Perfectly-Secure Asynchronous MPC for General Adversaries (Extended Abstract)

Ashish Choudhury*[1] and Nikhil Pappu[1]

International Institute of Information Technology Bangalore, India.
ashish.choudhury@iiitb.ac.in, nikhil.pappu@iiitb.org

Abstract. We study perfectly-secure *Multiparty Computation* (MPC) in the *asynchronous* communication setting, tolerating a generalized *non-threshold* adversary, characterized by an *adversary structure*. Ashwin Kumar et al (ACISP2002) presented a condition which is both necessary as well as sufficient for the existence of perfectly-secure MPC in this setting. However, we show that their protocol is flawed and present a new protocol (with the same necessity condition). Moreover, our protocol is conceptually simpler, and unlike their protocol, does not rely on *monotone span programs* (MSPs). As a sub-contribution, we also present an *asynchronous Byzantine agreement* protocol (tolerating a non-threshold adversary), which is used as a key component in our MPC protocol.

Keywords: Secure MPC, General Adversary Structures, Asynchronous Protocols, Byzantine Agreement, Non-threshold Model

1 Introduction

Secure MPC [8, 14, 25, 42, 46] is a widely studied problem in secure distributed computing. Informally, an MPC protocol allows a set of n mutually distrusting parties to compute any agreed upon function of their inputs, while keeping their respective inputs as private as possible. Due to its generality and powerful abstraction, the MPC problem has been widely studied and various interesting results have been achieved related to the theoretical possibility and practical feasibility of MPC protocols. A large bulk of MPC literature assumes a *threshold* adversary, i.e., an adversary that can corrupt any t out of the n parties. In [29], Hirt and Maurer initiated the study of MPC in the *non-threshold* adversarial model under a more general constraint on the adversary's corruption capability, where the adversary is allowed to corrupt any set of parties from a *pre-defined* collection of subsets of $\mathcal{P}$ called a *general adversary structure* (where $\mathcal{P}$ is the set of all parties) and presented necessity and sufficiency conditions for the same

* This research is an outcome of the R&D work undertaken in the project under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation (formerly Media Lab Asia).

in various settings. Although such a specification comes with the downside that there exist adversary structures for which known MPC protocols have communication complexities polynomial in the size of the adversary structure (which could be exponential in n) and the computation complexity of any MPC protocol is lower bounded by the size of the adversary structure [30], the added flexibility makes it applicable in more real-world scenarios, especially when n is not large.

The communication complexity of the MPC protocol of [29] is super-polynomial in the size of the adversary structure. Subsequent works presented polynomial time protocols [19,20,23,38,44] and focused on further improving the communication complexity [32,36]. These protocols can be categorized into two based on the secret-sharing scheme [43] they deploy. The protocols presented in [19,20,36,44] deploy a secret-sharing scheme based on *monotone span programs* (MSPs) [33], while those presented in [32,38] use a simpler additive secret-sharing scheme. The significance of both of these stems from the fact that for a given adversary structure, one might be more efficient compared to the other (see for e.g., [36]).

Our Motivation. All the above works on general adversaries are in the *synchronous* model, where the delays of messages in the network are bounded by a publicly known constant. However, real-life networks like the Internet are modelled more appropriately by the *asynchronous* communication model [11], where messages can be arbitrarily delayed with only the guarantee that messages of honest parties are delivered eventually. The *asynchronous* MPC (AMPC) problem has been studied, but in the *threshold* adversary setting [4,7,9,16,17,21,31,39,45]. These protocols are more involved and less efficient than their synchronous counterparts as in a completely asynchronous setting, from the view point of an honest party, it is impossible to distinguish between a slow but honest sender (whose messages are delayed arbitrarily) and a corrupt sender (who does not send any message). Consequently, to avoid an endless wait, no party can afford to receive messages from all its neighbours, thus ignoring communication from potentially slow honest parties. An MPC protocol tolerating a generalized adversary which is secure in an asynchronous network, would better model real-world scenarios due to the increased flexibility arising from relaxed assumptions on both the adversary and the underlying network. Adapting general adversary MPC protocols for the asynchronous setting was mentioned as an open problem in [28,44]. We focus on the design of an AMPC protocol with the highest level of security, namely *perfect-security* tolerating a *computationally unbounded malicious* adversary, characterized by a general adversary structure.

Existing Results and Our Contributions. The only work in the domain of AMPC in the non-threshold model is due to [34], which presented a necessary and sufficient condition for perfectly-secure AMPC. They showed that for perfect security (where the security properties are achieved in an error-free fashion even if the adversary is computationally unbounded), the set of parties $\mathcal{P}$ should satisfy the $\mathcal{Q}^{(4)}$ condition. That is, the union of *any four* subsets from the underlying adversary structure should not "cover" the entire party set $\mathcal{P}$. In this paper, our main contributions are as follows:

- The AMPC protocol of [34] is based on MSPs and utilizes the *player-elimination* framework, commonly deployed against a *threshold* adversary in the *synchronous* communication setting (see for example [5, 26]). However, we show that the player-elimination framework will not necessarily work, which further implies that the protocol of [34] is flawed (see Section 3).
- We present a *new* perfectly-secure AMPC protocol with the $\mathcal{Q}^{(4)}$ condition. The protocol is conceptually simpler than the protocol of [34] and does not deploy MSPs or utilize a player-elimination framework. The computation and communication complexities of the protocol are polynomial in n and the size of the underlying adversary structure. The multiplication protocols that have been deployed in the existing non-threshold protocols in the *synchronous* setting for evaluating multiplication gates [32, 38], will *not* work in the asynchronous communication setting. Intuitively, this is because the security of the synchronous protocols depends upon the availability of the messages of *all* honest parties, which is not guaranteed in the asynchronous setting. Hence, we design a new sub-protocol for securely evaluating multiplication gates asynchronously (see Section 6).
- As a sub-contribution, we present an *asynchronous Byzantine agreement* (ABA) protocol in the non-threshold model. The protocol is almost-surely terminating [1, 2]. That is, if the parties keep on running the protocol, then *asymptotically*, they terminate with probability 1. ABA is used as a sub-protocol, both in the protocol of [34] as well as ours. However, [34] does not provide any ABA protocol and simply states that the existing ABA protocol(s) in the threshold setting [13] can be generalized in a straightforward fashion for the non-threshold setting. Unfortunately, it turns out that the generalization is non-trivial and involves few subtleties. For instance, we find that generalizing these protocols for the non-threshold setting requires a *non-constant* expected number of asynchronous "rounds", while only a *constant* number of such rounds are required by existing *threshold* protocols [13, 15].

We follow the offline/online phase paradigm [3]. The offline phase (also called the preprocessing phase) generates additively-shared random and private multiplication triples, independent of the function f. These triples are later used in the online phase for securely evaluating the circuit representing f. While this paradigm is the de facto standard for designing generic MPC protocols, the *new* components are the protocols for perfectly-secure *asynchronous verifiable secret-sharing* (AVSS), asynchronous perfectly-secure multiplication and ABA, for instantiating the paradigm in the *non-threshold* setting. Due to space constraints, we are unable to give complete formal proofs for our protocols in this extended abstract and we defer the formal proofs to the full version of the paper.

2 Preliminaries

We assume a set of n mutually distrusting parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, connected by pair-wise private and authentic channels. The distrust in the system is modelled by a *computationally-unbounded* adversary Adv, specified by an adversary structure $\mathcal{Z} \subseteq 2^{\mathcal{P}}$, where each $Z \in \mathcal{Z}$ satisfies the condition that $Z \subset \mathcal{P}$. The adversary structure $\mathcal{Z}$ is *monotone* in the sense that if $Z \in \mathcal{Z}$, then any $Z' \subseteq Z$ also belongs to $\mathcal{Z}$. For convenience, we assume that $\mathcal{Z}$ consists of *maximal* subsets of parties, which can be potentially corrupted by Adv during the execution of a protocol. Consequently, we denote the adversary structure as $\mathcal{Z} = \{Z_1, \ldots, Z_q\}$, where $Z_1, \ldots, Z_q$ are maximal subsets of potentially corruptible parties. For $m = 1, \ldots, q$, we use the notation $\mathcal{G}_m$ to denote the set of parties $\mathcal{P} \setminus Z_m$. The adversary Adv can corrupt a set of parties $Z^\star \subset \mathcal{P}$ during the protocol execution, where $Z^\star$ is part of some set in the adversary structure (i.e., $Z^\star \subseteq Z$ for some $Z \in \mathcal{Z}$). The parties not under the control of Adv are called *honest*. The adversary is Byzantine (malicious) and can force the parties under its control to deviate from the protocol instructions in any arbitrary fashion. Note that the exact identity of $Z^\star$ won't be known to the honest parties at the beginning of the execution of any protocol, and need not be revealed at the end of the protocol.

