8: Protocol for agreement on a common set of parties satisfying some property Q.

Protocol ACS

CODE FOR THE PARTY P_i — Every party in $\mathcal{P}$ executes this code:

1. For each $P_j \in \mathcal{P}$ such that $Q(j) = 1$ (i.e. P_j satisfies the property Q), participate in ABA_j with input 1. Here for $j \in [n]$, ABA_j denotes the instance of an ABA protocol executed for $P_j \in \mathcal{P}$ to decide whether P_j will be in the common set.
2. Upon terminating $(n-t)$ instances of ABA with output 1, enter input 0 to all other instances of ABA, for which you have not entered a value yet.
3. Upon terminating all the n instances of ABA, let your SubSet_i be the set of all indices j for which ABA_j had output 1.
4. Set Com to be the set of parties corresponding to the indices in SubSet_i, output Com and terminate.

A.5 Asynchronous Broadcast

We recall the Bracha's asynchronous broadcast protocol from [13] and present it in Fig. 9.

Figure 9: Bracha's asynchronous broadcast protocol with $n = 3t + 1$ parties.

Protocol A-Cast

CODE FOR THE SENDER Sen (WITH INPUT m) — only Sen executes this code:

1. Send the message $(\texttt{MSG}, m)$ privately to all the parties.

CODE FOR THE PARTY P_i — every party in $\mathcal{P}$ executes this code:

1. Upon receiving a message $(\texttt{MSG}, m)$ from Sen, send the message $(\texttt{ECHO}, m)$ privately to all the parties.
2. Upon receiving $(n-t)$ messages $(\texttt{ECHO}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ privately to all the parties.
3. Upon receiving $(t+1)$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{READY}, m^\star)$ privately to all the parties.
4. Upon receiving $(n-t)$ messages $(\texttt{READY}, m^\star)$ that agree on the value of $m^\star$, send the message $(\texttt{OK}, m^\star)$ privately to all the parties, output $m^\star$ and terminate.

The properties of the protocol are stated in Theorem A.4.

Theorem A.4 ([13])**.** *Let* $\mathsf{Sen} \in \mathcal{P}$ *has a message* m *of size* ℓ*, which it wants to reliably send to all the parties in* $\mathcal{P}$*. Then for every possible* Adv *and every possible scheduler, protocol* $\mathsf{A\text{-}Cast}$ *achieves:*

(1) TERMINATION*: If* Sen *is honest then then all the honest parties eventually terminate. Moreover, even if* Sen *is corrupted and some honest party terminates, then all the honest parties eventually terminate.* **(2)** CORRECTNESS*: All the honest parties upon terminating output* $m^\star$*. Moreover, if* Sen *is honest then* $m^\star = m$*.* **(3)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of* $\mathcal{O}(n^2\ell)$*.*

B Sharing Protocol Sh and its Properties

In this section, we first recall the sharing protocol of the perfectly secure AVSS scheme of [28, 29]. Then we present the protocol Sh with the required modification and extension. We finally discuss how to share ℓ values where $\ell \geq t + 1$ and keep the broadcast communication independent of ℓ.

B.1 AVSS Protocol of [28, 29] for $2t$-sharing a Single Value

The goal of the sharing protocol of the AVSS scheme of [28, 29] is to enable D to $2t$-share (and *not* t-share) a secret s and is divided into three stages. The first stage is the *distribution stage*, where D selects a bi-variate polynomial $F(x, y)$ of *different* degrees in x and y; namely the degree of x is at most $2t$ and the degree of y is at most t. We call such a polynomial a bi-variate polynomial of degree-$(2t, t)$. The polynomial $F(x, y)$ is an otherwise random polynomial except that $F(0, 0) = s$. The dealer then hands the ith row polynomial $f_i(x)$ of degree at most $2t$ and the ith column polynomial $g_i(y)$ of degree at most t to the party P_i, where $f_i(x) \stackrel{def}{=} F(x, \alpha_i)$ and $g_i(y) \stackrel{def}{=} F(\alpha_i, y)$. The second stage is the *verification stage*, where the parties interact and verify that there exists a set of $3t + 1$ parties called CORE and a bi-variate polynomial of degree-$(2t, t)$, say $F'(x, y)$, such that the following holds:

- Every (honest) party P_i in CORE has a row polynomial, say $f'_i(x)$, of degree at most $2t$ and a column polynomial, say $g'_i(y)$, of degree at most t, such that $f'_i(x) = F'(x, \alpha_i)$ and $g'_i(y) = F'(\alpha_i, y)$ holds. We often say in this case that the row and column polynomials are consistent with $F'(x, y)$.
- If D is *honest* then $F'(x, y) = F(x, y)$ and hence $f'_i(x) = f_i(x)$ and $g'_i(y) = g_i(y)$ holds.

The last stage called *completion stage* ensures that even the (honest) parties *outside* the CORE possess their respective *column polynomial*, consistent with $F'(x, y)$; i.e. *every* $P_i \in \mathcal{P}$ possesses $g'_i(y)$, where $g'_i(y) = F'(\alpha_i, y)$. Once this is done, we say that D has committed the polynomial $F'(x, y)$ and the secret s', where $s' = F'(0, 0)$. Now s' is $2t$-shared through the polynomial $f'_0(x)$ with $f'_0(x) = F'(x, 0)$ of degree at most $2t$ and every party P_i will have its share s'_i of s', where $s'_i = f'_0(\alpha_i) = g'_i(0)$. For an *honest* D, we have $s' = s$. Furthermore, the privacy of the $2t$-sharing of s follows from the fact that the view of Adv in the protocol leaves $(t + 1)$ "degree of freedom" in $F(x, y)$. Informally this is because $(t + 1)(2t + 1)$ distinct points are required to know $F(x, y)$, but only $t(2t + 1) + t$ distinct points are revealed to Adv through the t row and column polynomials. We now recall the sharing protocol of AVSS scheme of [29, 28] in Fig. 10. Prior to that, we need to recall the following definition and algorithm presented originally in [13, 7].

Definition B.1 ((n, t)-star[13, 7])**.** *Let G be an undirected graph with the n parties in $\mathcal{P}$ as its vertex set. We say that a pair $(\mathcal{C}, \mathcal{D})$ of sets with $\mathcal{C} \subseteq \mathcal{D} \subseteq \mathcal{P}$ is an (n, t)-star in G, if the following holds:*

1. *$|\mathcal{C}| \geq n - 2t$;*
2. *$|\mathcal{D}| \geq n - t$;*
3. *For every $P_j \in \mathcal{C}$ and every $P_k \in \mathcal{D}$ the edge (P_j, P_k) exists in G.*

In [7], the authors have presented an elegant and efficient algorithm for finding an (n, t)-star, provided the graph contains a clique of size $n - t$. The algorithm, called FindStar outputs either an (n, t)-star or the message `star-Not-Found` in polynomial time. Whenever the input graph contains a clique of size $n-t$, FindStar always outputs an (n, t)-star in the graph.

B.2 Protocol Sh for t-sharing $t + 1$ Values

The protocol Sh that creates t-sharing of secrets in $\vec{S} = (s^{(1)}, \ldots, s^{(t+1)})$ is obtained by making the following modification in the first step of the *distribution stage* of the protocol presented in Figure 10. Specifically, D now embeds $t + 1$ secrets from $\vec{S}$ in the polynomial $F(x, y)$ as follows:

– D selects a random bivariate polynomial $F(x, y)$ of degree-$(2t, t)$ over $\mathbb{F}$, such that $F(\beta_l, 0) = s^{(l)}$, for $l \in [t+1]$.

Additionally, in the *completion stage*, we add the following step after the step 1 to enable every party P_i to compute $f'_i(x)$:

Figure 10: The sharing protocol of the AVSS of [28, 29].

Distribution Stage

The following code is executed only by D:

1. On having a secret s, select a random bivariate polynomial $F(x,y)$ of degree-$(2t,t)$ over $\mathbb{F}$, such that $F(0,0) = s$. Let $f_i(x) \stackrel{def}{=} F(x,\alpha_i)$ and $g_i(y) \stackrel{def}{=} F(\alpha_i, y)$, for $i \in [n]$.
2. For every $i \in [n]$, send the row polynomial $f_i(x)$ and the column polynomial $g_i(y)$ to the party P_i and terminate.

Verification Stage

i. CODE FOR P_i (FOR PAIR-WISE CONSISTENCY CHECKING): Every party in $\mathcal{P}$ (including D) executes the following code.

1. Wait to receive from D a row polynomial, say $f'_i(x)$, and a column polynomial, say $g'_i(y)$, of degree at most $2t$ and t respectively. Upon receiving, send f'_{ij} and g'_{ij} to the party P_j, for every $j \in [n]$, where $f'_{ij} \stackrel{def}{=} f'_i(\alpha_j)$ and $g'_{ij} \stackrel{def}{=} g'_i(\alpha_j)$.
2. Upon receiving f'_{ji} and g'_{ji} from the party P_j, check if $f'_i(\alpha_j) \stackrel{?}{=} g'_{ji}$ and $g'_i(\alpha_j) \stackrel{?}{=} f'_{ji}$. If both the equalities hold, then broadcast the message $\texttt{OK}(P_i, P_j)$.
3. Construct the undirected consistency graph G_i with $\mathcal{P}$ as the vertex set. Add an edge (P_j, P_k) in G_i upon receiving the message $\texttt{OK}(P_k, P_j)$ and $\texttt{OK}(P_j, P_k)$ from the broadcast of P_k and P_j respectively.

ii. CODE FOR D (FOR GENERATING CORE). Only D executes the following code: Let G_{D} be the consistency graph of D.

