Asynchronous Multiparty Computation with Linear Communication Complexity[*]

Ashish Choudhury[1,**], Martin Hirt[2], and Arpita Patra[1,***]

[1] University of Bristol, UK
{ashish.choudhary,arpita.patra}@bristol.ac.uk
[2] ETH Zurich, Switzerland
hirt@inf.ethz.ch

Abstract. Secure multiparty computation (MPC) allows a set of n parties to securely compute a function of their private inputs against an adversary corrupting up to t parties. Over the previous decade, the communication complexity of *synchronous* MPC protocols could be improved to $\mathcal{O}(n)$ per multiplication, for various settings. However, designing an *asynchronous* MPC (AMPC) protocol with linear communication complexity was not achieved so far. We solve this open problem by presenting two AMPC protocols with the corruption threshold $t < n/4$. Our first protocol is *statistically* secure (i.e. involves a negligible error) in a completely asynchronous setting and improves the communication complexity of the previous best AMPC protocol in the same setting by a factor of $\Theta(n)$. Our second protocol is *perfectly* secure (i.e. error free) in a *hybrid* setting, where one round of communication is assumed to be synchronous, and improves the communication complexity of the previous best AMPC protocol in the hybrid setting by a factor of $\Theta(n^2)$.

Like other efficient MPC protocols, we employ Beaver's circuit randomization approach (Crypto '91) and prepare shared random multiplication triples. However, in contrast to previous protocols where triples are prepared by first generating two random shared values which are then multiplied distributively, in our approach each party prepares its own multiplication triples. Given enough such shared triples (potentially partially known to the adversary), we develop a method to extract shared triples unknown to the adversary, avoiding communication-intensive multiplication protocols. This leads to a framework of independent interest.

1 Introduction

Threshold unconditionally secure multiparty computation (MPC) is a powerful concept in secure distributed computing. It enables a set of n mutually distrusting parties to jointly and securely compute a publicly known function f of their private inputs over some finite field $\mathbb{F}$, even in the presence of a *computationally unbounded*

[*] Full version of the paper available as Cryptology ePrint Archive, Report 2012/517.
[**] This work has been supported in part by EPSRC via grant EP/I03126X.
[***] Work supported by ERC Advanced Grant ERC-2010-AdG-267188-CRIPTO.

active adversary Adv, capable of corrupting any t out of the n parties. In a general MPC protocol [7,12,20,2], f is usually expressed as an arithmetic circuit (consisting of addition and multiplication gates) over $\mathbb{F}$ and then the protocol evaluates each gate in the circuit in a shared/distributed fashion. More specifically, each party secret share its private inputs among the parties using a linear secret-sharing scheme (LSS), say Shamir [21], with threshold t; informally such a scheme ensures that the shared value remains information-theoretically secure even if upto t shares are revealed. The parties then maintain the following invariant for each gate in the circuit: *given that the input values of the gate are secret-shared among the parties, the corresponding output value of the gate also remains secret-shared among the parties.* Finally the circuit output is publicly reconstructed. Intuitively, the privacy follows since each intermediate value during the circuit evaluation remains secret-shared. Due to the *linearity* of the LSS, the addition gates are evaluated *locally* by the parties. However, maintaining the above invariant for the multiplication (non-linear) gates requires the parties to interact. The focus therefore is rightfully placed on measuring the communication complexity (i.e. the total number of elements from $\mathbb{F}$ communicated) to evaluate the multiplication gates in the circuit.

In the recent past, several efficient unconditionally secure MPC protocols have been proposed [17,3,14,5,9]. 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] as well as for the *statistical* setting [9]. 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 [9]. 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. We note that if one is willing to reduce the resilience t from the optimal resilience by a constant fraction of t, then by using techniques like packed secret-sharing [16], one can break the $\mathcal{O}(n)$ barrier as shown in [13]. However, the resultant protocols are quiet involved. An alternate approach to break the $\mathcal{O}(n)$ barrier was presented in [15], where instead of involving all the n parties, only a designated set of $\Theta(\log n)$ parties are involved for shared evaluation of each gate. However the protocol involves a negligible error in the privacy; on contrary we are interested in protocols with no error in the privacy.

Our Motivation. The above results are obtained 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 appropriately model the real-life networks like the Internet. On contrary, in the asynchronous network model [6], there are no timing assumptions and the messages can be arbitrarily delayed. The protocols in the asynchronous model are much more involved due to the following phenomenon, which is impossible to avoid in a completely 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 to be endless and so the communication from t (potentially honest) parties has to be ignored. It is well known that perfectly-secure asynchronous MPC (AMPC) is possible if and only if $t < n/4$ [6], while statistically secure AMPC is possible if and only if $t < n/3$ [8]. The best known unconditional AMPC protocol is reported in [19]. The protocol is perfectly secure with resilience $t < n/4$ and communication complexity of $\mathcal{O}(n^2)$ 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. 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 [4] to enforce "input provision", i.e. to consider the inputs of all the n parties for the computation, which is otherwise impossible in a completely asynchronous setting. The best known AMPC protocol in the hybrid setting [4] has perfect security, resilience $t < n/4$ and communication complexity of $\mathcal{O}(n^3)$ field elements per multiplication gate. Thus, our protocol significantly improves over the hybrid model protocol of [4].