We say that a set of parties $S \subseteq \mathcal{P}$ satisfies the $\mathcal{Q}^{(k)}$ condition, if $S \not\subseteq Z_1 \cup \ldots \cup Z_k$, for every $Z_1, \ldots, Z_k \in \mathcal{Z}$. Notice that if S satisfies the $\mathcal{Q}^{(k)}$ condition, then it also satisfies the $\mathcal{Q}^{(k')}$ condition for any $1 \leq k' < k$. We assume that the set of parties $\mathcal{P}$ satisfies the $\mathcal{Q}^{(4)}$ condition, which is necessary for the existence of any perfectly-secure AMPC protocol tolerating $\mathcal{Z}$ [34].

In our protocols, all the computations are performed over some finite algebraic structure $\mathcal{K}$, which could be a finite ring or a field and we assume that the parties want to compute a function f, represented by a publicly known arithmetic circuit cir over $\mathcal{K}$. For simplicity and without loss of generality, we assume that each party $P_i \in \mathcal{P}$ has a single input $x^{(i)}$ to the function f, and there is a single output $y = f(x^{(1)}, \ldots, x^{(n)})$, which is supposed to be learnt by all the parties. Apart from the input and output gates, cir consists of 2-input gates of the form $g = (x, y, z)$, where x and y are the inputs and z is the output. The gate g can either be an addition gate (i.e. $z = x + y$) or a multiplication gate (i.e. $z = x \cdot y$). The circuit cir consists of M multiplication gates.

We follow the *asynchronous* communication model of [7,11], which does not put any restriction on the message delays and the only guarantee is that the messages of the honest parties are delivered *eventually*. The sequence of message delivery is controlled by an adversarial scheduler. Due to the absence of any globally known upper bound on the message delays, no party can wait to receive messages from *all* its neighbours to avoid an endless wait (as a corrupt neighbour may not send any message). Hence, any party has to proceed as soon as it receives messages from a set of parties in S, where $\mathcal{P} \setminus S \in \mathcal{Z}$.

2.1 Definitions

Definition 2.1 ($[\cdot]$-sharing). *A value $s \in \mathcal{K}$ is said to be $[\cdot]$-shared, if there exist values $s^{(1)}, \ldots, s^{(q)} \in \mathcal{K}$ where $s = s^{(1)} + \ldots + s^{(q)}$, such that for each $m = 1, \ldots, q$, all (honest) parties in the group $\mathcal{G}_m$ hold the share $s^{(m)}$. The notation $[s]$ denotes the vector of shares $(s^{(1)}, \ldots, s^{(q)})$.*

Note that a party P_i may possess more than one share in the vector $[s]$, depending upon the number of groups $\mathcal{G}_m$ in which P_i is present, which further depends upon the adversary structure $\mathcal{Z}$. It is easy to see that $[\cdot]$-sharings are linear: given $[a], [b]$ and public constants $c_1, c_2 \in \mathcal{K}$, the parties can *locally* compute their shares corresponding to $[c_1 \cdot a + c_2 \cdot b]$. In general, the parties can locally compute any *publicly* known linear function of $[\cdot]$-shared values.

In our protocols, we come across situations where there exists a *publicly* known value $s \in \mathcal{K}$ and the parties have to take some default $[\cdot]$-sharing of s.

Definition 2.2 (Default $[\cdot]$-sharing). *Let $s \in \mathcal{K}$ be publicly known. Then the vector $(s, 0, \ldots, 0$ $(q-1$ times$))$ is considered as a default $[\cdot]$-sharing of s. That is, the parties in $\mathcal{G}_1$ consider s as their share corresponding to $\mathcal{G}_1$, while for $m = 2, \ldots, q$, the parties in $\mathcal{G}_m$ consider 0 as their share corresponding to $\mathcal{G}_m$.*

We next recall a data-structure from [34] used in our secret-sharing protocol.

Definition 2.3 ($\mathcal{Z}-$clique [34]). *Let $G = (V, E)$ be an undirected graph where $V \subseteq \mathcal{P}$. Then a subset $V' \subseteq V$ is called a $\mathcal{Z}-$clique in G, if the following hold:*
- *The set of nodes V' constitute a clique in G. That is, for every $P_i, P_j \in V'$, the edge $(P_i, P_j) \in E$.*
- *The set of nodes $Z' \stackrel{def}{=} V \setminus V'$ is a subset of some set from $\mathcal{Z}$. That is, $Z' \subseteq Z_m$ for some $Z_m \in \mathcal{Z}$.*

To find whether there exists a $\mathcal{Z}-$clique in a given graph G, one can check whether the set of nodes in $V \setminus Z_m$ constitutes a clique in G for any $Z_m \in \mathcal{Z}$. The running time of this algorithm is $\mathcal{O}(|\mathcal{Z}| \cdot \text{poly}(n))$.

Definition 2.4 (ABA [37]). *Let Π be an asynchronous protocol for the parties in $\mathcal{P}$, where each party P_i has a binary input x_i and a binary output σ_i. Then, Π is said to be an ABA protocol if the following hold, where the probability is taken over the random coins and inputs of the honest parties and* Adv.
- **Termination***: If all honest parties invoke Π, then with probability 1, all honest parties eventually terminate*[1] *Π.*
- **Agreement***: $\sigma_i = \sigma_j$ holds for every honest P_i and P_j.*
- **Validity***: If all honest parties have the same input $x \in \{0, 1\}$, then $\sigma_i = x$ holds for every honest P_i.*

For designing our ABA protocol, we use another primitive called common coin.

Definition 2.5 (Common Coin (CC) [13]). *Let Π be an asynchronous protocol for the parties in $\mathcal{P}$, where each party has some local random input and a binary output. Then Π is called a p-common coin protocol, if the following holds:*
- **Correctness**: *For every value $\sigma \in \{0, 1\}$, with probability at least p, all honest parties output σ.*
- **Termination**: *If all honest parties invoke Π, then all honest parties eventually terminate Π.*

[1] The classic FLP impossibility result [22] implies that any deterministic ABA protocol will have non-terminating executions where honest parties do not terminate even if a single party is corrupted. As a result, the best that one can hope for is that the protocol terminates with probability 1 (i.e. almost-surely). See [11] for more details.

Formalizing the security definition of MPC is subtle, and in itself is an interesting field of research. In the synchronous setting, the standard definition is based on the *universally-composable* (UC) real-world/ideal-world based simulation paradigm [12]. On a very high level, any protocol Π_{real} for MPC is defined to be secure in this paradigm, if it "emulates" what is called as an ideal-world protocol Π_{ideal}. In Π_{ideal}, all the parties give their respective inputs for the function f to be computed to a *trusted third party* (TTP), who locally computes the function output and sends it back to all the parties and hence, no communication is involved among the parties in Π_{ideal}. Protocol Π_{real} is said to emulate Π_{ideal} if for any adversary attacking Π_{real}, there exists an adversary attacking Π_{ideal} that induces an identical *output* in Π_{ideal}, where the *output* is the concatenation of the outputs of the honest parties and the view of the adversary [24].