1. After every new receipt of some $\texttt{OK}(\star,\star)$ message, update G_{D}. If a new edge is added to G_{D}, then execute FindStar on G_{D}. Let there are α distinct (n,t)-stars that are found in the past, from different executions of FindStar, where $\alpha \geq 0$.
   (a) If an (n,t)-star is found from the current execution of FindStar that is distinct from all the previously obtained α (n,t)-stars, do the following:
      i. Call the new (n,t)-star as $(\mathcal{C}_{\alpha+1}, \mathcal{D}_{\alpha+1})$.
      ii. Construct a set $\mathcal{F}_{\alpha+1}$ and add P_j to $\mathcal{F}_{\alpha+1}$ if P_j has at least $2t+1$ neighbours from the set $\mathcal{C}_{\alpha+1}$ in G_{D}.
      iii. Construct a set $\mathcal{E}_{\alpha+1}$ and add P_k to $\mathcal{E}_{\alpha+1}$ if P_k has at least $3t+1$ neighbours from the set $\mathcal{F}_{\alpha+1}$ in G_{D}.
      iv. For each $\beta \in [\alpha]$, update the existing $\mathcal{F}_\beta$ and $\mathcal{E}_\beta$ sets (constructed earlier) as follows:
         A. Add P_j to $\mathcal{F}_\beta$, if $P_j \notin \mathcal{F}_\beta$ and P_j has now at least $2t+1$ neighbours from the set $\mathcal{C}_\beta$ in G_{D}.
         B. Add P_k to $\mathcal{E}_\beta$, if $P_k \notin \mathcal{E}_\beta$ and P_k has now at least $3t+1$ neighbours from the set $\mathcal{F}_\beta$ in G_{D}.
   (b) If an (n,t)-star that has been already found in the past is obtained from the current execution of FindStar, then execute the steps (a).iv(A-B) to update the existing $\mathcal{F}_\beta$s and $\mathcal{E}_\beta$s for $\beta \in [\alpha]$.

   Let $(\mathcal{E}_\gamma, \mathcal{F}_\gamma)$ be the first pair among the generated $(\mathcal{E}_\beta, \mathcal{F}_\beta)$s, such that $|\mathcal{E}_\gamma| \geq 3t+1$ and $|\mathcal{F}_\gamma| \geq 3t+1$. Assign $\mathsf{CORE} = \mathcal{E}_\gamma$ and broadcast $((\mathcal{C}_\gamma, \mathcal{D}_\gamma), (\mathcal{E}_\gamma, \mathcal{F}_\gamma))$.

iii. CODE FOR P_i (FOR VERIFYING CORE): Every party in $\mathcal{P}$ (including D) executes the following code.

1. Wait to receive $((\mathcal{C}_\gamma, \mathcal{D}_\gamma), (\mathcal{E}_\gamma, \mathcal{F}_\gamma))$ from the broadcast of D, such that $|\mathcal{E}_\gamma| \geq 3t+1$ and $|\mathcal{F}_\gamma| \geq 3t+1$.
2. Wait until $(\mathcal{C}_\gamma, \mathcal{D}_\gamma)$ becomes an (n,t)-star in the consistency graph G_i. For this, wait to receive the corresponding OK messages from the broadcast of the parties in $\mathcal{C}_\gamma$ and $\mathcal{D}_\gamma$.
3. Wait till every $P_j \in \mathcal{F}_\gamma$ has at least $2t+1$ neighbours from $\mathcal{C}_\gamma$ and every $P_k \in \mathcal{E}_\gamma$ has at least $3t+1$ neighbours from $\mathcal{F}_\gamma$ in G_i.

Once the above conditions are satisfied, set $\mathsf{CORE} = \mathcal{E}_\gamma$ and terminate.

Completion Stage

Let $F'(x,y)$ be the bi-variate polynomial of degree-$(2t,t)$, committed by D to the parties in CORE, such that the row and column polynomials of the parties in CORE are consistent with $F'(x,y)$. For every $P_i \notin \mathsf{CORE}$, let $f'_i(x) \stackrel{def}{=} F'(x,\alpha_i)$ and $g'_i(y) \stackrel{def}{=} F'(\alpha_i, y)$. The parties in $\mathcal{P}$ execute the following code to have the secret $F'(0,0)$ $2t$-shared though the polynomial $f'_0(x)$, where $f'_0(x) = F'(x,0)$:

1. For every $P_i \notin \mathsf{CORE}$, the parties in CORE and party P_i execute $\mathsf{OEC}(P_i, t, [g'_i(0)]_t^{\mathsf{CORE}})$, to enable P_i to reconstruct the polynomial $g'_i(y)$; for this every party $P_j \in \mathsf{CORE}$ sends $f'_j(\alpha_i)$ to P_i, where P_j holds the row polynomial $f'_j(x)$.
2. Party P_i outputs $g'_i(0)$ (which is the same as $f'_0(\alpha_i)$) as his share of the secret $F'(0,0)$ and terminate.

– For every $P_i \notin \mathsf{CORE}$, the parties in $\mathcal{P}$ and party P_i execute $\mathsf{OEC}(P_i, 2t, [f'_i(0)]^{\mathcal{P}}_{2t})$, to enable P_i to privately reconstruct the polynomial $f'_i(x)$; for this every party $P_j \in \mathcal{P}$ sends $g'_j(\alpha_i)$ to P_i, where P_j holds the column polynomial $g'_j(y)$.

Finally, we modify the last step of the completion stage as follows to enable the parties to output the shares corresponding to the t-sharing of the secrets:

– For each $i \in [n]$, party P_i outputs his share $\vec{S'_i}$, where $\vec{S'_i} = (f'_i(\beta_1), \ldots, f'_i(\beta_{t+1}))$ and terminate.

We appeal to the proof of the properties of the sharing protocol of the perfectly secure AVSS scheme of [29, 28] for proving the properties of our protocol Sh, stated in Theorem 4.1 (see Section 4 for the theorem statement).

Proof of Theorem 4.1:

TERMINATION: From the termination property of the AVSS scheme of [29, 28], it follows that if D is *honest* then the parties will eventually complete the sharing phase; moreover even if D is *corrupted* and some honest party completes the sharing phase, then every other honest party will also eventually do the same. Now the completion of the sharing phase of the AVSS scheme of [29, 28] implies that every party in $\mathcal{P}$ will eventually have its column polynomial. It now follows by the property of OEC that every honest party will eventually compute its row polynomial $f'_i(x)$ since each $f'_i(0)$ is already $2t$-shared among the parties via the column polynomials. After the computation of row polynomial, the parties output their shares of the secrets and terminate the protocol. This completes the proof of the termination property.

CORRECTNESS: The correctness property of the AVSS scheme of [29, 28] ensures that if some honest party terminates the sharing protocol, then there exists a unique bi-variate polynomial, say $F'(x, y)$, of degree-$(2t, t)$, such that every honest party $P_i \in \mathcal{P}$ has the column polynomial $g'_i(y)$ of degree at most t, where $g'_i(y) = F'(\alpha_i, y)$; moreover if D is honest then $F'(x, y) = F(x, y)$ and hence $g'_i(y) = g_i(y)$. Our Sh protocol ensure the same. We define D's committed secret in Sh as $\vec{S'} = (s'^{(1)}, \ldots, s'^{(t+1)})$, where $s'^{(l)}$ is the constant term of the polynomial $g'_{\beta_l}(y)$ of degree at most t and $g'_{\beta_l}(y) = F'(\beta_l, y)$. By the additional step added in the protocol Sh, every party P_i eventually computes its row polynomial $f'_i(x)$ of degree at most $2t$, where $f'_i(x) = F'(x, \alpha_i)$. The correctness of this step follows by the correctness of the OEC protocol. Now it is easy to see that the ith share $s'^{(l)}_i = g'_{\beta_l}(\alpha_i)$ of the committed secret $s'^{(l)}$ is the same as $F'(\beta_l, \alpha_i)$ and thus $f'_i(\beta_l)$. So every party P_i can compute its shares of the committed secrets by evaluating $f'_i(x)$ at $x = \beta_1, \ldots, \beta_{t+1}$ and hence $\vec{S'}$ will be t-shared. Moreover, it is easy to see that if D is *honest* then $\vec{S'} = \vec{S}$ will hold. This completes the proof of the correctness.

PRIVACY: For privacy, we consider an *honest* D. Without loss of generality, let $P_1, \ldots, P_t$ be under the control of Adv. It is easy to see that in the protocol Sh, Adv will obtain t row and column polynomials, namely $f_1(x), \ldots, f_t(x), g_1(y), \ldots, g_t(y)$. Each row polynomial is of degree at most $2t$ and provides $2t+1$ distinct points on $F(x, y)$. On the other hand, each column polynomial is of degree at most t and can provide $t+1$ distinct points; however out of these $t + 1$ points, t points are the common points with the row polynomials and already counted in the view of Adv. So each column polynomial adds one new point on $F(x, y)$ to the view of Adv. This in total provides Adv with $t(2t + 1) + t$ distinct points on $F(x, y)$. Moreover, the points $F(\beta_l, 0)$ for $l \in [t + 1]$, which are the secrets in $\vec{S}$ are not distributed in the protocol.

Now $F(x, y)$ is of degree-$(2t, t)$ and it requires the knowledge of $(t + 1)(2t + 1)$ distinct points on $F(x, y)$ to uniquely interpolate the polynomial; this gives $(2t+1)(t+1)-t(2t+1)-t = t+1$ degree of freedom for $F(x, y)$. More specifically, from the view point of the adversary, for every possible choice of the $t + 1$ secrets, there exists a unique bi-variate polynomial of degree-$(2t, t)$, which is consistent with the $t + 1$ secrets (which constitute $t + 1$ distinct points on the polynomial) and the $t(2t + 1) + t$ distinct points known to Adv in the protocol. This implies that the information (namely the row and column polynomials) obtained by Adv is distributed independently of the $t + 1$ secrets.