2 Overview of Our Protocols

Without loss of generality, we assume $n = 4t + 1$; thus $t = \Theta(n)$. We follow the well-known "offline-online" paradigm used in most of the recent MPC protocols [3,4,14,5,9]: the offline phase produces t-sharing[1] of c_M random *multiplication triples* $\{(a^{(i)}, b^{(i)}, c^{(i)})\}_{i \in [c_M]}$ unknown to Adv, where $c^{(i)} = a^{(i)} b^{(i)}$ and c_M denotes the number of multiplication gates in the circuit. The multiplication triples are *independent* of f; so this phase can be executed well ahead of the actual circuit evaluation. Later, during the online phase the shared triples are used for the shared evaluation of the multiplication gates in the circuit, using the standard Beaver's circuit randomization technique [2] (see Sec. 4). The efficiency of the MPC protocol is thus reduced to the efficiency of generating shared random multiplication triples. Our new proposed approach for the task of generating random triples outperforms the existing ones in terms of the efficiency and simplicity.

The traditional way of generating the shared multiplication triples is the following: first the individual parties are asked to t-share random *pairs* of values

[1] A value v is d-shared (see Definition 1) if there exists a polynomial $p(\cdot)$ of degree at most d with $p(0) = v$ and every party holds a distinct point on $p(\cdot)$.

on which a "randomness extraction" algorithm (such as the one based on Vandermonde matrix [14]) is applied to generate t-sharing of "truly" random pairs $\{a^{(i)}, b^{(i)}\}_{i \in [c_M]}$. 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 individual parties to "directly" share random multiplication triples and then "extract" 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, built with the following modules:

– **Multiplication Triple Sharing (Section 7.1).** The first module allows a party P_i to "verifiably" t-share $\Theta(n)$ random multiplication triples with $\mathcal{O}(n^2)$ communication complexity and thus requires $\mathcal{O}(n)$ "overhead". The verifiability ensures that the shared triples are indeed multiplication triples. If P_i is *honest*, the shared triples remain private from Adv. Such triples, shared by the individual parties are called *local* triples.
– **Multiplication Triple Extraction (Section 7.2).** The second module allows the parties to securely extract $\Theta(n)$ t-shared random multiplication triples unknown to Adv from a set of $3t+1$ local t-shared multiplication triples with $\mathcal{O}(n^2)$ communication complexity (and thus with $\mathcal{O}(n)$ "overhead"), provided that at least $2t+1$ out of the $3t+1$ local triples are shared by the *honest* parties (and hence are random and private). We stress that known techniques for extracting shared *random values* from a set of shared random and non-random values fail to extract shared *random multiplication triples* from a set of shared random and non-random multiplication triples.

For our first module, we present two protocols: the first one *probabilistically* verifies the correctness of the shared multiplication triples, leading to our statistical AMPC protocol. The second protocol verifies the shared multiplication triples in an *error-free* fashion in a hybrid setting, leading to our perfectly-secure hybrid AMPC protocol. For the second module, we present an *error-free* triple-extraction protocol. We do not employ (somewhat complex) techniques like *player elimination* [17,5] and *dispute control* [3,14,9] in our protocols. These techniques have been used in the most recent synchronous unconditional MPC protocols to obtain linear complexity. Briefly, these techniques suggest to carry out a computation optimistically first assuming no corruption will take place and in case corruption occurs, fault/dispute is detected and memorized so that the same fault/dispute does not cause failure in the subsequent computation. However, their applicability is yet to be known in the asynchronous setting. Central to our protocols lie the following two building blocks.

Verifiable Secret Sharing with Linear Overhead (Section 5): We propose a *robust asynchronous verifiable secret sharing* (AVSS) protocol that allows a *dealer* D to "verifiably" t-share $(t+1) = \Theta(n)$ secret values with $\mathcal{O}(n^2)$ communication complexity (i.e. $\mathcal{O}(n)$ overhead). The protocol is obtained by modifying the perfectly-secure AVSS protocol of [19] that allows D to $2t$-share a *single* value. To the best of our knowledge, we are unaware of any robust secret-sharing protocol (with $t < n/4$) having linear overhead, even in the synchronous setting.