Extending the above definition to the asynchronous setting brings a lot of additional technicalities to deal with the eventual message delivery in the system, controlled by an adversarial scheduler. In the case of the asynchronous setting, the local output of the honest parties is only an approximation of the pre-specified function f over a subset $\mathcal{C}$ of the local inputs, the rest being taken to be 0, where $\mathcal{P} \setminus \mathcal{C} \in \mathcal{Z}$ (this is analogous to the definition of asynchronous MPC in the threshold setting [7, 9, 11]). Protocol Π_{real} is said to be *perfectly-secure* in the asynchronous setting if the local outputs of the honest players are correct, Π_{real} terminates eventually with probability 1 for all honest parties, and the output of Π_{real} is identically distributed with respect to the output of Π_{ideal} (which involves a TTP that computes an approximation of f). We refer to [18] for the complete formalization of the UC-security definition of MPC in the asynchronous communication setting, with eventual message delivery. As the main focus of the paper is to present a simple asynchronous MPC protocol, to avoid bringing in additional technicalities, we defer giving the security proofs of our protocols as per the asynchronous UC framework to the full version of the paper.

2.2 Existing Asynchronous Primitives

Asynchronous Reliable Broadcast (Acast). The protocol allows a designated sender $\mathsf{S} \in \mathcal{P}$ to *asynchronously* send some message m identically to all the parties, even in the presence of Adv. If S is *honest*, then every honest party eventually terminates with output m. If S is *corrupt and* some honest party terminates with output $m^{\star}$, then eventually every honest party terminates with output $m^{\star}$. In the threshold setting, Bracha [10] presented an asynchronous reliable broadcast protocol tolerating $t < n/3$ corruptions. The protocol is generalized for a $\mathcal{Q}^{(3)}$ adversary structure in [35]. The protocol needs a communication of $\mathcal{O}(\mathsf{poly}(n) \cdot \ell)$ bits for broadcasting an ℓ-bit message. We stress that the termination guarantees of the Acast protocol are "asynchronous" in the sense that if some *honest* P_i terminates an Acast instance with some output, then the further participation of P_i in the instance is *no* longer necessary to ensure the eventual termination (and output computation) of the other honest parties.

We say "P_i *broadcasts* m" to mean that $P_i \in \mathcal{P}$ acts as an S and invokes an instance of protocol Acast to broadcast m, and the parties participate in this instance. The notation "P_j *receives* $m^\star$ *from the broadcast of* P_i" means that P_j terminates an instance of protocol Acast invoked by P_i as S, with output $m^\star$.

Agreement on a Common Subset (ACS). In our asynchronous protocols, we come across situations of the following kind: there exists a set of parties $S \subseteq \mathcal{P}$ where S satisfies the $\mathcal{Q}^{(k)}$ condition with $k \geq 2$. Each party in S is supposed to act as a dealer and verifiably-share some value(s). While the honest dealers in S invoke the required sharing instance(s), the corrupt dealers in S may not do the same. To avoid an endless wait, the parties should terminate immediately after completing the sharing instances of a subset of dealers S', where $S \setminus S' \subseteq Z$, for some $Z \in \mathcal{Z}$. However, the set Z might be different for different honest parties, as the order in which the parties terminate various sharing instances may differ, implying that the subset S' might be different for different honest parties. Protocol ACS allows the honest parties to agree on a common subset S' satisfying the above properties. The primitive was introduced in [7] in the threshold setting and generalized for the non-threshold setting in [34]. The complexity of the protocol is equivalent to that of $|S| = \mathcal{O}(n)$ ABA instances.

Beaver's Circuit-Randomization. We use the Beaver's circuit-randomization method [3] to evaluate multiplication gates in our MPC protocol. If the underlying secret-sharing scheme is linear (which is the case for $[\cdot]$-sharing), then the method allows for evaluation of a multiplication gate with secret-shared inputs at the expense of publicly reconstructing two secret-shared values, using an auxiliary secret-shared multiplication triple. In more detail, let $g = (x, y, z)$ be a multiplication gate such that the parties hold $[x]$ and $[y]$, and the goal is to compute a $[\cdot]$-sharing of $z = x \cdot y$. Moreover, let $([a], [b], [c])$ be a shared multiplication triple available with the parties, such that $c = a \cdot b$. We note that $z = (x - a + a) \cdot (y - b + b)$ and hence $z = (x - a) \cdot (y - b) + b \cdot (x - a) + a \cdot (y - b) + a \cdot b$. Based on this idea, to compute $[z]$, the parties first locally compute $[d] = [x - a] = [x] - [a]$ and $[e] = [y - b] = [y] - [b]$, followed by publicly reconstructing d and e. The parties then locally compute $[z] = d \cdot e + d \cdot [b] + e \cdot [a] + [c]$.

If a and b are random and private, then the view of the adversary remains independent of x and y. Namely, even after learning d and e, the privacy of the gate inputs and output is preserved. We denote this protocol as $\mathsf{Beaver}(([x], [y]), ([a], [b], [c]))$, which can be executed in a completely asynchronous setting. If the underlying public reconstruction protocol terminates for the honest parties (which will be the case for $[\cdot]$-sharing), then protocol Beaver eventually terminates for all the honest parties. The complexity of the protocol is equivalent to that of two instances of publicly reconstructing a $[\cdot]$-shared value.

3 The Flaw in the AMPC Protocol of [34]

The protocol of [34] deploys the player-elimination framework [27], commonly deployed against *threshold* adversaries in the *synchronous* communication setting for obtaining efficient protocols (see for example [5, 26]). As part of the framework, the AMPC protocol of [34] deploys several *non-robust* sub-protocols which succeed with all the honest parties receiving the correct output if the potentially corrupt parties behave honestly. Else, all the honest parties agree upon a subset of *conflicting* parties, which is either a triplet or a pair of parties, such that it is guaranteed to consist at least one corrupt party. We stress that the adversary has the flexibility to decide the choice of (corrupt and honest) parties who make it to the conflicting set. The non-robust sub-protocols are executed repeatedly until they succeed, each time with a *new* set of parties, which is obtained by discarding the conflicting set of parties from the previously considered set of parties. Each time a conflicting set is obtained, the adversary structure also gets updated for the next iteration by *excluding* subsets (from the adversary structure) which have zero overlap with the conflicting set.

In [34], it is claimed that the *updated* party set still satisfies the $\mathcal{Q}^{(4)}$ condition with respect to the *updated* adversary structure after every update, which is necessary for maintaining security in the subsequent invocations of the non-robust sub-protocols. However, we show that this need not be the case, implying the breach of security in the subsequent invocations. Consider the following adversary structure $\mathcal{Z}$ over the set of parties $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5, P_6, P_7\}$:

$$\mathcal{Z} = \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_1, P_5, P_6\}, \{P_7\}\}$$

Clearly, $\mathcal{P}$ satisfies the $\mathcal{Q}^{(4)}$ condition. Let the parties agree upon the conflicting set $\{P_1, P_2, P_7\}$. Then the updated party set will be $\mathcal{P}' = \{P_3, P_4, P_5, P_6\}$, while as per [34], the updated $\mathcal{Z}$ remains the same. Now, it is easy to see that $\mathcal{P}'$ does not satisfy even the $\mathcal{Q}^{(3)}$ condition with respect to $\mathcal{Z}$, as $\mathcal{P}' \subseteq \{P_1, P_3\} \cup \{P_1, P_4\} \cup \{P_1, P_5, P_6\}$. Hence, the instances of Acast and ABA (which are used with the non-robust sub-protocols) in the subsequent invocations fail, as the existence of the $\mathcal{Q}^{(3)}$ condition is necessary for the security of Acast and ABA.

4 Perfectly-Secure AVSS

We begin by presenting a protocol for the sharing phase.

4.1 Sharing Protocol

We present a protocol called Sh, which allows a designated dealer $\mathsf{D} \in \mathcal{P}$ to verifiably $[\cdot]$-share a value $s \in \mathcal{K}$. The protocol eventually terminates for an *honest* D. The verifiability here ensures that even for a *corrupt* D, if some honest party terminates, then there exists some fixed value, say $s^\star$ (which could be different from s), such that $s^\star$ is eventually $[\cdot]$-shared among the parties. The protocol proceeds as follows: D first creates a q-out-of-q additive sharing of s and generates the shares $s^{(1)}, \ldots, s^{(q)}$. The share $s^{(m)}$ is given to *all* the parties in the group $\mathcal{G}_m$. This distribution of shares maintains the privacy of s for an honest D, as there exists *at least* one group, say $\mathcal{G}_m$, consisting only of *honest* parties whose corresponding share $s^{(m)}$ will not be known to the adversary.