COMMUNICATION COMPLEXITY: The communication complexity directly follows from the communication complexity analysis of the AVSS scheme of [29, 28]. □

B.3 Sharing more than $t+1$ Values Together using Sh

If D wants to t-share ℓ secrets, where $\ell > t+1$, then it can form $\frac{\ell}{t+1}$ groups, each consisting of $t+1$ secrets (without loss of generality, we assume that ℓ is a multiple of $t+1$) and execute an instance of Sh for each group. This requires a private communication of $\mathcal{O}(n\ell)$ field elements, as $t+1 = \Theta(n)$. To keep the total broadcast communication independent of ℓ, we follow the same trick as used in [29, 28] for keeping the broadcast communication independent of ℓ while generating $2t$-sharing of ℓ secrets. We first note that the broadcast communication occurs only in the verification stage of the protocol. We execute all instances of the Sh protocol (each handling $t+1$ secrets) in parallel so that each party broadcasts only when the *pre-condition* for the broadcast is satisfied in all the Sh instances. Specifically, we ensure that the parties (locally) construct a *single* consistency graph for all instances of Sh, instead of $\frac{\ell}{t+1}$ individual consistency graphs. For this, we ask every party P_i to broadcast the OK message involving party P_j (namely the $\texttt{OK}(P_i, P_j)$ message) only if the ith row polynomial is "pair-wise consistent" with the jth column polynomial and the ith column polynomial is pair-wise consistent with the jth row polynomial in *all* the $\frac{\ell}{t+1}$ instances of Sh. That is, P_i should check whether $f'_i(\alpha_j) \stackrel{?}{=} g'_{ji}$ and $g'_i(\alpha_j) \stackrel{?}{=} f'_{ji}$ holds in all the $\frac{\ell}{t+1}$ instances of Sh, before broadcasting the message $\texttt{OK}(P_i, P_j)$. Now a single consistency graph is constructed for all the $\frac{\ell}{t+1}$ instances and accordingly a single $\mathcal{C}, \mathcal{D}, \mathcal{E}$ and $\mathcal{F}$ set is broadcasted. The above trick keeps the broadcast communication to be $\mathcal{O}(n^2)$ for all the $\frac{\ell}{t+1}$ instances.

C Proof of Theorem 5.1

TERMINATION: This property follows from the termination property of BatchBeaver (see Theorem 3.1).

CORRECTNESS: By construction, it is ensured that the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ are of degree $\frac{3t}{2}, \frac{3t}{2}$ and $3t$ respectively and $\mathsf{X}(\alpha_i) = \mathbf{x}^{(i)}, \mathsf{Y}(\alpha_i) = \mathbf{y}^{(i)}$ and $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds for $i \in [3t+1]$. To argue the second statement in the correctness property, we first show that if the input triples are multiplication triple then $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds. For this, it is enough to show the multiplicative relation $\mathsf{Z}(\alpha_i) = \mathsf{X}(\alpha_i)\mathsf{Y}(\alpha_i)$, which is the same as $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}\mathbf{y}^{(i)}$, holds for $i \in [3t+1]$. For $i \in [\frac{3t}{2}+1]$, the relation $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}\mathbf{y}^{(i)}$ holds since we have $\mathbf{x}^{(i)} = x^{(i)}$, $\mathbf{y}^{(i)} = y^{(i)}$, $\mathbf{z}^{(i)} = z^{(i)}$ and the triple $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication triple by assumption. For $i \in [\frac{3t}{2}+2, 3t+1]$, we have $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}\mathbf{y}^{(i)}$ due to the correctness of the protocol BatchBeaver and the assumption that the triples used in BatchBeaver, namely $\left\{(x^{(i)}, y^{(i)}, z^{(i)})\right\}_{i \in [\frac{3t}{2}+2, 3t+1]}$ are multiplication triples. Proving the other way, that is, if $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true then all the input triples are multiplication triples is easy. Since $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$, it implies that $\mathbf{z}^{(i)} = \mathbf{x}^{(i)}\mathbf{y}^{(i)}$ for $i \in [3t+1]$. This trivially implies $\left\{(x^{(i)}, y^{(i)}, z^{(i)})\right\}_{i \in [\frac{3t}{2}+1]}$ are multiplication triples. On the other hand, if some triple in $\left\{(x^{(i)}, y^{(i)}, z^{(i)})\right\}_{i \in [\frac{3t}{2}+2, 3t+1]}$, say $(x^{(j)}, y^{(j)}, z^{(j)})$ is not a multiplication triple, then $(\mathbf{x}^{(j)}, \mathbf{y}^{(j)}, \mathbf{z}^{(j)})$ is not a multiplication triple as well (by the correctness of the Beaver's technique), which is a contradiction.

PRIVACY: First note that if Adv knows more than $\frac{3t}{2}$ input triples, then it knows all the three polynomials completely. Now to prove the privacy, we show that if Adv knows the input triple $(x^{(i)}, y^{(i)}, z^{(i)})$, then it also knows the output triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$. If $i \in [\frac{3t}{2}+1]$, this follows trivially since $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is the same as $(x^{(i)}, y^{(i)}, z^{(i)})$. Else if $i \in [\frac{3t}{2}+2, 3t+1]$, then Adv knows the t-sharing of $(x^{(i)}, y^{(i)}, z^{(i)})$ which is used to compute the t-sharing of $\mathbf{z}^{(i)}$ from the t-sharing of $\mathbf{x}^{(i)}$ and $\mathbf{y}^{(i)}$. Since the values $(\mathbf{x}^{(i)} - x^{(i)})$ and $(\mathbf{y}^{(i)} - y^{(i)})$ are disclosed during the computation of $\mathbf{z}^{(i)}$, Adv knows $\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$ and hence $\mathbf{z}^{(i)}$. So we proved that the knowledge of t' input triples allows Adv to know t' output triples. It now follows that Adv knows t' points on the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$.

COMMUNICATION COMPLEXITY: The communication complexity follows from the fact that one instance of BatchBeaver is invoked which requires a private communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$. □

D Properties of the Preprocessing Phase

D.1 Proof of Theorem 6.1

TERMINATION: We start with noting that D starts TripleSh by invoking an instance of Sh and the termination property of the protocol Sh (see Theorem 4.1) implies that if D is *honest*, then the instance of Sh will terminate. Once the instance of Sh is terminated, the remaining sub-protocols in TripleSh, namely the instance of Rand and the instances of BatRecPubl will eventually terminate. So the honest parties will terminate the protocol TripleSh when D is honest. On the other hand, if D is *corrupted* and some honest party has terminated the protocol TripleSh, then it implies that the honest party has terminated all the sub-protocols in TripleSh, namely the instance of Sh, the instance of Rand and the instances of BatRecPubl. The termination property of Sh (see Theorem 4.1) implies that in this case, every other honest party will eventually terminate the instance of Sh too. Subsequently, all the other sub-protocols, namely Rand and BatRecPubl will eventually terminate for each honest party. This proves that if some honest party terminates TripleSh, then every other honest party will eventually terminate TripleSh.

CORRECTNESS: We first consider the easy case of an *honest* D. The correctness property of Sh (see Theorem 4.1) implies that the triples in $\vec{S}$ and $\vec{S}_{sac}$ will be t-shared among the parties. Since the triples are indeed multiplication triples, the condition $\gamma^{(i)} = 0$ is true for every $i \in [\ell]$ and any r. It thus follows that the multiplication triples in $\vec{S}$ will be t-shared.

We next consider a *corrupted* D and assume that some honest party has terminated the protocol TripleSh. This implies that it has terminated all the sub-protocols in TripleSh, including the instance of Sh. The correctness property of the Sh protocol (see Theorem 4.1) implies that there exists triples, say $\vec{S}' = \{(x'^{(i)}, y'^{(i)}, z'^{(i)})\}_{i\in[\ell]}$ and $\vec{S}'_{sac} = \{(f'^{(i)}, g'^{(i)}, h'^{(i)})\}_{i\in[\ell]}$ that are t-shared among the parties. We next claim that if the ith triple in $\vec{S}'$ is not a multiplication triple for some $i \in [\ell]$, then the parties will see that $\gamma^{(i)} \neq 0$, except with probability at most $\frac{2}{|\mathbb{F}|}$ and they will output the default publicly known t-sharing of ℓ multiplication triples. So let the ith triple in $\vec{S}'$ be a non-multiplication triple. In this case, the condition $r(z'^{(i)} - x'^{(i)}y'^{(i)}) = (h'^{(i)} - f'^{(i)}g'^{(i)})$ will be true if either $r = 0$ or $r = (h'^{(i)} - f'^{(i)}g'^{(i)})(z'^{(i)} - x'^{(i)}y'^{(i)})^{-1}$. However, the challenge r is completely random (follows from the property of Rand) and is known to the corrupted D only after the instance of Sh is completed. This implies that the only way of ensuring that the condition $r(z'^{(i)} - x'^{(i)}y'^{(i)}) = (h'^{(i)} - f'^{(i)}g'^{(i)})$ holds is by correctly guessing the value of r, which can be done with probability at most $\frac{2}{\mathbb{F}}$. This completes the proof of correctness.

PRIVACY: We consider an *honest* D. In this case, the privacy property of the protocol Sh (see Theorem 4.1) implies that the multiplication triples in $\vec{S} = \{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i\in[\ell]}$ and $\vec{S}_{sac} = \{(f^{(i)}, g^{(i)}, h^{(i)})\}_{i\in[\ell]}$ are not known to Adv till the end of Sh. We now fix an $i \in [\ell]$ and show that the information obtained by Adv in the rest of the protocol does not leak anything about the triple $(x^{(i)}, y^{(i)}, z^{(i)})$, except the fact that it is a multiplication triple. In TripleSh, $\rho^{(i)}$, $\gamma^{(i)}$ and $\sigma^{(i)}$ are publicly reconstructed. However, $\rho^{(i)} = rx^{(i)} - f^{(i)}$ provides no information about $x^{(i)}$, as $f^{(i)}$ is random from the view point of Adv. Similarly, as $g^{(i)}$ is random from the view point of Adv, learning $\sigma^{(i)} = y^{(i)} - g^{(i)}$ provides no information about $y^{(i)}$. Finally, the fact that $\gamma^{(i)} = 0$ gives Adv no further information about the triple except that it is a multiplication triple.