Transforming Independent Triples to Co-related Triples with Linear Overhead (Section 6): Taking $3t+1 = \Theta(n)$ t-shared input triples (which may not be multiplication triples), say $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i \in [3t+1]}$, the protocol outputs $3t+1$ t-shared triples, say $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i \in [3t+1]}$, lying on three polynomials of degree $3t/2$, $3t/2$ and $3t$ respectively. Namely, 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, 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_i values. The protocol has communication complexity $\mathcal{O}(n^2)$ (i.e. $\mathcal{O}(n)$ overhead). The protocol further ensures the following one-to-one correspondence between the input and the output triples: **(1).** the ith output triple is a *multiplication triple* if and only if the ith input triple is a multiplication triple; **(2).** the ith output triple is known to Adv if and only if the ith input triple is known to Adv. The former guarantees that the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true if and only if all the $3t+1$ input triples are multiplication triples, while the later guarantees that if Adv knows t' input triples, then it implies $\frac{3t}{2} + 1 - t'$ "degree of freedom" in the polynomials $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, provided $t' \leq \frac{3t}{2}$. The protocol is borrowed from the batch verification protocol of [9], where the goal was to *probabilistically* check whether a set of input triples are multiplication triples.

Given the above two building blocks, our first module (of the framework) is realized by asking each party P_i to invoke the AVSS protocol to generate t-sharing of $3t+1$ random multiplication triples $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i \in [3t+1]}$. All that is left is to verify if the shared triples are indeed multiplication triples. This is achieved by transforming the shared triples to $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i \in [3t+1]}$ using the triple transformation protocol and then verifying if $\mathsf{Z}(\cdot) \overset{?}{=} \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ where $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ are the underlying polynomials, associated with $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i \in [3t+1]}$. Two different methods are then proposed for the verification; one leads to our statistical AMPC and the other leads to our perfect AMPC in hybrid model.

The second module takes the set of $3t+1$ *local* t-shared multiplication triples (verifiably shared by individual parties), say $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i \in [3t+1]}$ such that at least $2t+1$ of them are shared by the *honest* parties. Using our triple transformation protocol, shared multiplication triples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})\}_{i \in [3t+1]}$ are then computed. Since all the input triples are guaranteed to be multiplication triples, the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ holds. Moreover, as Adv may know at most t input local triples, t output triples are leaked, leaving $\frac{t}{2}$ "degree of freedom" in the polynomials, which is used to extract $\frac{t}{2} = \Theta(n)$ random multiplication triples.

3 Model, Definitions and Notations

We assume a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of $n = 4t+1$ parties, connected by pairwise private and authentic channels; here t is the number of parties which can be under the control of a computationally unbounded Byzantine adversary Adv. The adversary can force the corrupted parties to deviate in any arbitrary manner during the execution of a protocol. The communication channels are asynchronous allowing 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*; to model the worst case scenario, we assume that the scheduler is under the control of Adv. The scheduler can schedule the messages exchanged between the honest parties, without having access to the "contents" of these messages.

The function f to be computed is specified as an 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 $|\mathbb{F}| \geq n^2 \cdot 2^{\kappa}$, 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 by c_M and c_R the number of multiplication and random gates in C respectively. Similar to [14,5], for the sake of efficiency, we evaluate $t+1$ multiplication gates at once in our AMPC protocol by applying the Beaver's method, assuming that the circuit is well-spread, with sufficiently many "independent" multiplication gates to evaluate in parallel. By $[X]$ we denote the set $\{1, \ldots, X\}$, while $[X, Y]$ with $Y \geq X$ denote the set $\{X, X+1, \ldots, Y\}$.

Definition 1 (d-sharing [3,4,14,5]). *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 $P_i \in \overline{\mathcal{P}}$ holds a* share *s_i of s, such that there exists a polynomial $p(\cdot)$ of degree at most d, where $p(0) = s$ and $p(\alpha_i) = s_i$ holds for every (honest) $P_i \in \overline{\mathcal{P}}$. The vector of shares 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 vector $\boldsymbol{S} = (s^{(1)}, \ldots, s^{(\ell)})$ of ℓ values is said to be d-shared among a set of parties $\overline{\mathcal{P}}$ if each $s^{(l)} \in \boldsymbol{S}$ 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 n parties. A standard property of d-sharings is its *linearity*: given sharings $[x^{(1)}]_d, \ldots, [x^{(\ell)}]_d$ and a publicly known linear function $g : \mathbb{F}^{\ell} \rightarrow \mathbb{F}^m$ w $g(x^{(1)}, \ldots, x^{(\ell)}) = (y^{(1)}, \ldots, y^{(m)})$, then $g([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d) = ([y^{(1)}]_d, \ldots, [y^{(m)}]_d)$. By saying that the parties $\mathsf{compute}$ (locally) $([y^{(1)}]_d, \ldots, [y^{(m)}]_d) = g([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d)$, we mean that every party P_i (locally) 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.