While the above distribution of information is sufficient for an *honest* D to generate $[s]$, a potentially *corrupt* D may distribute "inconsistent" shares to the honest parties. So the parties publicly verify whether D has distributed a common share to *all* the (honest) parties in every group $\mathcal{G}_m$, without revealing any additional information about this share. For this, each $P_i \in \mathcal{G}_m$ upon receiving a share $s_i^{(m)}$ from D, sends it to every party in $\mathcal{G}_m$ to check whether they received the same share from D as well, which should be the case for an honest D. If $\mathcal{G}_m$ consists of only honest parties and if D is honest, then this exchange of information in $\mathcal{G}_m$ does not reveal any information about the share $s^{(m)}$. The parties in $\mathcal{G}_m$ then publicly confirm the pair-wise consistency of their common share. That is, a party P_i broadcasts an $\mathtt{OK}_m(P_i, P_j)$ message, if P_i receives the value $s_j^{(m)}$ from $P_j \in \mathcal{G}_m$ (namely, the share which P_j received from D) and finds that $s_i^{(m)} = s_j^{(m)}$ holds. Similarly, P_j broadcasts an $\mathtt{OK}_m(P_j, P_i)$ message, if P_j receives the value $s_i^{(m)}$ from P_i and finds that $s_i^{(m)} = s_j^{(m)}$ holds.

The next step is to check if "sufficiently" many parties in each $\mathcal{G}_m$ confirm the receipt of a common share from D, for which the broadcasted $\mathtt{OK}_m(\star, \star)$ messages are used. To avoid an endless wait, the parties cannot afford to wait and receive a confirmation from *all* the parties in $\mathcal{G}_m$, as corrupt parties in $\mathcal{G}_m$ may not respond. Hence, the parties wait for confirmations only from a subset of parties $\mathcal{C}_m \subseteq \mathcal{G}_m$, such that the parties excluded from $\mathcal{C}_m$ (who either do not respond or respond with negative confirmations) constitute a potential adversarial subset from $\mathcal{Z}$. Due to the asynchronous nature of communication, the parties may get confirmations in different order and as a result, different parties may have different versions of $\mathcal{C}_m$. To ensure that all the parties agree on a *common* $\mathcal{C}_m$ set, D is assigned the task of collecting the positive confirmations from the parties in $\mathcal{G}_m$, followed by preparing the set $\mathcal{C}_m$ and broadcasting it. We call the subsets $\mathcal{C}_m$ as "core" sets to signify that the parties in these subsets have publicly confirmed the receipt of a common share on the behalf of the group $\mathcal{G}_m$.

To construct $\mathcal{C}_m$, D prepares a "consistency graph" G_m over the set of parties in $\mathcal{G}_m$ based on the $\mathtt{OK}_m(\star, \star)$ messages and searches for the presence of a $\mathcal{Z}-\mathsf{clique}$ in G_m. As soon as a $\mathcal{Z}-\mathsf{clique}$ in G_m is found, the set of parties belonging to the $\mathcal{Z}-\mathsf{clique}$ is assigned as $\mathcal{C}_m$ and broadcasted. Upon receiving $\mathcal{C}_m$ from the broadcast of D, the parties check its validity. For this, every party constructs its own local copy of the consistency graph G_m based on the various broadcasted $\mathtt{OK}_m(\star, \star)$ messages and checks whether the parties in $\mathcal{C}_m$ constitute a $\mathcal{Z}-\mathsf{clique}$ in G_m. Once the parties receive a valid $\mathcal{C}_m$ for each $\mathcal{G}_m$, they ensure that D has distributed a common share, say $s^{(m)}$, to all the (honest) parties in $\mathcal{C}_m$. However, this does not conclude the $[\cdot]$-sharing of D's secret, as in each $\mathcal{G}_m$, the parties not included in $\mathcal{C}_m$ (i.e., parties in $\mathcal{G}_m \setminus \mathcal{C}_m$) may not possess $s^{(m)}$. Hence, the final step is to ensure that even the excluded parties possess $s^{(m)}$.

To get the correct $s^{(m)}$, the parties excluded from $\mathcal{C}_m$ (i.e., parties in $\mathcal{G}_m \setminus \mathcal{C}_m$) take the "help" of the parties in $\mathcal{C}_m$. More specifically, as part of the pair-wise consistency check, P_i receives the value $s_j^{(m)}$ from "several" parties in $\mathcal{C}_m$. Party P_i checks if there is a subset of parties $Q = \mathcal{C}_m \setminus Z$ for some $Z \in \mathcal{Z}$, such that all the parties P_j in Q reported the same $s_j^{(m)}$ to P_i, in which case P_i sets $s^{(m)}$ to this common value $s_j^{(m)}$. The idea is that Q satisfies the $\mathcal{Q}^{(1)}$ condition (since $\mathcal{C}_m$ satisfies the $\mathcal{Q}^{(2)}$ condition) and hence, includes at least one honest party who sends the correct value of the missing share $s^{(m)}$ to P_i.

Protocol $\mathsf{Sh}(\mathsf{D}, s)$

- **Distribution of shares by** D: The following code is executed only by D.
  - On having the input $s \in \mathcal{K}$, randomly select $s^{(1)}, \ldots, s^{(q)} \in \mathcal{K}$, subject to the condition that $s = s^{(1)} + \ldots + s^{(q)}$.
  - For $m = 1, \ldots, q$, send the share $s^{(m)}$ to every party P_i in the group $\mathcal{G}_m$.
- **Pair-wise consistency**: For $m = 1, \ldots, q$, the parties in $\mathcal{G}_m$ do the following.
  - Each $P_i \in \mathcal{G}_m$ on receiving the share $s_i^{(m)}$ (for the group $\mathcal{G}_m$) from D, sends $s_i^{(m)}$ to every party $P_j \in \mathcal{G}_m$. On receiving $s_j^{(m)}$ from P_j, broadcast $\mathtt{OK}_m(P_i, P_j)$, provided $s_j^{(m)} = s_i^{(m)}$ holds.
- **Identification of core sets**: For $m = 1, \ldots, q$, D does the following.
  - Initialize a set $\mathcal{C}_m$ to $\emptyset$. Additionally, construct an undirected graph G_m with the parties in $\mathcal{G}_m$ as the vertex set.
  - Add the edge (P_i, P_j) to G_m, if $\mathtt{OK}_m(P_i, P_j)$ and $\mathtt{OK}_m(P_j, P_i)$ are received from the broadcast of P_i and P_j respectively, where $P_i, P_j \in \mathcal{G}_m$.
  - Keep updating the graph G_m by repeatedly executing the above step till a $\mathcal{Z}-\mathsf{clique}$, say $\mathcal{W}_m$, is identified in G_m.
  - Once a $\mathcal{Z}-\mathsf{clique}$ $\mathcal{W}_m$ is identified, assign $\mathcal{C}_m := \mathcal{W}_m$.