COMMUNICATION COMPLEXITY. We note that the instance of Sh invoked by D requires a private communication of $\mathcal{O}(n\ell)$ and broadcast communication of $\mathcal{O}(n^2)$; this follows from the communication complexity of the Sh protocol to share ℓ values, where $\ell \geq t+1$ (see Theorem 4.1 and Sec. 4.1). The instances of BatRecPubl will require a private communication of $\mathcal{O}(n\ell)$ following the argument as follows: each instance of BatRecPubl requires a private communication of $\mathcal{O}(n^2)$ (see Theorem A.2) and TripleSh invokes $3\lceil\frac{\ell}{t+1}\rceil$ instances of BatRecPubl; moreover $t + 1 = \Theta(n)$. □

D.2 Proof of Theorem 6.2

Recall that for simplicity, although without loss of generality, while describing the protocol, we assumed Com to contain the first $3t+1$ parties. We follow the same assumption here to keep the proof simple.

TERMINATION: This property follows from the termination property of the protocol BatchBeaver (see Theorem 3.1).

CORRECTNESS: To argue correctness, we have to show that each triple in $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{l\in[\ell],\, i\in[\frac{t}{2}]}$ is a correct multiplication triple and is t-shared. For the proof, we fix an $l \in [\ell]$ and show that $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{i\in[\frac{t}{2}]}$ is indeed a multiplication triple. Recall that $\mathbf{a}^{(i,l)} = \mathsf{X}^l(\beta_i), \mathbf{b}^{(i,l)} = \mathsf{Y}^l(\beta_i)$ and $\mathbf{c}^{(i,l)} = \mathsf{Z}^l(\beta_i)$, for each $i \in [\frac{t}{2}]$, for three polynomials $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$. To complete the proof, it is enough to show that the protocol ensures the multiplicative relation $\mathsf{Z}^l(\cdot) = \mathsf{X}^l(\cdot)\mathsf{Y}^l(\cdot)$ holds. However, this follows from the correctness property of the triple transformation protocol TripTrans and the fact that $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$ are computed from the $3t+1$ triples $\{(x^{(i,l)}, y^{(i,l)}, z^{(i,l)})\}_{i\in[3t+1]}$, all of which are multiplication triples.

PRIVACY: We fix $l \in [\ell]$ and prove the privacy for the triples in $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{i\in[\frac{t}{2}]}$. Specifically, we show that the view of the adversary in the protocol TripExt is distributed independently of the multiplication triples in $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{i\in[\frac{t}{2}]}$. For this, we first claim that Adv learns at most t points on the polynomials $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$. However, this follows from the privacy property of the protocol TripTrans and the fact that at most t input triples $\{(x^{(i,l)}, y^{(i,l)}, z^{(i,l)})\}_{i\in[3t+1]}$ which are used to compute $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$ will be known to Adv. Given the above claim, the proof goes as follows: since the degree of $\mathsf{X}^l(\cdot)$ is at most $\frac{3t}{2}$, from the view point of the adversary, for every possible choice of $\{\mathbf{a}^{(i,l)}\}_{i\in[\frac{t}{2}]}$, there exists a unique polynomial $\mathsf{X}^l(\cdot)$ of degree at most $\frac{3t}{2}$, which will be consistent with the points $\{(\beta_i, \mathbf{a}^{(i,l)})\}_{i\in[\frac{t}{2}]}$ and the t points from $\{(\alpha_i, \mathbf{x}^{(i,l)})\}_{i\in[3t+1]}$ known to Adv. Thus $\{\mathsf{X}^l(\beta_i) = \mathbf{a}^{(i,l)}\}_{i\in[\frac{t}{2}]}$ will be random to Adv. The same argument allows us to claim that $\{\mathbf{b}^{(i,l)}\}_{i\in[\frac{t}{2}]}$ and $\{\mathbf{c}^{(i,l)}\}_{i\in[\frac{t}{2}]}$ will be random to Adv subject to $\mathsf{Z}^l(\beta_i) = \mathsf{X}^l(\beta_i)\mathsf{Y}^l(\beta_i)$.

COMMUNICATION COMPLEXITY: For communication complexity, we note that ℓ instances of the protocol TripTrans are executed, where each instance incurs a private communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$. □

D.3 Protocol PreProc and Proof of Theorem 6.3

Protocol PreProc is presented in Fig. 11.

Proof of Theorem 6.3:

TERMINATION: We first prove that the instance of ACS will terminate with an output consisting of $3t+1$ parties. The instances of TripleSh corresponding to at least $3t+1$ honest parties in $\mathcal{P}$ will be eventually terminated by all the honest parties. Now the termination of ACS follows from the termination property of the underlying Byzantine agreement protocol. Given that ACS terminates with an output consisting of $3t+1$ parties, the termination property of TripExt (see Theorem 6.2) ensures that all the honest parties will terminate the protocol PreProc.

CORRECTNESS: It is easy to see that if the honest parties terminate PreProc then the output triples are indeed t-shared. The correctness property of TripleSh ensures that the triples shared by the parties in Com are indeed t-shared and the correctness property of TripExt ensures that its output triples are also t-shared. What remains to be shown is that the output triples are *multiplication* triples. Here we note that except with probability at most $\frac{2t}{|\mathbb{F}|}$, all the triples shared by the parties in Com are indeed multiplication triples (follows from the correctness

Figure 11: An asynchronous protocol to generate t-sharing of $c_M + c_R$ random multiplication triples.

Protocol PreProc

TRIPLE SHARING — For $i \in [n]$, every party P_i executes the following code:

- Act as a dealer D and execute an instance of TripleSh to t-share ℓ random multiplication triples $\{(x^{(i,l)}, y^{(i,l)}, z^{(i,l)})\}_{l\in[\ell]}$, where $\ell = \frac{2(c_M+c_R)}{t}$. Let this instance of TripleSh be denoted as $\mathsf{TripleSh}_i$.
- For every $j \in [n]$ participate in the instance $\mathsf{TripleSh}_j$.

AGREEMENT ON A COMMON SET — For $i \in [n]$, every party P_i executes the following code:

- Create an accumulative set $\mathcal{C}_i = \emptyset$.
- Upon terminating the instance $\mathsf{TripleSh}_j$, include P_j in $\mathcal{C}_i$.
- Participate in an instance of ACS with the accumulative set $\mathcal{C}_i$ as your input.
- Wait to terminate the instance of ACS with an output Com consisting of $3t + 1$ parties.

TRIPLE EXTRACTION AND TERMINATION — The parties in $\mathcal{P}$ execute the following code:

- The parties execute $\mathsf{TripExt}(\mathsf{Com}, \{([x^{(i,l)}]_i, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{l\in[\ell],\ P_i\in\mathsf{Com}})$, output sharing of $c_M + c_R$ triples $\{([\mathbf{a}^{(i,l)}]_t, [\mathbf{b}^{(i,l)}]_t, [\mathbf{c}^{(i,l)}]_t)\}_{l\in[\ell],\ i\in[\frac{t}{2}]}$, where $\ell = \frac{2(c_M+c_R)}{t}$ and terminate.

property of TripleSh and the union bound). Given this pre-condition, it follows from the correctness of the protocol TripExt that the output triples are indeed multiplication triples.

PRIVACY: Given that there will be at least $2t + 1$ honest parties in the set Com and the multiplication triples shared by the honest parties in Com are random and unknown to Adv, the privacy property of TripExt ensures that the output triples in the protocol PreProc are random and unknown to Adv.

COMMUNICATION COMPLEXITY: By substituting $\ell = \frac{2(c_M+c_R)}{t}$ and $t = \Theta(n)$ in Theorem 6.1, we find that a single instance of TripleSh in PreProc will require a private communication of $\mathcal{O}(c_M+c_R)$ and broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$. So n instances of TripleSh will require private communication of $\mathcal{O}((c_M+c_R)n)$ and broadcast of $\mathcal{O}(n^3)$. Similarly substituting $\ell = \frac{2(c_M+c_R)}{t}$ in Theorem 6.2, we find that the instance of TripExt in PreProc will require a private communication of $\mathcal{O}((c_M + c_R)n)$. So overall, PreProc requires a private communication of $\mathcal{O}((c_M + c_R)n)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$. □

E The AMPC Protocol and Its Phases

E.1 The Input phase

The steps for the protocol Input are given in Fig. 12. Recall that we assume that each party has c_i inputs for the computation, such that $c_i \geq t + 1$ and $c_1 + \ldots + c_n = c_I$, where c_I is the number of input gates in the circuit. The properties of the protocol Input are stated in Theorem E.1.

Theorem E.1. *For every possible* Adv *and for every possible scheduler, protocol* Input *achieves:*

1. TERMINATION*: All the honest parties terminate the protocol with probability* 1.
2. CORRECTNESS*: Each honest party will output its shares corresponding to t-sharing of the inputs of the parties in* Com.
3. PRIVACY*: The information obtained by* Adv *in the protocol is distributed independently of the inputs of the honest parties in the set* Com.

Figure 12: Protocol for the input phase.

Protocol Input

1. INPUT SHARING — For $i \in [n]$, every party P_i executes the following code:
   - On having the input $\{\mathsf{x}^{(i,\ell)}\}_{\ell \in [c_i]}$ where c_i is the number of inputs from P_i for the computation, act as a dealer and execute an instance of Sh to t-share $\{\mathsf{x}^{(i,\ell)}\}_{\ell \in [c_i]}$. Let this instance of Sh be denoted by Sh_i.
   - For every $j \in [n]$, participate in the instance Sh_j.
2. AGREEMENT ON A COMMON SET AND TERMINATION — For $i \in [n]$, every party P_i executes the following code:
   - Create an accumulative set $\mathcal{C}_i = \emptyset$.
   - On terminating the instance Sh_j, include P_j in $\mathcal{C}_i$.
   - Participate in an instance of ACS with the accumulative set $\mathcal{C}_i$ as your input.
   - Wait to terminate the instance of ACS with an output Com consisting of $3t + 1$ parties.
   - For every $P_j \in \mathsf{Com}$ output the shares obtained in the instance Sh_j and for every $P_j \notin \mathsf{Com}$, output c_j 0s as the shares and terminate.