4 Existing Building Blocks

Private and Public Reconstruction of d-shared Values: Let $[v]_d^{\overline{\mathcal{P}}}$ be a d-sharing of v, shared through a polynomial $p(\cdot)$, where $d < |\overline{\mathcal{P}}| - 2t$. The *online error correction* (OEC) algorithm [6], based on the Reed-Solomon (RS) error-correction allows any designated party P_R to reconstruct $p(\cdot)$ and thus $v = p(0)$. We call the protocol as $\mathsf{OEC}(P_R, d, [v]_d^{\overline{\mathcal{P}}})$, which has communication complexity $\mathcal{O}(n)$. Moreover if P_R is *honest* then no additional information about v is leaked.

Let $\{[u^{(i)}]_d^{\overline{\mathcal{P}}}\}_{i \in [t+1]}$ be a set of d-shared values where $d < |\overline{\mathcal{P}}| - 2t$. The goal is to make *every* party in $\mathcal{P}$ reconstruct $\{u^{(i)}\}_{i \in [t+1]}$. This is achieved by protocol $\mathsf{BatRecPubl}$ with communication complexity $\mathcal{O}(n^2)$ by using the idea of "data expansion", based on RS codes, as used in [14,5].

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 $[x \cdot y]_t$ from $[x]_t$ and $[y]_t$, at the expense of two *public reconstructions*, using a *pre-computed* t-shared random multiplication triple (from the offline 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, where $\ell \geq t+1$. $\mathsf{BatRecPubl}$ is then used to efficiently perform the public reconstruction of the 2ℓ (e and d) values with a communication of $\mathcal{O}(\lceil \frac{2\ell}{t+1} \rceil \cdot n^2) = \mathcal{O}(n\ell)$ field elements. We call the protocol as $\mathsf{BatchBeaver}(\{([x^{(i)}]_t, [y^{(i)}]_t, [a^{(i)}]_t, [b^{(i)}]_t, [c^{(i)}]_t)\}_{i \in [\ell]})$.

Agreement on a Common Subset (ACS) and Asynchronous Broadcast: Protocol ACS [6,8] allows the (honest) parties to agree on a *common* subset Com of $(n-t)$ parties, who have correctly shared "values"; the values may be the inputs of the individual parties or a multiplication triple or a random value. The protocol has communication complexity $\mathcal{O}(\text{poly}(n))$.

Bracha's asynchronous broadcast protocol (called $\mathsf{A\text{-}Cast}$) [10] allows a sender $\mathsf{Sen} \in \mathcal{P}$ 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 does the same. The protocol needs a communication of $\mathcal{O}(n^2|m|)$ for a message of size $|m|$. We say that P_i receives m from the broadcast of P_j if P_i outputs m in the instance of $\mathsf{A\text{-}Cast}$ where P_j is acting as Sen.

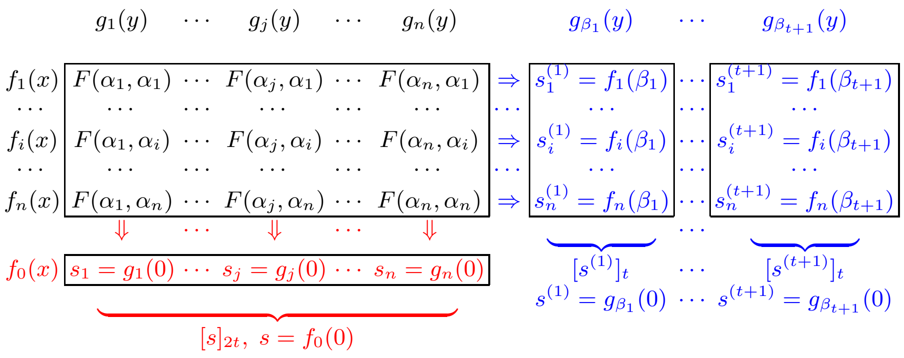
Generating a Random Value: Protocol Rand is a standard protocol to generate a uniformly random value and has communication complexity $\mathcal{O}(\text{poly}(n))$.

5 Verifiably Generating Batch of t-shared Values

We design a protocol called Sh, which allows a dealer $\mathsf{D} \in \mathcal{P}$ to "verifiably" t-share ℓ values $\boldsymbol{S} = (s^{(1)}, \ldots, s^{(\ell)})$, where $\ell \geq t+1$. The "verifiability" ensures that if the honest parties terminate the protocol then the output sharings are t-sharing. Moreover the shared secrets are private if D is *honest*. The protocol communicates $\mathcal{O}(n\ell)$ field elements and broadcasts $\mathcal{O}(n^2)$ field elements. We first explain the protocol assuming that $\boldsymbol{S}$ contains $t+1$ secrets.

The starting point of Sh is the sharing protocol of the perfectly-secure AVSS scheme of [19]. The AVSS protocol of [19] enables D to $2t$-share (note the degree of sharing) 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 $f_i(x) \overset{def}{=} F(x, \alpha_i)$ and $g_i(y) \overset{def}{=} F(\alpha_i, y)$. 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 $s' \overset{def}{=} F'(0, 0)$ to be $2t$-shared through the polynomial $f'_0(x)$ of degree at most $2t$ where $f'_0(x) \overset{def}{=} F'(x, 0)$ and every *honest* party P_j holds its share s'_j of the secret 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 $f_0(x) \overset{def}{=} F(x, 0)$.