  Upon computing non-empty $\mathcal{C}_1, \ldots, \mathcal{C}_q$, broadcast these sets.
- **Verification of core sets**: The parties wait to receive non-empty sets $\mathcal{C}_1, \ldots, \mathcal{C}_q$ from the broadcast of D. Upon receiving them, the parties verify each of these sets $\mathcal{C}_m$ for $m = 1, \ldots, q$ by executing the following steps.
  - Construct an undirected graph G_m with the parties in $\mathcal{G}_m$ as the vertex set.
  - For every $P_i, P_j \in \mathcal{C}_m$, wait for messages $\mathtt{OK}_m(P_i, P_j)$ and $\mathtt{OK}_m(P_j, P_i)$ from P_i and P_j respectively. On receiving them, add the edge (P_i, P_j) to G_m.
  - Mark $\mathcal{C}_m$ as *valid*, if the parties in $\mathcal{C}_m$ constitute a $\mathcal{Z}-\mathsf{clique}$ in graph G_m.
- **Share computation and termination**: If the sets $\mathcal{C}_1, \ldots, \mathcal{C}_q$ are marked as valid, then for $m = 1, \ldots, q$, each party P_i in each group $\mathcal{G}_m$ executes the following steps to compute its share corresponding to $\mathcal{G}_m$.
  - If $P_i \in \mathcal{C}_m$, then set $s^{(m)} := s_i^{(m)}$, where $s_i^{(m)}$ has been received from D.
  - Else, set $s^{(m)}$ to the common value, say $s_j^{(m)}$, received from a subset of parties $P_j \in \mathcal{C}_m \setminus Z$ for some $Z \in \mathcal{Z}$, during the pair-wise consistency check of shares for the group $\mathcal{G}_m$.

  Each party $P_i \in \mathcal{P}$, upon computing the share corresponding to each $\mathcal{G}_m$ such that $P_i \in \mathcal{G}_m$, terminates the protocol.

Fig. 1: The Secret-Sharing Protocol.

Theorem 4.1. *Let $\mathcal{P}$ satisfy the $\mathcal{Q}^{(4)}$ condition. Moreover, let D have input s in protocol Sh. Then, the following holds in protocol Sh.*

- **Termination***: If D is honest, then each honest party eventually terminates the protocol. On the other hand, if D is corrupt and some honest party terminates, then eventually, every other honest party terminates the protocol.*

- **Correctness**: *If some honest party terminates, then there exists some fixed value $s^\star \in \mathcal{K}$, where $s^\star = s$ for an honest* D, *such that $s^\star$ is eventually $[\cdot]$-shared among the parties.*
- **Privacy**: *If* D *is honest, then the view of* Adv *is independent of s.*
- **Communication Complexity**: *The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot \mathsf{poly}(n))$ elements from $\mathcal{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(|\mathcal{Z}| \cdot \mathsf{poly}(n))$ bits.*

4.2 Reconstruction Protocol

Let $s \in \mathcal{K}$ be a value which is $[\cdot]$-shared. Protocol Rec allows the parties to reconstruct s. Let $[s] = (s^{(1)}, \ldots, s^{(q)})$. To reconstruct s, party P_i needs to obtain the shares $s^{(1)}, \ldots, s^{(q)}$. While P_i holds all the shares $s^{(k)}$ of s corresponding to the groups $\mathcal{G}_k$ where $P_i \in \mathcal{G}_k$, party P_i needs to obtain the missing shares $s^{(m)}$ for the groups $\mathcal{G}_m$ where $P_i \notin \mathcal{G}_m$. For this, all the parties in $\mathcal{G}_m$ send the share $s^{(m)}$ to P_i. Let $s_j^{(m)}$ be the value received by P_i from $P_j \in \mathcal{G}_m$. Party P_i checks if there is a subset of parties $Q = \mathcal{G}_m \setminus Z$ for some $Z \in \mathcal{Z}$, such that all the parties P_j in Q reported the same value $s_j^{(m)}$ to P_i, in which case P_i sets $s^{(m)}$ to this common value $s_j^{(m)}$. The idea is that the set Q will satisfy the $\mathcal{Q}^{(1)}$ condition (since $\mathcal{G}_m$ satisfies the $\mathcal{Q}^{(2)}$ condition) and hence, will include at least one honest party P_j, who sends the correct value of the missing share $s^{(m)}$ to P_i.

Protocol $\mathsf{Rec}([s])$

Let $[s] = (s^{(1)}, \ldots, s^{(q)})$ where $s = s^{(1)} + \ldots + s^{(q)}$.

- **Exchanging the shares**: For $m = 1, \ldots, q$, the parties in $\mathcal{G}_m$ do the following.
  - Send the share $s^{(m)}$ to every party in $\mathcal{P} \setminus \mathcal{G}_m$.
- **Computing the missing shares**: Each $P_i \in \mathcal{P}$ does the following.
  - Corresponding to every group $\mathcal{G}_m$ for which $P_i \notin \mathcal{G}_m$, set $s^{(m)}$ to be the common value received from a set of parties in $\mathcal{G}_m \setminus Z$, for some $Z \in \mathcal{Z}$.
  - Output $s = \displaystyle\sum_{\mathcal{G}_k : P_i \in \mathcal{G}_k} s^{(k)} + \sum_{\mathcal{G}_m : P_i \notin \mathcal{G}_m} s^{(m)}$ and terminate.

Fig. 2: The Reconstruction Protocol.

The properties of protocol Rec are stated in Lemma 4.2.

Lemma 4.2. *Let s be a value which is $[\cdot]$-shared among a set of parties $\mathcal{P}$, such that $\mathcal{P}$ satisfies the $\mathcal{Q}^{(3)}$ condition. Then in protocol* Rec, *the following holds.*
- *Each honest party eventually terminates the protocol with output s.*
- *The protocol needs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot \mathsf{poly}(n))$ elements from $\mathcal{K}$.*

5 Asynchronous Byzantine Agreement (ABA)

Our ABA protocol follows the traditional route of building ABA via common coin (CC) protocol [6,10,41], which in turn reduces to AVSS. We therefore begin with the design of a CC protocol in the non-threshold setting.

5.1 Common Coin (CC) Protocol

The CC protocol CC (Fig. 3) consists of two stages. In the first stage, a uniformly random, yet *unknown* value Coin_i over $\{0, \ldots, n-1\}$ is "attached" to every party P_i. Then, once it is ensured that a "sufficiently large" number of parties FS have been attached with their respective Coin values, in the second stage, these Coin values are *publicly* reconstructed, and an output bit is computed taking into account these reconstructed values. However, due to the asynchronous nature of communication, each (honest) party may have a *different* FS set and hence, a potentially different output bit. To circumvent this problem, the protocol ensures that there is a *non-empty* overlap among the contents of the FS sets of all (honest) parties. Ensuring this common overlap is the crux of the protocol.

The existing CC protocols in the *threshold* setting [1, 2, 13, 40] ensure that the overlap is some *constant* fraction of the number of parties n, which in turn guarantees that the "success probability" (namely the probability with which all honest parties have the same final output bit) is a *constant* fraction. This property further guarantees that the resultant ABA protocol in the threshold setting requires a constant expected number of asynchronous rounds of communication. Unfortunately, it is not clear whether a straightforward generalization of the threshold CC protocol for the non-threshold setting, will always lead to a *non-empty* overlap among the FS sets of all the (honest) parties, for *every* possible adversary structure $\mathcal{Z}$. Hence, we modify the steps of the protocol so that it is *always* guaranteed (irrespective of $\mathcal{Z}$) that there exists at least a *non-empty* overlap among the FS sets of all the (honest) parties, ensuring that the success probability is at least $\frac{1}{n}$ (this leads to an ABA protocol which requires $\mathcal{O}(n^2)$ expected number of asynchronous communication rounds). The details follow.

The first stage is implemented by making each party act as a dealer to run n instances of Sh and share n random values from $\mathcal{K}$, one on the behalf of each party. To ensure that $\mathsf{Coin}_i \in \{0, \ldots, n-1\}$, the parties set $\mathcal{K}$ to either a finite ring or a field, where $|\mathcal{K}| \geq n$. Each party P_i creates a dynamic set of *accepted dealers* $\mathcal{AD}_i$, which includes all the dealers whose Sh instances terminate for P_i. The termination property of Sh guarantees that these dealers are eventually included in the accepted-dealer set of every other honest party as well. Party P_i then waits for a "sufficient" number of dealers to be accepted, such that $\mathcal{AD}_i$ is guaranteed to contain at least one *honest* dealer. For this, P_i keeps on expanding $\mathcal{AD}_i$ until $\mathcal{P} \setminus \mathcal{AD}_i \in \mathcal{Z}$ holds (which eventually happens for an honest P_i), thus guaranteeing that the resultant $\mathcal{AD}_i$ satisfies the $\mathcal{Q}^{(1)}$ condition. Once $\mathcal{AD}_i$ achieves this property, P_i assigns $\mathcal{AD}_i$ to the set AD_i and publicly announces the same. This is interpreted as P_i having *attached* the set of dealers AD_i to itself. Then, the summation of the values modulo n shared by the dealers in AD_i on the behalf of P_i, is set to be Coin_i. The value Coin_i won't be known to anyone at this point (including P_i), as the value(s) shared by the honest dealer(s) in AD_i on the behalf of P_i is(are) not yet known, owing to the privacy property of Sh.