4. COMMUNICATION COMPLEXITY*: The protocol incurs a private communication of $\mathcal{O}(c_I n)$ and broadcast communication of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$ and requires one invocation to* ACS*.*

PROOF: TERMINATION: We first claim that the instance of ACS will eventually terminate for each honest party (with probability 1) with an output consisting of $3t + 1$ parties. This follows from the fact that there are $3t + 1$ honest parties in $\mathcal{P}$ and the Sh instance dealt by each honest party will be terminated eventually. Now it is easy to see that once the instance of ACS terminates, the parties will terminate the protocol Input.

CORRECTNESS: It follows from the correctness of Sh.

PRIVACY: The privacy property of the protocol Sh implies that the information obtained by Adv in the instances of Sh executed by the honest parties (dealers) in Com is distributed independently of the values shared in those instances. This implies the privacy property for the protocol Input.

COMMUNICATION COMPLEXITY: Assuming that every party P_i has c_i inputs for the computation where $c_i \geq t + 1$, each instance of Sh requires a private communication of $\mathcal{O}(c_i n)$ and broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$; this follows from the communication complexity of the protocol Sh. So in total, protocol Input will incur a private communication of $\mathcal{O}(c_I n)$ and broadcast of $\mathcal{O}(n^3)$, since $c_1 + \ldots + c_n = c_I$. □

E.2 The Computation phase

The computation phase is implemented by the protocol CircEval. The protocol works given t-sharing of $c_M + c_R$ random multiplication triples, say $\{([\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)\}_{i \in [c_M + c_R]}$ and t-sharing of the inputs of the parties. We assume that it is a common knowledge that the shared triple $([\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)$ is associated with the ith multiplication gate in the circuit and the shared triple $([\mathbf{a}^{(c_M+i)}]_t, [\mathbf{b}^{(c_M+i)}]_t, [\mathbf{c}^{(c_M+i)}]_t)$ is associated with the ith random gate in the circuit. Moreover, we evaluate[7] $t + 1$ multiplication gates at once, assuming that they are independent of each other. Protocol CircEval is presented in Fig. 13.
The properties of the protocol CircEval are stated in Theorem E.2.

Theorem E.2. *Given t-sharings $\{([\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)\}_{i \in [c_M + c_R]}$ of $c_M + c_R$ random multiplication triples and t-sharings of the inputs of the parties (for the computation), protocol* CircEval *achieves the following for every possible* Adv *and every possible scheduler:*

[7]Recall that evaluating a gate means to compute t-sharing of the output of the gate from t-sharing of the inputs of the gate.

Figure 13: Protocol for evaluating the circuit using preprocessed multiplication triples.

Protocol $\mathsf{CircEval}(\{([\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)\}_{i\in[c_M+c_R]})$

For all the gates in the circuit the (honest) parties in $\mathcal{P}$ do the following (depending upon the type of gate) and then terminate:

- ADDITION/LINEAR GATE: The parties locally apply the linear function on their respective shares of the inputs of the gate.
- RANDOM GATE: If this is the ith random gate in the circuit then the parties locally output their shares corresponding to the sharing $[\mathbf{a}^{(c_M+i)}]_t$.
- MULTIPLICATION GATES: Up to $t+1$ multiplication gates are processed simultaneously. Let $([\mathsf{x}^{(1)}]_t, [\mathsf{y}^{(1)}]_t), \ldots, ([\mathsf{x}^{(t+1)}]_t, [\mathsf{y}^{(t+1)}]_t)$ be the input sharings of the gates and $([\mathbf{a}^{(1)}]_t, [\mathbf{b}^{(1)}]_t, [\mathbf{c}^{(1)}]_t), \ldots, ([\mathbf{a}^{(t+1)}]_t, [\mathbf{b}^{(t+1)}]_t, [\mathbf{c}^{(t+1)}]_t)$ be the associated multiplication triples. The parties execute $\mathsf{BatchBeaver}(\{([\mathsf{x}^{(i)}]_t, [\mathsf{y}^{(i)}]_t, [\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)\}_{i\in[t+1]})$ to compute $\{[\mathsf{x}^{(i)}\mathsf{y}^{(i)}]_t\}_{i\in[t+1]}$.
- OUTPUT GATE (OUTPUT $[s]_t$ TO PARTY P_R): The parties execute $\mathsf{OEC}(P_R, t, [s]_t)$.

1. TERMINATION*: All the honest parties eventually terminate the protocol.*
2. CORRECTNESS*: All the gates are evaluated correctly.*
3. PRIVACY*: For every gate in the circuit, the evaluation of the gate reveals no additional information to* Adv.
4. COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}((c_M + c_O)n)$ elements from $\mathbb{F}$.*

PROOF: The correctness and termination property follows from the correctness and termination property of BatchBeaver and OEC. For privacy, we see that the evaluation of random and linear gates require no communication among the parties. Moreover, for the multiplication gates, we use the protocol BatchBeaver by employing the random multiplication triples from the preprocessing phase and so the privacy follows from the privacy property of BatchBeaver. For communication complexity, we observe that each instance of BatchBeaver in CircEval requires a private communication of $\mathcal{O}(n^2)$ (follows by substituting $\ell = t+1$ in Theorem 3.1) and so evaluating c_M multiplication gates will require a private communication of $\mathcal{O}(\frac{c_M}{t+1}n^2) = \mathcal{O}(c_M n)$, as $t+1 = \Theta(n)$. The reconstruction of each output value requires a private communication of $\mathcal{O}(n)$ (follows from the property of OEC). □

E.3 The AMPC Protocol

Let f be a publicly known function over $\mathbb{F}$, expressed as an arithmetic circuit C over $\mathbb{F}$, consisting of c_I, c_O, c_M and c_R number of input, output, multiplication and random gates respectively. Then the statistical AMPC protocol AsynAMPC to securely compute the function f is given in Fig. 14.

Figure 14: The statistical AMPC protocol.

Protocol $\mathsf{AsynAMPC}(C, c_I, c_M, c_R, c_O)$

The parties in $\mathcal{P}$ do the following and terminate:

1. Invoke PreProc to generate t-sharing of $c_M + c_R$ random multiplication triples.
2. Invoke Input to generate the t-sharing of the inputs of the parties for the computation.
3. Invoke CircEval to evaluate the circuit C.

We now prove Theorem 7.1.

(Informal) Proof of Theorem 7.1: We first note that properties of the protocol PreProc imply that all the honest parties will terminate the protocol PreProc and will output t-sharing of $c_M + c_R$ random multiplication triples,

except with probability $\frac{2t}{|\mathbb{F}|}$. Given this precondition, it is easy to see that each honest party will eventually terminate Input and CircEval with correct outputs. Now if $|\mathbb{F}| \geq 2t \cdot 2^{\kappa}$, then $\frac{2t}{|\mathbb{F}|} \leq 2^{-\kappa}$ and thus it follows that except with error probability at most $2^{-\kappa}$, protocol TripleSh will satisfy the correctness property (of AMPC). The privacy property follows from the privacy property of PreProc, Input and CircEval, while the communication complexity follows from the communication complexity of PreProc, Input and CircEval. □

F Error-free AMPC Protocol in the Hybrid Model

This section is organized as follows: we first discuss the preprocessing phase in the hybrid model, followed by the input phase protocol and finally present the MPC protocol.

F.1 Error-free Preprocessing Phase

The error-free preprocessing phase protocol consists of two subprotocols: an error-free triple sharing protocol HybTripleSh which allows a dealer $\mathsf{D} \in \mathcal{P}$ to verifiably t-share multiplication triples, assuming the first communication round to be a synchronous round. The second subprotocol is the triple extraction protocol TripExt (used in the asynchronous preprocessing phase protocol). Looking ahead, in our hybrid model protocol HybTripleSh, each party will be asked to t-share a number of random multiplication triples in the synchronous round by using Shamir secret sharing [33]. However no sanity check on the generated sharings is performed (also is not required for the working of HybTripleSh). Such triples are said to be generated "non-verifiably" and might allow the corrupted parties to do arbitrary sharings in place of t-sharings. By noting that any arbitrary distribution of values to the honest parties in the name of sharing can be perceived as a d-sharing for an appropriate $d \leq 3t$ (since there are $3t+1$ honest parties and the shares of these honest parties can define a polynomial of degree at most $3t$), we refer a sharing with an arbitrary degree below $3t+1$ as an arbitrary sharing and we denote such an arbitrary sharing without specifying the subscript that denotes the degree of sharing; for example $[v]$ instead of $[v]_d$. Note that an arbitrary sharing with degree at most t becomes a valid sharing. Looking ahead, if the corrupted parties perform arbitrary sharing (instead of t-sharing) then this might lead to inputting arbitrary sharings to the protocols OEC, BatRecPubl and BatchBeaver (which will be used in the hybrid protocol HybTripleSh), which may cause problem for the termination. For such sharings, our goal is therefore to first modify the protocols OEC, BatRecPubl and BatchBeaver so that they always terminate, even if the input sharings are not t-sharings. Furthermore, the modified protocols guarantee the correctness property of OEC, BatRecPubl and BatchBeaver, respectively, when the inputs sharings are valid t-sharing. The correctness property is therefore not desirable when the inputs are not t-sharing. We next discuss how to obtain the modified protocols and note that a linear function applied on a set of arbitrary sharings leads to arbitrary sharings.