$g_1(y)$ · · · $g_j(y)$ · · · $g_n(y)$
$g_{\beta_1}(y)$ · · · $g_{\beta_{t+1}}(y)$
$f_1(x)$ $F(\alpha_1, \alpha_1)$ · · · $F(\alpha_j, \alpha_1)$ · · · $F(\alpha_n, \alpha_1)$ ⇒ $s_1^{(1)} = f_1(\beta_1)$ · · · $s_1^{(t+1)} = f_1(\beta_{t+1})$
$f_i(x)$ $F(\alpha_1, \alpha_i)$ · · · $F(\alpha_j, \alpha_i)$ · · · $F(\alpha_n, \alpha_i)$ ⇒ $s_i^{(1)} = f_i(\beta_1)$ · · · $s_i^{(t+1)} = f_i(\beta_{t+1})$
$f_n(x)$ $F(\alpha_1, \alpha_n)$ · · · $F(\alpha_j, \alpha_n)$ · · · $F(\alpha_n, \alpha_n)$ ⇒ $s_n^{(1)} = f_n(\beta_1)$ · · · $s_n^{(t+1)} = f_n(\beta_{t+1})$
$f_0(x)$ $s_1 = g_1(0)$ · · · $s_j = g_j(0)$ · · · $s_n = g_n(0)$
$[s]_{2t},\ s = f_0(0)$
$[s^{(1)}]_t$ · · · $[s^{(t+1)}]_t$
$s^{(1)} = g_{\beta_1}(0)$ · · · $s^{(t+1)} = g_{\beta_{t+1}}(0)$

Fig. 1. Pictorial representation of the values distributed in the AVSS of [19] 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) of degree $2t$, while in Sh, $t+1$ values $s^{(1)}, \ldots, s^{(t+1)}$ are t-shared through the column polynomials $g_{\beta_1}(y), \ldots, g_{\beta_{t+1}}(y)$ (shown in blue color) of degree t.

In the above sharing protocol of [19], we note that Adv's view leaves $(t+1)(2t+1) - t(2t+1) - t = (t+1)$ "degree of freedom" in $F(x, y)$ when D is *honest*. 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 while $(t+1)(2t+1)$ distinct points are required to completely define $F(x, y)$. While [19] used the $t+1$ degree of freedom for 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 $\boldsymbol{S} = (s^{(1)}, \ldots, s^{(t+1)})$, 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 its shares of the secrets in $\boldsymbol{S}$ by local computation on 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 $s_i^{(l)} = g_{\beta_l}(\alpha_i) = f_i(\beta_l)$.

So all that is left is to ensure that *every honest* P_i gets $f_i(x)$ in Sh protocol. For this recall that the sharing protocol of [19] 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 where $F'(\alpha_j, y) = g'_j(y)$ holds; furthermore for an honest D, $F'(x, y) = F(x, y)$ holds. Now note that $g'_j(\alpha_i)$ is the same as $f'_i(\alpha_j)$ and thus every P_j holds a point on every $f'_i(x)$. Now P_i can reconstruct $f'_i(x)$ 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. Finally, we note that for a *corrupted* D, $\boldsymbol{S'} = (F'(\beta_1, 0), \ldots, F'(\beta_{t+1}, 0))$ will be t-shared and for an honest D, $\boldsymbol{S'} = \boldsymbol{S}$ will hold.

Our idea of embedding several secrets in a single *bi-variate* polynomial is different from the notion of *packed secret-sharing* [16] where k secrets are embedded in a single *univariate* polynomial of degree t and each party receives a single share (a distinct point on the polynomial). In the latter, a single share is the share for k secrets and the robust reconstruction of the secrets is possible *only if* at most $t - k + 1$ parties are corrupted. Protocol Sh, on the other hand, ensures that *each* secret in $\boldsymbol{S}$ is *independently* t-shared and thus the robust reconstruction of each secret is possible even when the adversary corrupts t parties.

Sharing More Than $t + 1$ Values Together: On having ℓ secrets 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 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 [19] describes the same idea to keep the broadcast communication independent of ℓ when D $2t$-shares ℓ secrets. 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.

6 Transforming Independent Triples to Co-related Triples

Protocol $\mathsf{TripTrans}$ takes as input a set of $(3t + 1)$ "independent" shared triples, say $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i \in [3t+1]}$, and outputs a set of $(3t + 1)$ "co-related" shared triples, say $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t)\}_{i \in [3t+1]}$, such that: **(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 $\mathsf{Z}(\alpha_i) = \mathbf{z}^{(i)}$ holds, for $i \in [3t + 1]$. **(b)** The ith output triple $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{z}^{(i)})$ is a multiplication triple if and only if the ith input triple $(x^{(i)}, y^{(i)}, z^{(i)})$ is a multiplication triple. This further implies that $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ 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 $\frac{3t}{2} + 1 - t'$ "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 (see Fig. 2) is inherited from the protocol for the batch verification of the multiplication triples proposed in [9]. 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 $i \in [\frac{3t}{2}+1]$.