On receiving the set AD_j from P_j, each P_i verifies if the set is "valid" by checking if the Sh instances of dealers in AD_j terminate for P_i. That is, $\mathsf{AD}_j \subseteq \mathcal{AD}_i$ holds. Once the validity of AD_j is confirmed, P_i publicly "approves" the same by broadcasting an $\texttt{OK}$ message for P_j (this implicitly means P_i's approval for the yet unknown, but well defined value Coin_j). Party P_i then waits for the approval of AD_j from a set of parties S_j *including itself*, such that $\mathcal{P} \setminus S_j \in \mathcal{Z}$, guaranteeing that S_j satisfies the $\mathcal{Q}^{(2)}$ condition. After this, P_j is included by P_i in a dynamic set of *accepted parties* $\mathcal{AP}_i$. Notice that the acceptance of P_j by P_i implies the eventual acceptance of P_j by every other honest party, as the corresponding approval (namely the $\texttt{OK}$ messages) for AD_j are publicly broadcasted. Waiting for an approval for P_j (and hence Coin_j) from a set of parties which satisfies the $\mathcal{Q}^{(2)}$ condition, ensures that sufficiently many honest parties have approved P_j. Later, this property is crucial to ensure a non-empty overlap among the FS sets of honest parties. Party P_i keeps on expanding its accepted-party set $\mathcal{AP}_i$ until $\mathcal{P} \setminus \mathcal{AP}_i \in \mathcal{Z}$ holds and then publicly announces it with a $\texttt{Ready}$ message and the corresponding $\mathcal{AP}_i$ set, denoted by AP_i.

On receiving the $\texttt{Ready}$ message and AP_j from P_j, each P_i verifies if the set is "valid" by checking if the parties in AP_j (and hence the corresponding Coin values) are accepted by P_i itself; i.e. $\mathsf{AP}_j \subseteq \mathcal{AP}_i$ holds. Upon successful verification, P_j is included by P_i in a dynamic set of *supportive parties* $\mathcal{SP}_i$. The interpretation of $\mathcal{SP}_i$ is that each party in $\mathcal{SP}_i$ is "supporting" the beginning of the second stage of the protocol, by presenting a sufficiently-large valid set of accepted-parties (coins). Notice that the inclusion of P_j to $\mathcal{SP}_i$ implies the eventual inclusion of P_j by every other honest party in its respective $\mathcal{SP}$ set. Once the set of supportive-parties becomes sufficiently large, i.e. $\mathcal{P} \setminus \mathcal{SP}_i \in \mathcal{Z}$ holds, P_i sets a boolean indicator $\texttt{Flag}_i$ to 1, marking the beginning of the second stage. Let SP_i denote the set of supportive-parties $\mathcal{SP}_i$ when $\texttt{Flag}_i$ is set to 1.

The second stage involves publicly reconstructing the unknown Coin values which were accepted by P_i till this point. Let FS_i be defined to be the set of accepted-parties $\mathcal{AP}_i$ when $\texttt{Flag}_i$ is set to 1. This implies that the union of the AP_j sets of all the parties in SP_i is a subset of FS_i, as each $\mathsf{AP}_j \subseteq \mathcal{AP}_i$. The parties proceed to reconstruct the value Coin_k corresponding to each $P_k \in \mathsf{FS}_i$. For this, the parties start executing the corresponding Rec instances, that are required for reconstructing the secrets shared by the accepted-dealers AD_k on the behalf of P_k. If any of the Coin_k values turns out to be 0, P_i sets the overall output to 0, else, it outputs 1 and terminates the protocol.

To argue that there exists a non-empty overlap among the FS sets of the honest parties, we consider the first *honest* party P_i to broadcast a $\texttt{Ready}$ message and claim that the set AP_i will be the common overlap (see Lemma 5.3). The termination of CC by an honest P_i may hamper the termination of other honest parties, as the corresponding Rec instances required to reconstruct the set of accepted Coin values by other honest parties may not terminate. This is because the termination of a Rec instance is not necessarily guaranteed if some *honest* parties do not participate. To circumvent this, before terminating, P_i publicly announces it along with the corresponding SP_i and FS_i sets. Any party who has not yet terminated the protocol, upon receiving these sets, locally verifies their validity and tries to compute its final output in the same way as done by P_i.

Protocol CC

All computations in the protocol are done over $\mathcal{K}$, which is either a finite ring or a field, with $|\mathcal{K}| \geq n$.

– The following code is executed by each $P_i \in \mathcal{P}$:

1. For $1 \leq j \leq n$, choose a random secret $s_{ij} \in_R \mathcal{K}$ on the behalf of P_j, and as a D, invoke an instance of $\mathsf{Sh}(P_i, s_{ij})$ of Sh. Denote this invocation by Sh_{ij}. Participate in the invocations Sh_{jk} for every $P_j, P_k \in \mathcal{P}$.
2. Initialize a set of *accepted dealers* $\mathcal{AD}_i$ to $\emptyset$. Add a party P_j to $\mathcal{AD}_i$, if Sh_{jk} has terminated for all $1 \leq k \leq n$. Wait until $\mathcal{P} \setminus \mathcal{AD}_i \in \mathcal{Z}$. Then, assign $\mathsf{AD}_i = \mathcal{AD}_i$ and broadcast the message $(\mathtt{Attach}, \mathsf{AD}_i, P_i)$. The set AD_i is considered to be the *set of dealers attached* to P_i. Let $\mathsf{Coin}_i \stackrel{\text{def}}{=} (\sum_{P_j \in \mathsf{AD}_i} s_{ji})$ mod n. We say[a] that the *coin* Coin_i *is attached to party* P_i.
3. If the message $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ is received from the broadcast of P_j, then broadcast a message $\mathtt{OK}(P_i, P_j)$, if all the dealers attached to P_j are accepted by P_i, i.e. $\mathsf{AD}_j \subseteq \mathcal{AD}_i$ holds.
4. Initialize a set of *accepted parties* $\mathcal{AP}_i$ to $\emptyset$. Add P_j to $\mathcal{AP}_i$, if the $\mathtt{OK}(\star, P_j)$ message is received from the broadcast of a set of parties S_j including P_i, such that $\mathcal{P} \setminus S_j \in \mathcal{Z}$. Wait until $\mathcal{P} \setminus \mathcal{AP}_i \in \mathcal{Z}$. Then, assign $\mathsf{AP}_i = \mathcal{AP}_i$ and broadcast the message $(\mathtt{Ready}, P_i, \mathsf{AP}_i)$.
5. Consider P_j to be *supportive* and include it in the set $\mathcal{SP}_i$ (initialized to $\emptyset$), if P_i receives the message $(\mathtt{Ready}, P_j, \mathsf{AP}_j)$ from the broadcast of P_j and each party in AP_j is accepted by P_i, i.e. $\mathsf{AP}_j \subseteq \mathcal{AP}_i$ holds. Wait until $\mathcal{P} \setminus \mathcal{SP}_i \in \mathcal{Z}$. Then, set $\mathtt{Flag}_i = 1$ (initialized to 0). Let SP_i and FS_i denote the contents of $\mathcal{SP}_i$ and $\mathcal{AP}_i$ respectively, when $\mathtt{Flag}_i$ becomes[b] 1.
6. Wait until $\mathtt{Flag}_i = 1$. Then, reconstruct the value of the coin attached to each party in FS_i as follows:
   – Start participating in the instances $\mathsf{Rec}(P_j, s_{jk})$ corresponding to each $P_j \in \mathsf{AD}_k$, such that $P_k \in \mathsf{FS}_i$, Denote this instance of Rec as Rec_{jk} and let r_{jk} be the corresponding output.
   – For every $P_k \in \mathsf{FS}_i$, compute $\mathsf{Coin}'_k = (\sum_{P_j \in \mathsf{AD}_k} r_{jk}) \mod n$.
7. Wait until the coins attached to all the parties in FS_i are computed. If there exists a party $P_k \in \mathsf{FS}_i$ where $\mathsf{Coin}'_k = 0$, then output 0. Else, output 1. Then, broadcast the message $(\mathtt{Terminate}, P_i, \mathsf{SP}_i, \mathsf{FS}_i)$ followed by terminating CC.
8. If a message $(\mathtt{Terminate}, P_j, \mathsf{SP}_j, \mathsf{FS}_j)$ is received from the broadcast of P_j, then check the following conditions:
   – $\mathsf{SP}_j \subseteq \mathcal{SP}_i$ and $\mathsf{FS}_j \subseteq \mathcal{AP}_i$ hold.
   – The value of Coin'_k attached to every $P_k \in \mathsf{FS}_j$ is computed.