- **mOEC$(P_R, [v])$:** mOEC is obtained from OEC by modifying the steps for P_R, the party that would like to reconstruct the input sharing, as follows: party P_R waits for the shares from *any* $3t+1$ parties and includes them in $\mathcal{I}$. Then it applies RSDec$(t, t, \mathcal{I})$. If a polynomial of degree at most t is obtained from RSDec, then it outputs the same polynomial (and its constant term). Otherwise it outputs a default polynomial of degree at most t (and its constant term). Now it is easy to note that (an honest) P_R always terminates the protocol (irrespective of the degree of sharing of v), since the $3t+1$ honest parties in $\mathcal{P}$ will send their shares to P_R. Furthermore, if the input sharing is a t-sharing, then clearly the output of P_R in mOEC is the same as it would be in the protocol OEC (with input $[v]_t$); this follows from the property of RSDec. Note that mOEC has the same communication complexity as that of OEC.

- **mBatRecPubl$(\mathcal{P}, [u^{(1)}], \ldots, [u^{(t+1)}])$:** mBatRecPubl is obtained by modifying BatRecPubl where each instance of OEC is replaced by mOEC. The termination now follows from the termination of mOEC. Furthermore, when the input sharings are t-sharings, then the output (of the honest parties) in mBatRecPubl will be the same as it would be in the protocol BatRecPubl (with inputs $[u^{(1)}]_t, \ldots, [u^{(t+1)}]_t$), i.e. *all* the honest parties will *robustly* reconstruct $u^{(1)}, \ldots, u^{(t+1)}$. However, if all the input sharings are not t-sharings, then

different (honest) parties may end up reconstructing different $t+1$ values, depending upon the output of the instances of mOEC. The communication complexity of mBatRecPubl is the same as BatRecPubl.

- mBatchBeaver($\{([x^{(i)}]_t, [y^{(i)}]_t, [a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[\ell]}$)**:** it is obtained by modifying BatchBeaver, where each instance of BatRecPubl is replaced by mBatRecPubl. We note that in mBatchBeaver, *only* the sharings $\{([a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[\ell]}$ of the triples may be arbitrary (and possibly the triples may be non-multiplication triples); the sharings of the pairs $\{([x^{(i)}]_t, [y^{(i)}]_t)\}_{i\in[\ell]}$ (whose product need to be computed) will be t-sharings. The termination now follows from the termination of mBatRecPubl. Furthermore, when the sharings $\{[a^{(i)}], [b^{(i)}], [c^{(i)}])\}_{i\in[\ell]}$ are t-sharing of multiplication triples, then the output (of the honest parties) in mBatchBeaver will be the same as it would be in the protocol BatchBeaver; i.e. the parties will output $\{[x^{(i)}y^{(i)}]_t\}_{i\in[\ell]}$. Otherwise, the parties will output $\{[z^{(i)}]\}_{i\in[\ell]}$, where each $z^{(i)}$ can be any arbitrary value. The communication complexity of mBatchBeaver is the same as BatchBeaver.

We are now ready to present the error-free protocol HybTripleSh for triple sharing in the hybrid model.

F.1.1 The Error-free Protocol for Verifiably Sharing Multiplication Triples

We present a protocol called HybTripleSh, which allows a dealer $\mathsf{D} \in \mathcal{P}$ to t-share $\frac{t\ell}{2}$ random multiplication triples in a verifiable fashion with a private communication of $\mathcal{O}(n^2\ell)$, a broadcast communication of $\mathcal{O}(n^2)$ and n invocations of any existing almost surely terminating ABA protocol [1, 30], where $\ell \geq t+1$. The efficient almost surely terminating ABA protocol of [30] designed with $4t+1$ parties with a communication complexity of poly(n) is good enough for our purpose. Since the high level idea of HybTripleSh has been already discussed in Section 8, we present the protocol in Fig. 15 rightaway where the idea discussed in Section 8 is applied ℓ times in parallel.

Theorem F.1. *In a hybrid network model where the first communication round is a synchronous round, protocol* HybTripleSh *achieves the following for every possible* Adv *and every possible scheduler:*

(1) TERMINATION*: If* D *is honest then all the honest parties eventually terminate the protocol with probability* 1*. If* D *is corrupted and some honest party terminates, then all the honest parties eventually terminate the protocol with probability* 1*.* **(2)** CORRECTNESS*: If* D *is honest then $\frac{t\ell}{2}$ multiplication triples will be t-shared, where $\ell \geq t+1$. If* D *is corrupted and some honest party terminates then $\frac{t\ell}{2}$ multiplication triples will be t-shared, where $\ell \geq t+1$.* **(3)** PRIVACY*: If* D *is honest, then the view of* Adv *in the protocol is distributed independently of the $\frac{t\ell}{2}$ output multiplication triples, where $\ell \geq t+1$.* **(4)** COMMUNICATION COMPLEXITY*: The protocol requires a private communication of $\mathcal{O}(n^2\ell)$ and broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$, plus n invocations to ABA.*

PROOF: TERMINATION: The termination property follows from the termination property of Sh, mBatchBeaver, mBatRecPubl, the termination property of the almost surely terminating ABA and the fact that the first synchronous round will always terminate.

CORRECTNESS: We first consider an *honest* D, where the t-shared triples $\{([x^{(i,l)}]_t, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{i\in[3t+1],\, l\in[\ell]}$ will be multiplication triples. By the property of TripTrans, the relation $\mathsf{Z}^l(\cdot) = \mathsf{X}^l(\cdot)\mathsf{Y}^l(\cdot)$ will hold for all $l \in [\ell]$ and the triples $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{i\in[n],\, l\in[\ell]}$ will be multiplication triples. Now consider an *honest* P_i in $\mathcal{P}$, where the dummy triples shared by P_i will be multiplication triples and will be t-shared. It thus follows from the properties of mBatchBeaver that $\{\mathsf{z}^{(i,l)}\}_{l\in[\ell]} = \{\mathbf{x}^{(i,l)}\mathbf{y}^{(i,l)}\}_{l\in[\ell]}$ will hold and each such $\mathsf{z}^{(i,l)}$ will be t-shared. This further implies that $\{[\gamma^{(i,l)}]\}_{l\in[\ell]} = \{[0]_t\}_{l\in[\ell]}$ will hold, as $\mathsf{z}^{(i,l)} = \mathbf{z}^{(i,l)}$ will hold. It thus follows from the properties of mBatRecPubl that each honest P_j will reconstruct $\gamma^{(i,l,j)} = 0$ for all $l \in [\ell]$ and will input 0 to the instance $\mathsf{ABA}^{(i)}$ which further implies that the honest parties will set $\mathsf{flag}_i = 1$ at the end of $\mathsf{ABA}^{(i)}$. Next consider a *corrupted* P_i, where the dummy triples (probably non-multiplication triples) may be arbitrarily shared by P_i. Corresponding to such an i, if the output of $\mathsf{ABA}^{(i)}$ is 1, then the parties will robustly reconstruct the triples $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{l\in[\ell]}$, which in this case are multiplication triples and so the parties will set $\mathsf{flag}_i = 1$. So for an honest D, $\mathsf{flag}_i = 1$ will hold for all $i \in [n]$. It is now easy to see that the multiplication triples $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{i\in[\frac{t}{2}],\, l\in[\ell]}$ will be t-shared among the parties.

Figure 15: An error-free protocol for t-sharing $\frac{t\ell}{2}$ random multiplication triples where $\ell \geq t+1$.

Protocol $\mathsf{HybTripleSh}(\mathsf{D})$

First Synchronous Round

For $i \in [n]$, every party P_i executes the following code:

1. Select ℓ random dummy multiplication triples $\{(f^{(i,l)}, g^{(i,l)}, h^{(i,l)})\}_{l\in[\ell]}$ and t-share them using Shamir secret sharing.
2. For every $j \in [n]$, receive from P_j the shares of the dummy triples selected by P_j, till the first round is over. If some share is not received from P_j by the end of first round, then set them to a publicly known default value.

Let $\{([f^{(i,l)}], [g^{(i,l)}], [h^{(i,l)}])\}_{l\in[\ell],\ P_i\in\mathcal{P}}$ denote the sharings of the dummy triples done by the party P_i.