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

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

[a] This is a linear function.

Fig. 2. Transforming independent shared triples to co-related shared triples

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 $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t)\}_{i\in[\frac{3t}{2}+1]}$. 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 when all the input triples are multiplication triples; this is because for $i \in [\frac{3t}{2}+2, 3t+1]$, $\mathbf{x}^{(i)} = \mathsf{X}(\alpha_i) \neq x^{(i)}$ 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 $\{([x^{(i)}]_t, [y^{(i)}]_t, [z^{(i)}]_t)\}_{i\in[\frac{3t}{2}+2,3t+1]}$ used for the Beaver's technique are never touched before for any computation.

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 $[\mathbf{z}^{(i)}]_t$, Adv will learn $\mathbf{x}^{(i)}$, $\mathbf{y}^{(i)}$ and $\mathbf{z}^{(i)}$. 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 $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$, leaving $\frac{3t}{2} + 1 - t'$ degree of freedom in these polynomials. We note that all the honest parties eventually terminate the protocol and the protocol incurs communication of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$.

7 The Framework for Generating Multiplication Triples

We are now ready to present our new framework for generating t-sharing of $c_M + c_R$ random multiplication triples unknown to Adv, which requires communication of $\mathcal{O}((c_M + c_R)n)$ and broadcast of $\mathcal{O}(n^3)$ field elements. As discussed earlier, the framework consists of two modules, elaborated next.

7.1 Module I: Verifiably Sharing Multiplication Triples

A Probabilistic Solution in a Completely Asynchronous Setting: Our protocol $\mathsf{TripleSh}$ allows a party $\mathsf{D} \in \mathcal{P}$ to verifiably share multiplication triples with linear "overhead", where the verification resorts to a probabilistic approach. In the protocol, D is asked to t-share $3t + 1$ random multiplication triples $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i \in [3t+1]}$ using Sh. To check if the triples are multiplication triples, the shared triples are first transformed to $\{([\mathbf{x}^{(i)}]_t, [\mathbf{y}^{(i)}]_t, [\mathbf{z}^{(i)}]_t)\}_{i \in [3t+1]}$ via $\mathsf{TripTrans}$ and then the relation $\mathsf{Z}(\cdot) \overset{?}{=} \mathsf{X}(\cdot) \cdot \mathsf{Y}(\cdot)$ is verified through a public checking of $\mathsf{Z}(\alpha) \overset{?}{=} \mathsf{X}(\alpha) \cdot \mathsf{Y}(\alpha)$ for a random α. To ensure that no *corrupted* D can pass this test, α should be generated using Rand once D completes sharing of the triples. It follows via the property of $\mathsf{TripTrans}$ that if some of the input triples $\{(x^{(i)}, y^{(i)}, z^{(i)})\}_{i \in [3t+1]}$ are not multiplication triples, then $\mathsf{Z}(\alpha) \neq \mathsf{X}(\alpha) \cdot \mathsf{Y}(\alpha)$ except with probability at most $\frac{3t}{|\mathbb{F}|}$ for a random α, since $\mathsf{Z}(\alpha)$ is of degree at most $3t$. Moreover if D is honest then Adv will learn only one point on $\mathsf{X}(\cdot), \mathsf{Y}(\cdot)$ and $\mathsf{Z}(\cdot)$ (namely at α) leaving $\frac{3t}{2}$ "degree of freedom" in these polynomials. So if the verification passes, then the parties output $\frac{3t}{2}$ shared triples $\{([\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t)\}$ on the "behalf" of D, where $\mathbf{a}^{(i)} = \mathsf{X}(\beta_i), \mathbf{b}^{(i)} = \mathsf{Y}(\beta_i)$ and $\mathbf{c}^{(i)} = \mathsf{Z}(\beta_i)$ for $\frac{3t}{2}$ β_is distinct from the random α.

In $\mathsf{TripleSh}$, the above idea is applied on ℓ batches of $3t + 1$ t-shared triples, where $\ell \geq t + 1$ and a single random α is used for all the ℓ batches. Using $\mathsf{BatRecPubl}$, we then efficiently perform the public reconstruction of 3ℓ values, namely the values of the polynomials at α. The protocol thus outputs $\ell \cdot \frac{3t}{2} = \Theta(n\ell)$ shared multiplication triples, with communication complexity $\mathcal{O}(n^2\ell)$ and requires broadcast of $\mathcal{O}(n^2)$ elements from $\mathbb{F}$.