   If the above conditions hold, then compute the output as follows and terminate: If for some $P_k \in \mathsf{FS}_j$, $\mathsf{Coin}'_k = 0$, then output 0. Else, output 1.

[a] The value of Coin_i will not be known to anyone at this step, including P_i.

[b] Note that $\bigcup_{P_j \in \mathsf{SP}_i} \mathsf{AP}_j \subseteq \mathsf{FS}_i$ holds, as each $\mathsf{AP}_j \subseteq \mathcal{AP}_i$.

Fig. 3: The Common Coin Protocol

The properties of protocol CC are stated in Lemmas 5.1 - 5.5. In all these lemmas, we assume that the set of parties $\mathcal{P}$ satisfies the $\mathcal{Q}^{(4)}$ condition.

Lemma 5.1. *If each honest party invokes protocol* CC*, then each honest party eventually terminates* CC*.*

Lemma 5.2. *Once some honest party P_i receives the message* $(\texttt{Attach}, \mathsf{AD}_k, P_k)$ *from the broadcast of any party P_k, then a unique value Coin_k is fixed such that the following holds:*

- *All honest parties attach Coin_k with P_k.*
- *The value Coin_k is distributed uniformly over $\{0, \ldots, n-1\}$ and is independent of the values attached with the other parties.*

Lemma 5.3. *Once some honest party sets its* $\texttt{Flag}$ *to 1, then there exists a set, say $\mathcal{M}$, such that:* **(1):** *$\mathcal{P} \setminus \mathcal{M} \in \mathcal{Z}$;* **(2):** *For each $P_j \in \mathcal{M}$, some honest party receives the message $(\texttt{Attach}, \mathsf{AD}_j, P_j)$ from the broadcast of P_j.* **(3):** *Whenever any honest party P_i sets its $\texttt{Flag}_i = 1$, it holds that $\mathcal{M} \subseteq \mathsf{FS}_i$.*

Lemma 5.4. *If all the honest parties have completed the protocol, then for every value $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{n}$, all the honest parties output σ.*

Lemma 5.5. *The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot \mathsf{poly}(n))$ elements from $\mathbb{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(|\mathcal{Z}| \cdot \mathsf{poly}(n))$ bits.*

5.2 Voting Protocol

For designing the ABA protocol using the blueprint of [6, 10, 41], we need another sub-protocol called the voting protocol. Informally, the voting protocol does "whatever can be done deterministically" to reach agreement. In a voting protocol, every party has a single bit as input. The protocol tries to find whether there is a detectable majority for some value among the inputs of the parties. In the protocol, each party's output can have *five* different forms:

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

The voting protocol ensures the following properties: **(1):** If each honest party has the same input σ, then every honest party outputs $(\sigma, 2)$; **(2):** If some honest party outputs $(\sigma, 2)$, then every other honest party outputs either $(\sigma, 2)$ or $(\sigma, 1)$; **(3):** If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$, then each honest party outputs either $(\sigma, 1)$ or $(\Lambda, 0)$.

In [13], a voting protocol is presented in the threshold setting tolerating $t < n/3$ corruptions and which requires a constant number of asynchronous "rounds" of communication. The protocol can be easily generalized for the non-threshold setting, if the set of parties $\mathcal{P}$ satisfies the $\mathcal{Q}^{(3)}$ condition. We defer the formal details to the full version of the paper.

5.3 Asynchronous Byzantine Agreement (ABA) Protocol

Once we have the protocols CC and Vote, we get an ABA protocol (see Fig 4) by generalizing the blueprint of [6,10,41]. The ABA protocol proceeds in iterations, where in each iteration, the parties execute an instance of Vote and CC. For the first iteration, the inputs of the parties are their inputs for the ABA protocol. For subsequent iterations, the inputs are decided based on the outcome of Vote and CC of the previous iteration as follows: If the output of Vote is $(\sigma, 1)$ then a party sticks to input σ for the next iteration, else it sets the output of CC as its input for the next iteration. This process is repeated until the party obtains output $(\sigma, 2)$ during an instance of Vote, in which case it broadcasts σ. Finally, once a party receives σ from the broadcast of a set of parties in some set S_{acc} such that $\mathcal{P} \setminus S_{acc} \in \mathcal{Z}$ (implying that the set S_{acc} includes at least one honest party), it outputs σ and terminates. The termination guarantees of protocol Acast ensure that the same set of broadcast messages are eventually received by every other honest party, leading to their termination as well. The idea here is that if all *honest* parties start with the same input bit, then the instance of Vote during the first iteration enables them to reach agreement on this common bit. Else, the instance of CC ensures that all honest parties have the same input bit for the next iteration with probability at least $\frac{1}{n}$.

Protocol ABA

– Each $P_i \in \mathcal{P}$ executes the following code:

1. Set $r = 0$ and $v_i = x_i$, where $x_i \in \{0, 1\}$ is the input bit for party P_i.
2. Repeat until terminating: (each iteration is considered as a round)
   a. Set $r = r + 1$. Participate in Vote with input v_r and wait for its termination. Let this instance of Vote be denoted by Vote_r. Let (y_r, m_r) denote the output of Vote_r.
   b. Invoke CC and wait until its termination. Let this instance of CC be denoted by CC_r. Let c_r denote the output of CC_r.
   c. Consider the following cases:
      i. If $m_r = 2$, then set $v_{r+1} = y_r$ and broadcast the message (`Terminate with` v_r). Participate in only one more instance of Vote and only one more instance of CC.
      ii. If $m_r = 1$, then set $v_{r+1} = y_r$.
      iii. Else, set $v_{r+1} = c_r$.
   d. Upon receiving the message (`Terminate with` σ) for some value σ from the broadcast of parties in some set S_{acc} such that $\mathcal{P} \setminus S_{acc} \in \mathcal{Z}$, output σ and terminate the protocol.

Fig. 4: The ABA Protocol in the Non-threshold Setting

Theorem 5.6. *Let the set of parties $\mathcal{P}$ satisfy the $\mathcal{Q}^{(4)}$ condition. Then protocol* ABA *is an ABA protocol with expected running time of $\mathsf{R} = \mathcal{O}(n^2)$. The protocol incurs an expected communication complexity of $\mathcal{O}(\mathsf{R} \cdot |\mathcal{Z}| \cdot \mathsf{poly}(n))$ elements from $\mathbb{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(\mathsf{R} \cdot |\mathcal{Z}| \cdot \mathsf{poly}(n))$ bits.*

6 Perfectly-Secure Preprocessing Phase

In this section, we present a preprocessing phase protocol with perfect security. We first present a perfectly-secure multiplication protocol, which allows the parties to securely generate a $[\cdot]$-sharing of the product of two $[\cdot]$-shared values.