The Remaining Asynchronous Protocol

1. D selects $(3t+1)\ell$ random multiplication triples $\{(x^{(i,l)}, y^{(i,l)}, z^{(i,l)})\}_{i\in[3t+1],\ l\in[\ell]}$ and t-shares them by executing an instance of Sh and the parties participate in the instance of Sh.
2. After terminating the instance of Sh, the parties do the following for each $l \in [\ell]$:
   (a) Execute $\mathsf{TripTrans}(\{([x^{(i,l)}]_t, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{i\in[3t+1]})$ to compute $(\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{i\in[3t+1]})$. Let $\mathsf{X}^l(\cdot)$, $\mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$ denote the associated polynomials.
   (b) Compute (locally) $(\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{i\in[3t+2,n]})$ such that $\mathbf{x}^{(i,l)} = \mathsf{X}^l(\alpha_i)$, $\mathbf{y}^{(i,l)} = \mathsf{Y}^l(\alpha_i)$ and $\mathbf{z}^{(i,l)} = \mathsf{Z}^l(\alpha_i)$ for $i \in [3t+2, n]$.
3. For $i \in [n]$, the parties verify the ℓ shared triples $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{l\in[\ell]}$ using the ℓ shared dummy triples $\{([f^{(i,l)}], [g^{(i,l)}], [h^{(i,l)}])\}_{l\in[\ell]}$ shared by P_i as follows:
   (a) The parties execute $\mathsf{mBatchBeaver}(\{[\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [f^{(i,l)}], [g^{(i,l)}], [h^{(i,l)}]\}_{l\in[\ell]})$ and compute the sharings $\{[\mathsf{z}^{(i,l)}]\}_{l\in[\ell]}$.
   (b) The parties compute (locally) $\{[\gamma^{(i,l)}]\}_{l\in[\ell]}$, where $[\gamma^{(i,l)}] = [\mathsf{z}^{(i,l)}] - [\mathbf{z}^{(i,l)}]_t$, for $l \in [\ell]$.
   (c) The parties execute $\lceil \frac{\ell}{t+1} \rceil$ instances of $\mathsf{mBatRecPubl}$ on $\{[\gamma^{(i,l)}]\}_{l\in[\ell]}$ to reconstruct these values. Let the output of $\mathsf{mBatRecPubl}$ for P_j be $\{\gamma^{(i,l,j)}\}_{l\in[\ell]}$[a].
   (d) The parties participate in the instance $\mathsf{ABA}^{(i)}$ of ABA where party P_j inputs 0 if $\gamma^{(i,l,j)} = 0$ for all $l \in [\ell]$, otherwise P_j inputs 1.
   (e) If the output of $\mathsf{ABA}^{(i)}$ is 0 then the parties (locally) set a Boolean flag $\mathsf{flag}_i = 1$ to indicate that $\{(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})\}_{l\in[\ell]}$ are correct multiplication triples. Else if the output of $\mathsf{ABA}^{(i)}$ is 1 then the parties execute $\lceil \frac{3\ell}{t+1} \rceil$ instances of $\mathsf{BatRecPubl}$ on $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{l\in[\ell]}$ to reconstruct $\{(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})\}_{l\in[\ell]}$. After reconstruction, the parties check if all the triples $\{(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})\}_{l\in[\ell]}$ are multiplication triples and accordingly (locally) set $\mathsf{flag}_i = 1$ or $\mathsf{flag}_i = 0$
4. If $\mathsf{flag}_i = 1$ for all $i \in [n]$, then the parties output $\frac{t\ell}{2}$ sharings $\{([\mathbf{a}^{(i,l)}]_t, [\mathbf{b}^{(i,l)}]_t, [\mathbf{c}^{(i,l)}]_t)\}_{i\in[\frac{t}{2}],\ l\in[\ell]}$ and terminate, where $\mathbf{a}^{(i,l)} = \mathsf{X}^l(\beta_i)$, $\mathbf{b}^{(i,l)} = \mathsf{Y}^l(\beta_i)$ and $\mathbf{c}^{(i,l)} = \mathsf{Z}^l(\beta_i)$, for $i \in [\frac{t}{2}]$ and $l \in [\ell]$. Else, the parties output $\frac{t\ell}{2}$ default sharings of multiplication triples and terminate.

[a] Each instance of $\mathsf{mBatRecPubl}$ can be used to reconstruct $t+1$ shared values and so to reconstruct ℓ shared values, $\lceil \frac{\ell}{t+1} \rceil$ instances are required. If the $\gamma^{(i,l)}$s are not t-shared then different parties may reconstruct different values at the end of $\mathsf{mBatRecPubl}$ and so the extra index j here is used to capture the values which are reconstructed by P_j at the end of $\mathsf{mBatRecPubl}$.

On the other hand, if D is *corrupted*, then there are two possible cases: the parties either output default t-sharing of $\frac{t\ell}{2}$ publicly known multiplication triples or the parties output triples as computed in the protocol. We focus on the second case, which also implies that $\mathsf{flag}_i = 1$ for all $i \in [n]$. Here, it is enough to show that corresponding to every *honest* P_i, the triples $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{l\in[\ell]}$ are indeed multiplication triples, which will further confirm that $\mathsf{Z}^l(\cdot) = \mathsf{X}^l(\cdot)\mathsf{Y}^l(\cdot)$ holds, implying that all the triples shared by D are multiplication triples. An honest P_i will correctly t-share its random dummy multiplication triples. This implies that the parties will correctly obtain $\{[\mathsf{z}^{(i,l)} = \mathbf{x}^{(i,l)}\mathbf{y}^{(i,l)}]_t\}_{l\in[\ell]}$ at the end of $\mathsf{mBatchBeaver}$. Now if all the ℓ triples in $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{l\in[\ell]}$ are not multiplication triples, then clearly all the ℓ values in $\{\gamma^{(i,l)} = \mathsf{z}^{(i,l)} - \mathbf{z}^{(i,l)}\}_{l\in[\ell]}$ will not be 0. However, if

this is the case, then every honest party would have input 1 to $\mathsf{ABA}^{(i)}$ and obtain 1 as its output. This is because each $\gamma^{(i,l)}$ would be t-shared (as the dummy triples were t-shared) and so all the honest parties would correctly reconstruct $\{\gamma^{(i,l)}\}_{l\in[\ell]}$ at the end of $\mathsf{mBatRecPubl}$. Moreover, after obtaining 1 as the output of $\mathsf{ABA}^{(i)}$, the parties would have publicly reconstructed the triples $\{(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})\}_{l\in[\ell]}$ and would have set $\mathsf{flag}_i = 0$ after finding that all the triples in $\{(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})\}_{l\in[\ell]}$ are not multiplication triples. However, this is a contradiction.

PRIVACY: For privacy, we consider an *honest* D. We first fix an $l \in [\ell]$ and then claim that Adv will learn at most t points on the polynomials $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$, which are of degree at most $\frac{3t}{2}, \frac{3t}{2}$ and $3t$ respectively. Then following the same arguments as used for the triple extraction protocol $\mathsf{TripExt}$, it will follow that the multiplication triples $\{(\mathbf{a}^{(i,l)}, \mathbf{b}^{(i,l)}, \mathbf{c}^{(i,l)})\}_{i\in[\frac{t}{2}]}$ will be random from the view point of Adv. Since D is honest, clearly all the shared triples in $\{([x^{(i,l)}]_t, [y^{(i,l)}]_t, [z^{(i,l)}]_t)\}_{i\in[3t+1]}$ will be random and unknown to Adv and so from the properties of $\mathsf{TripTrans}$ (by substituting $t' = 0$), it follows that all the shared triples in $\{([\mathbf{x}^{(i,l)}]_t, [\mathbf{y}^{(i,l)}]_t, [\mathbf{z}^{(i,l)}]_t)\}_{i\in[n]}$ will be random and so Adv will not learn any point on $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$ after $\mathsf{TripTrans}$. Now there can be at most t *corrupted* P_is and corresponding to them, Adv will learn the multiplication triple $(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})$ during its verification, as Adv will know the corresponding dummy triple $(f^{(i,l)}, g^{(i,l)}, h^{(i,l)})$, used for its verification. However, corresponding to the *honest* P_is, the random dummy multiplication triples $(f^{(i,l)}, g^{(i,l)}, h^{(i,l)})$ will be t-shared and will be not known to Adv. This further implies that while computing $[\mathsf{z}^{(i,l)} = \mathbf{x}^{(i,l)}\mathbf{y}^{(i,l)}]_t$ using $([f^{(i,l)}]_t, [g^{(i,l)}]_t, [h^{(i,l)}]_t)$, no additional information about the multiplication triple $(\mathbf{x}^{(i,l)}, \mathbf{y}^{(i,l)}, \mathbf{z}^{(i,l)})$ will be leaked to Adv. Finally, the sharings $[\mathsf{z}^{(i,l)}]_t$ and $[\mathbf{z}^{(i,l)}]_t$ will be independent, except that $\mathsf{z}^{(i,l)} = \mathbf{z}^{(i,l)}$ and so Adv will already know that $\gamma^{(i,l)} = 0$ and thus no new information is added to its view after the public reconstruction of $[\gamma^{(i,l)}]_t$. So overall, Adv will learn t values on $\mathsf{X}^l(\cdot), \mathsf{Y}^l(\cdot)$ and $\mathsf{Z}^l(\cdot)$.

COMMUNICATION COMPLEXITY: During the synchronous round, each party has to Shamir share ℓ multiplication triples, which will incur a total private communication of $\mathcal{O}(n^2\ell)$ elements from $\mathbb{F}$. Sharing of $(3t+1)\ell$ triples by D during the instance of Sh requires a private communication of $\mathcal{O}(n^2\ell)$ elements and broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$. Apart from this, there will be ℓ instances of $\mathsf{TripTrans}$ incurring $\mathcal{O}(n^2\ell)$ private communication, n instances of $\mathsf{mBatchBeaver}$ incurring $\mathcal{O}(n^2\ell)$ private communication, $\lceil \frac{n\ell}{t+1} \rceil$ instances of $\mathsf{mBatRecPubl}$ incurring $\mathcal{O}(n^2\ell)$ private communication and at most $\lceil \frac{3n\ell}{t+1} \rceil$ instances of $\mathsf{BatRecPubl}$ incurring $\mathcal{O}(n^2\ell)$ private communication. Moreover, n invocations of ABA are involved, one on behalf of each party. So overall, the protocol requires a private communication of $\mathcal{O}(n^2\ell)$ elements and broadcast communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$ and n invocations to the ABA protocol. □

F.2 The Error-free Preprocessing Phase Protocol HybPrePro in the Hybrid Model

The goal of the protocol $\mathsf{HybPrePro}$ is to generate t-sharing of $c_M + c_R + c_I(t+1)$ random multiplication triples in an error-free fashion, assuming that the first communication round is a synchronous round. The protocol goes along the same line as the asynchronous preprocessing phase protocol $\mathsf{PreProc}$, with the following differences:

- Instead of t-sharing $\frac{2(c_M+c_R)}{t}$ random multiplication triples, each party now t-shares $\frac{2(c_M+c_R+c_I(t+1))}{t}$ random multiplication triples. This is because now we require *additional* $c_I(t+1)$ random sharings from the preprocessing phase (to be used for the input provision during the input phase).
- Instead of executing an instance of the probabilistic triple sharing protocol $\mathsf{TripleSh}$, each party shares the triples using an instance of the error-free triple sharing protocol $\mathsf{HybTripleSh}$.

After each party shares the triples, as in the protocol $\mathsf{PreProc}$, we execute an instance of ACS to agree on a set Com of $3t+1$ "good" dealers, who have correctly shared the multiplication triples. We then execute an instance of the triple extraction protocol $\mathsf{TripExt}$ to extract t-sharing of $c_M + c_R + c_I(t+1)$ random multiplication triples. As the protocol is a straight forward modification of the protocol $\mathsf{TripleSh}$ we do not provide the formal details. The properties of the protocol are stated in Theorem F.2.