An Error-Free Solution in a Hybrid Setting: An inherent drawback of a completely asynchronous setting is that the inputs of up to t potentially honest parties may get ignored. To get rid of this, [4] 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 (with some additional technicalities). We further utilize the first synchronous round to present an *error-free* triple sharing protocol called $\mathsf{HybTripleSh}$ for t-sharing multiplication triples.

$\mathsf{HybTripleSh}$ follows the footstep of $\mathsf{TripleSh}$, except that it verifies the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ in an error-free fashion, by leaking at most t points on the polynomials to Adv. Since this leaves at least $\frac{t}{2}$ degree of freedom on each of the polynomials for an *honest* D, the parties output $\frac{t}{2}$ shared multiplication triples $\{[\mathbf{a}^{(i)}]_t, [\mathbf{b}^{(i)}]_t, [\mathbf{c}^{(i)}]_t\}_{i \in [\frac{t}{2}]}$ on the behalf of D after successful verification, where $\mathbf{a}^{(i)} = \mathsf{X}(\beta_i), \mathbf{b}^{(i)} = \mathsf{Y}(\beta_i)$ and $\mathbf{c}^{(i)} = \mathsf{Z}(\beta_i)$. The idea for the error-free verification is the following: *each* party P_i is given "access" to the triple $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ and is given the responsibility of confirming if it is a multiplication triple. If the confirmation comes from *all* the parties, then it can be concluded that the relation $\mathsf{Z}(\cdot) = \mathsf{X}(\cdot)\mathsf{Y}(\cdot)$ is true. This is because the confirmation comes from at least $(n - t) = 3t + 1$ *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 $3t$. Moreover, at most t values on each polynomial are leaked to Adv through the t corrupted parties (for an *honest* D). Unfortunately, in a completely asynchronous setting, we cannot wait for the confirmation from all the parties in $\mathcal{P}$, as the wait may turn out to be endless[2]. The synchronous round in the hybrid setting comes to our rescue.

In the synchronous round, every party P_i is asked to "non-verifiably" t-share a $\mathsf{dummy\ multiplication\ triple}$, say $(f^{(i)}, g^{(i)}, h^{(i)})$ which is used later to verify if $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is a multiplication triple on behalf of P_i, although *without further participation* of P_i. 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 however ensures that a dummy triple is non-verifiably shared on the behalf of *every* party P_i. Even if a corrupted P_i 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. By defining "good" dummy triples as the ones that are t-shared and are multiplication triples, we now show how the verification is carried out using these dummy triples. Note that the honest parties share good dummy triples.

Given a dummy triple $(f^{(i)}, g^{(i)}, h^{(i)})$, we check if $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ is a multiplication triple by computing the sharing of the product of $\mathsf{X}(\alpha_i)$ and $\mathsf{Y}(\alpha_i)$ via the Beaver's technique and using the shared dummy triple and then publicly verifying if the resultant product is the same as $\mathsf{Z}(\alpha_i)$. The latter can be verified by checking if the difference of the product and $\mathsf{Z}(\alpha_i)$ is 0 or not. If P_i is *honest* then the dummy triple is random and thus no information is leaked about $(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$. If the checking fails, then the sharing of

[2] The confirmation is needed from all the n parties as we need $3t + 1$ "true" confirmations and t corrupted parties may provide a "false" confirmation.

$(\mathsf{X}(\alpha_i), \mathsf{Y}(\alpha_i), \mathsf{Z}(\alpha_i))$ 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 found to be a non-multiplication triple then D is definitely corrupted in which case the protocol is 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 dummy 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 losing 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 dummy triple due to the reason that it is not t-shared correctly is more intricate to handle. The problem could be during the reconstruction of the values that are not t-shared, while executing the Beaver's technique. We solve this problem via 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 when the input sharing is not t-shared. So an *asynchronous Byzantine agreement* (ABA) protocol [1] is run to agree on a unique value.

Finally we note that in the protocol $\mathsf{HybTripleSh}$, the above idea is actually applied on ℓ batches of $3t+1$ t-shared triples in parallel, where $\ell \geq t+1$. This allows the efficient public reconstruction of all the required sharings (corresponding to the ℓ batches) using $\mathsf{BatRecPubl}$. The protocol thus outputs $\ell \cdot \frac{t}{2} = \Theta(n\ell)$ shared multiplication triples.

7.2 Module II: Extracting Random Multiplication Triples

Let $\mathsf{Com} \subset \mathcal{P}$ be a publicly known set of $3t+1$ parties, such that every party in Com has verifiably t-shared ℓ multiplication triples among the parties in $\mathcal{P}$, where the triples shared by the honest parties are random and unknown to Adv. Protocol $\mathsf{TripExt}$ then "extracts" $\ell \cdot \frac{t}{2} = \Theta(n\ell)$ *random* t-shared multiplication triples unknown to Adv from these $\ell \cdot (3t+1)$ "local" t-shared multiplication triples with a communication of $\mathcal{O}(n^2\ell)$. The 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 $\mathsf{TripTrans}$ 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 $\left\{([\mathsf{X}^l(\beta_i)]_t, [\mathsf{Y}^l(\beta_i)]_t, [\mathsf{Z}^l(\beta_i)]_t)\right\}_{i\in[\frac{t}{2}]}$.