Theorem F.2. *Assuming a hybrid network where the first communication round is a synchronous round, for every possible* Adv *and every possible scheduler, protocol* HybPrePro *achieves:*

(1) TERMINATION*: All the honest parties eventually terminate with probability* 1. **(2)** CORRECTNESS*:* $c_M + c_R + c_I(t+1)$ *multiplication triples will be* t*-shared among the parties.* **(3)** PRIVACY*: The view of* Adv *during the protocol will be independent of the output multiplication triples.* **(4)** COMMUNICATION COMPLEXITY*: The protocol incurs private communication of* $\mathcal{O}((c_M + c_R)n + c_I n^2)$ *and broadcast of* $\mathcal{O}(n^3)$ *elements from* $\mathbb{F}$*, plus one invocation to* ACS *and* n^2 *invocations to ABA.*

PROOF: The termination, correctness and privacy follow from the properties of the protocol HybTripleSh and TripExt and similar arguments as used in Theorem 6.3. For communication complexity, we see that each instance of HybTripleSh will incur a private communication of $\mathcal{O}(c_M + c_R + c_I(t+1))$ and broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$, plus n invocations to ABA. This is because each instance of HybTripleSh outputs t-sharing of $\frac{2(c_M+c_R+c_I(t+1))}{t}$ multiplication triples, which implies that ℓ (for the protocol HybTripleSh) is $\frac{4(c_M+c_R+c_I(t+1))}{t^2}$ (recall that HybTripleSh outputs $\frac{t\ell}{2}$ multiplication triples). So n instances of HybTripleSh incurs private communication of $\mathcal{O}((c_M + c_R)n + c_I n^2)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$ and n^2 invocations to ABA. □

F.3 The Input Phase Protocol in the Hybrid Model

We now briefly recall from [4] how using the first synchronous round, we can enforce input provision and consider the inputs of all the n parties for the computation. For the ease of explanation, we assume that each party P_i has a *single* input $\mathsf{x}^{(i)} \in \mathbb{F}$ for the computation. The idea is that during the first synchronous round, each party P_i computes a degree-t Shamir sharing [33] of his input $\mathsf{x}^{(i)}$ and sends one share to each party. If a party does not receive the share corresponding to the input of some party by the end of this synchronous round, then it substitutes a default publicly known share. In [4], this (trivial) one round protocol is called PrepareInputs. An easy observation is that if P_i is *honest* then $\mathsf{x}^{(i)}$ will be t-shared at the end of PrepareInputs. However, for a *corrupted* P_i, its input $\mathsf{x}^{(i)}$ may not be t-shared. Note that the protocol PrepareInputs need to be executed parallely with the preprocessing phase protocol HybPrePro, as the protocol HybPrePro also need to use the first synchronous round. Now later, the parties execute the fully asynchronous protocol Input (the protocol for the input phase in the completely asynchronous protocol AsynAMPC), where the input for a party P_i will be now a vector $\tilde{\mathsf{x}}^{(i)}$ consisting of his "real input" $\mathsf{x}^{(i)}$, plus his shares $\mathsf{x}_i^{(j)}$ of the inputs $\mathsf{x}^{(j)}$ of the other parties. As a result of the asynchronous protocol Input, the parties will agree on a common set Com of $3t+1$ parties, whose vectors are (eventually) t-shared among the parties. For every party $P_i \in \mathsf{Com}$, the t-sharing $[\mathsf{x}^{(i)}]_t$ of his input is obtained directly from the t-sharing of the first component of his t-shared vector $\tilde{\mathsf{x}}^{(i)}$. On the other hand, for every party $P_k \notin \mathsf{Com}$, the t-sharing $[\mathsf{x}^{(k)}]_t$ of his input $\mathsf{x}^{(k)}$ is "restored" using a protocol called RestoreInputs [4]. We next present the high level description of RestoreInputs; for the complete details, see [4].

The idea behind the protocol RestoreInputs is the following: consider a party $P_k \notin \mathsf{Com}$. There will be $3t+1$ shares of his input $\mathsf{x}^{(k)}$, namely $\{\mathsf{x}_i^{(k)}\}_{P_i \in \mathsf{Com}}$, each of which will be t-shared as a component of the input vector $\tilde{\mathsf{x}}^{(i)}$ corresponding to the parties in Com. Now at most t parties in Com might have t-shared incorrect shares of the input $\mathsf{x}^{(k)}$; these t-shared incorrect shares are error-corrected in a shared fashion and finally the t-sharing of $\mathsf{x}^{(k)}$ is restored. To perform the error correction in a shared fashion, $t+1$ random t-sharings from the preprocessing phase are used and this is why, $c_I(t+1)$ additional t-shared multiplication triples are generated in the preprocessing phase protocol HybPrePro. Note that the protocol RestoreInputs is a completely asynchronous protocol and will be executed *only after* the preprocessing phase is over. We also note that the error correction may fail in case if P_k is *corrupted* and did not t-shared his input $\mathsf{x}^{(k)}$ during the synchronous protocol PrepareInputs; however, if this is the case then all the honest parties will come to known this at the end of RestoreInputs and a default t-sharing is substituted as the input sharing for such a corrupted P_k.

In a more detail, the shared error correction is done as follows: let $P_k \in \mathsf{Com}$ and the goal is to restore $[\mathsf{x}^{(k)}]_t$ by using $t+1$ random t-sharings from the preprocessing phase, say $[r^{(0)}]_t, \ldots, [r^{(t)}]_t$, provided the party P_k has correctly t-shared his input $\mathsf{x}^{(k)}$ among the parties in Com during the synchronous protocol PrepareInputs. For this, we first define a blinding polynomial $b(x) \stackrel{def}{=} r^{(0)} + r^{(1)}x + \ldots + r^{(t)}x^t$ and let every party $P_i \in \mathcal{P}$ to privately

reconstruct b_i, where $b_i = b(\alpha_i)$. For this, the parties first locally compute $[b_i]_t = [r^{(0)}]_t + \alpha_i[r^{(1)}]_t + \ldots + \alpha_i^t[r^{(t)}]_t$, followed by the private reconstruction of $[b_i]_t$ towards P_i. Thus, each party will have a distinct point on the blinding polynomial, which has degree at most t. The next step is that for every $P_i \in \mathsf{Com}$, the parties locally compute $[d_i]_t = [\mathsf{x}_i^{(k)}]_t + [b_i]_t$, followed by the public reconstruction of these $[d_i]_t$s. Finally, the parties apply the RS error correction on the $3t+1$ publicly reconstructed d_is to correct t errors and check whether there exists a polynomial, say $p(\cdot)$, having degree at most t, such that at least $2t+1$ reconstructed d_is lie on $p(\cdot)$. If this is the case, then every party P_i outputs $p(\alpha_i) - b_i$ as his share for $\mathsf{x}^{(k)}$; otherwise the parties output a default t-sharing. The intuition here is that the parties try to reconstruct the polynomial $p(\cdot) = b(\cdot) + q(\cdot)$, where $q(\cdot)$ is the polynomial through which P_k has shared $\mathsf{x}^{(k)}$ during the synchronous protocol PrepareInputs. Now if P_k is *honest*, then $q(\cdot)$ will be a polynomial of degree at most t and so will be the polynomial $p(\cdot)$. So even if the corrupted parties in Com share incorrect shares (points) of $q(\cdot)$, the error correction will correctly output $b(\cdot) + q(\cdot)$ and every honest party P_i will have the correct share $q(\alpha_i) = p(\alpha_i) - b(\alpha_i)$. Moreover, the input $\mathsf{x}^{(k)}$ will remain private, due to the blinding polynomial. For the complete protocol steps and the formal details, see [4].

Thus, the input phase in the hybrid model is "splitted" into two parts: the first part, namely PrepareInputs utilizes the synchronous round, while the second part restores all the inputs using the protocol Input and RestoreInputs, both of which are executed in a complete asynchronous setting. In our protocol, we will execute the protocols PrepareInputs and RestoreInputs, assuming that each party P_i has c_i inputs, where $c_i \geq t+1$. It is easy to see that protocol PrepareInputs will require a private communication of $\mathcal{O}(c_I n)$ elements from $\mathbb{F}$, as $c_1 + \ldots + c_n = c_I$. In the protocol Input, each party will now have to t-share $c_i + c_I$ values using Sh: c_i values corresponding to his real inputs and c_I values corresponding to the shares of the inputs of all the parties. This will require a total private communication of $\mathcal{O}(c_I n^2)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$. The protocol RestoreInputs will be executed on behalf of t parties not in Com. An instance of RestoreInputs executed on the behalf of a party P_i will require private reconstruction of nc_i values (namely n points on each of the c_i blinding polynomial) and public reconstruction of nc_i values (namely n points on each of the c_i blinded polynomial $p(\cdot)$). So a single instance of RestoreInputs will require a private communication of $\mathcal{O}(c_i n^2)$ elements from $\mathbb{F}$ and so for the t instances it will be of $\mathcal{O}(c_I n^2)$ elements from $\mathbb{F}$.

F.4 The Error-free AMPC Protocol

Finally, the error-free AMPC protocol HybridAMPC in the hybrid model is a sequence of the following steps:

1. The parties start executing the protocol HybPrePro from the first synchronous round to generate t-sharing of $c_M + c_R + c_I(t+1)$ random multiplication triples. Parallely, the parties execute the protocol PrepareInputs.

2. After terminating HybPrePro and PrepareInputs, the parties execute the protocol Input. In the protocol Input, the input for a party is a vector of his real input plus the shares of the inputs of the other parties received during PrepareInputs.

3. After terminating Input, the parties execute RestoreInputs to restore the t-sharing of the inputs of the parties not in the common set Com, where Com is the set of $3t+1$ parties, whose vectors are t-shared during Input.

4. After terminating RestoreInputs, the parties execute CircEval to evaluate the circuit and then terminate.