7.3 Module I + Module II ⇒ Preprocessing (Offline) Phase

Our preprocessing phase protocol now consists of the following 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, either using an instance TripleSh (if it is a completely asynchronous setting) or an an instance of HybTripleSh (if it is a hybrid setting). **(2)** The parties then execute an instance of ACS to decide on a common set Com of $3t+1$ dealers who have correctly shared multiplication triples in their respective instances of TripleSh/HybTripleSh. **(3)** Finally the parties execute the triple-extraction protocol TripExt on the triples shared by the parties in Com to extract $\ell \cdot \frac{t}{2} = (c_M + c_R)$ random shared multiplication triples. Now depending upon whether we use the protocol TripleSh or HybTripleSh above, we get either a completely asynchronous preprocessing phase protocol PreProc involving an error of at most $t \cdot \frac{3t}{|\mathbb{F}|} = \frac{3t^2}{|\mathbb{F}|}$ in the output or an error-free preprocessing phase protocol HybPrePro for the hybrid setting. The output triples will be private, as the multiplication triples of the honest dealers in Com are random and private.

8 The New AMPC Protocols

Once we have a preprocessing phase protocol, the online phase protocol for the shared circuit evaluation is straight forward (as discussed in the introduction); we refer to the full version of the paper for complete details. We note that in our hybrid AMPC protocol, during the offline phase, apart from t-sharing of $(c_M + c_R)$ random multiplication triples, the parties generate t-sharing of $n \cdot (t+1)$ *additional* multiplication triples. The additional triples are used to enforce "input provision" from *all* the n parties during the online phase by using the method of [4]; see the full version of the paper for details.

Theorem 1 (The AMPC Theorem). *Let $f : \mathbb{F}^n \rightarrow \mathbb{F}$ be a function expressed as an arithmetic circuit over a finite field $\mathbb{F}$, consisting of c_M and c_R multiplication and random gates. Then for every possible* Adv*, there exists a statistical AMPC protocol with error probability at most $2^{-\kappa}$ to securely compute f, provided $|\mathbb{F}| \geq \max\{3t^2 \cdot 2^\kappa, 2n\}$ for a given error parameter κ. The protocol incurs communication of $\mathcal{O}((c_M+c_R)n)$ elements and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$ and requires two invocations to* ACS *and n invocations to* Rand.

If the first communication round is synchronous, then there exists a perfect AMPC protocol to securely compute f, provided $|\mathbb{F}| \geq 2n$. In the protocol, the inputs of all (the honest) parties are considered for the computation. The protocol requires communication of $\mathcal{O}((c_M+c_R)n+n^3)$ and broadcast of $\mathcal{O}(n^3)$ elements from $\mathbb{F}$. It also requires two invocations to ACS *and n^2 invocations to ABA.*

References

1. Abraham, A., Dolev, D., Halpern, J.: An almost-surely terminating polynomial protocol for asynchronous Byzantine agreement with optimal resilience. In: PODC, pp. 405–414 (2008)

2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)
3. Beerliová-Trubíniová, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006)
4. Beerliová-Trubíniová, Z., Hirt, M.: Simple and efficient perfectly-secure asynchronous MPC. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 376–392. Springer, Heidelberg (2007)
5. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
6. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC, pp. 52–61 (1993)
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: STOC, pp. 1–10 (1988)
8. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: PODC, pp. 183–192 (1994)
9. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 663–680. Springer, Heidelberg (2012)
10. Bracha, G.: An asynchronous [(n-1)/3]-resilient consensus protocol. In: PODC, pp. 154–162 (1984)
11. Canetti, R.: Studies in secure multiparty computation and applications. PhD thesis, Weizmann Institute, Israel (1995)
12. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols. In: STOC, pp. 11–19. ACM (1988)
13. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010)
14. Damgård, I.B., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007)
15. Dani, V., King, V., Movahedi, M., Saia, J.: Brief announcement: Breaking the $\mathcal{O}(nm)$ bit barrier, secure multiparty computation with a static adversary. In: PODC, pp. 227–228 (2012)
16. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: STOC, pp. 699–710 (1992)
17. Hirt, M., Maurer, U.M., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)
18. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes. North-Holland Publishing Company (1978)
19. Patra, A., Choudhury, A., Rangan, C.P.: Communication efficient perfectly secure VSS and MPC in asynchronous networks with optimal resilience. In: Bernstein, D.J., Lange, T. (eds.) AFRICACRYPT 2010. LNCS, vol. 6055, pp. 184–202. Springer, Heidelberg (2010); full version available as Cryptology ePrint Archive, Report 2010/007
20. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: STOC, pp. 73–85 (1989)
21. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)