6.1 Perfectly-Secure Multiplication

Let a and b be two $[\cdot]$-shared values. Protocol Mult (Fig 5) allows the parties to securely generate a $[\cdot]$-sharing of $[a \cdot b]$. The high level idea of the protocol is as follows. Let $[a] = (a^{(1)}, \ldots, a^{(q)})$ and $[b] = (b^{(1)}, \ldots, b^{(q)})$. Then, the protocol securely computes a $[\cdot]$-sharing of each of the terms $a^{(l)} \cdot b^{(m)}$, after which the parties set $[a \cdot b]$ to the sum of the $[\cdot]$-sharing of each of the terms $a^{(l)} \cdot b^{(m)}$. The computation of a $[\cdot]$-sharing of $[a^{(l)} \cdot b^{(m)}]$ happens as follows. Let $\mathcal{Q}_{l,m}$ be the set of parties who own both $a^{(l)}$ as well as $b^{(m)}$. Since $\mathcal{P}$ satisfies the $Q^{(4)}$ condition, it follows that $\mathcal{Q}_{l,m}$ satisfies the $\mathcal{Q}^{(2)}$ condition. Each party in the set $\mathcal{Q}_{l,m}$ is asked to independently share $a^{(l)} \cdot b^{(m)}$ by executing an instance of Sh.

Due to the asynchronous nature of communication, the parties cannot afford to terminate the Sh instances of *all* the parties in $\mathcal{Q}_{l,m}$, as the corrupt parties in $\mathcal{Q}_{l,m}$ may not invoke their Sh instances. So, the parties invoke an instance of ACS to agree on a common subset of parties $\mathcal{R}_{l,m}$ of $\mathcal{Q}_{l,m}$, whose Sh instances eventually terminate, such that the set of excluded parties $\mathcal{Q}_{l,m} \setminus \mathcal{R}_{l,m}$ belongs to $\mathcal{Z}$. Notice that $\mathcal{R}_{l,m}$ satisfies the $\mathcal{Q}^{(1)}$ condition, as $\mathcal{Q}_{l,m}$ satisfies the $\mathcal{Q}^{(2)}$ condition. This implies that there exists at least one *honest* party in $\mathcal{R}_{l,m}$ who correctly shares $a^{(l)} \cdot b^{(m)}$. However, since the exact identity of the honest parties in $\mathcal{R}_{l,m}$ is not known, the parties check if *all* the parties in $\mathcal{R}_{l,m}$ shared the same value by computing their differences and publicly checking if the differences are all 0. The idea here is that if *all* the parties in $\mathcal{R}_{l,m}$ share the *same* value in their respective instances of Sh, then any of these sharings can be taken as a $[\cdot]$-sharing of $a^{(l)} \cdot b^{(m)}$. However, if any of the parties in $\mathcal{R}_{l,m}$ shares an incorrect $a^{(l)} \cdot b^{(m)}$, then it will be detected, in which case the parties publicly reconstruct both $a^{(l)}$ as well as $b^{(m)}$ and compute a default $[\cdot]$-sharing of $a^{(l)} \cdot b^{(m)}$. Notice that in the latter case, the privacy of a and b is still preserved, as the shares $a^{(l)}$ and $b^{(m)}$ are already known to the adversary.

Protocol $\mathsf{Mult}([a], [b])$

Let $[a] = (a^{(1)}, \ldots, a^{(q)})$ and $[b] = (b^{(1)}, \ldots, b^{(q)})$, with parties in $\mathcal{G}_m$ holding the shares $a^{(m)}$ and $b^{(m)}$ respectively, for $m = 1, \ldots, q$.

- For every ordered pair $(l, m) \in \{1, \ldots, q\} \times \{1, \ldots, q\}$, the parties do the following to compute $[a^{(l)} \cdot b^{(m)}]$.
  - Let $\mathcal{Q}_{l,m} \subset \mathcal{P}$ be the set of parties who own both the shares $a^{(l)}$ and $b^{(m)}$. That is, $\mathcal{Q}_{l,m} \stackrel{\text{def}}{=} \mathcal{G}_l \cap \mathcal{G}_m$. Each party $P_i \in \mathcal{Q}_{l,m}$ acts as a dealer and $[\cdot]$-shares $a^{(l)} \cdot b^{(m)}$ by invoking an instance $\mathsf{Sh}(P_i, a^{(l)} b^{(m)})$ of Sh.
  - The parties participate in the Sh instances invoked by various parties in $\mathcal{Q}_{l,m}$.
  - The parties execute an instance $\mathsf{ACS}(\mathcal{Q}_{l,m})$ of ACS to agree on a subset of parties $\mathcal{R}_{l,m} \subseteq \mathcal{Q}_{l,m}$, where $\mathcal{Q}_{l,m} \setminus \mathcal{R}_{l,m} \subseteq Z$ for some $Z \in \mathcal{Z}$, such that the

Sh instances of all the parties in $\mathcal{R}_{l,m}$ eventually terminate for all (honest) parties.

- Let $r \stackrel{\text{def}}{=} |\mathcal{R}_{l,m}|$ and let $\mathcal{R}_{l,m} = \{P_{\alpha_1}, \ldots, P_{\alpha_r}\}$, where v_i is the value shared by the party $P_{\alpha_i} \in \mathcal{R}_{l,m}$ in its Sh instance[a].
- The parties check whether $v_1, \ldots, v_r$ are all equal. For this, the parties locally compute the $r-1$ differences $[d_1] = [v_1]-[v_2], \ldots, [d_{r-1}] = [v_1]-[v_r]$. This is followed by publicly reconstructing the differences $d_1, \ldots, d_{r-1}$ by invoking instances $\mathsf{Rec}([d_1]), \ldots, \mathsf{Rec}([d_{r-1}])$ of Rec and checking if all of them are 0.
  - If $d_1, \ldots, d_{r-1}$ are all 0, then the parties set $[a^{(l)}b^{(m)}] = [v_1]$.
  - Else, the parties reconstruct $a^{(l)}, b^{(m)}$ by invoking instances of Rec. The parties then set $[a^{(l)} \cdot b^{(m)}]$ to a default $[\cdot]$-sharing of $a^{(l)} \cdot b^{(m)}$.

– The parties set $[a \cdot b] = \sum_{(l,m) \in \{1,\ldots,q\} \times \{1,\ldots,q\}} [a^{(l)} \cdot b^{(m)}]$ and terminate.

[a] If P_{α_i} is *honest*, then $v_i = a^{(l)} \cdot b^{(m)}$ holds.

Fig. 5: The Perfectly-Secure Multiplication Protocol.

The properties of protocol Mult are stated in the following theorem.

Theorem 6.1. *Let a and b be $[\cdot]$-shared among a set of parties $\mathcal{P}$, satisfying the $\mathcal{Q}^{(4)}$ condition. Then in Mult, the honest parties eventually terminate with a $[\cdot]$-sharing of $a \cdot b$. Moreover, the view of Adv in the protocol is independent of a and b. The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}|^3 \cdot \mathsf{poly}(n))$ elements from $\mathcal{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(|\mathcal{Z}|^3 \cdot \mathsf{poly}(n))$ bits. In addition, $|\mathcal{Z}|^2$ instances of ACS are required.*

Multiplying Multiple Shared Values Simultaneously: Let $([a_1], [b_1]), \ldots, ([a_\ell], [b_\ell])$ be ℓ pairs of $[\cdot]$-shared values and the goal be to securely compute a $[\cdot]$-sharing of the product $[c_i]$, where $c_i = a_i \cdot b_i$, for $i = 1, \ldots, \ell$. A straightforward way of doing this is to invoke ℓ independent instances of protocol Mult, where the i^{th} instance is used for computing $[a_i \cdot b_i]$ from $[a_i]$ and $[b_i]$. This will require a number of ACS instances which is proportional to ℓ, namely $\ell \cdot |\mathcal{Z}|^2$ instances of ACS will be involved. Instead, the number of ACS instances can be made independent of ℓ. For this, the parties execute ℓ instances of the protocol Mult. Let these instances be denoted as $\mathsf{Mult}_1, \ldots, \mathsf{Mult}_\ell$. Then, for each of these ℓ instances, for the ordered pair (l, m), the parties execute a *single* instance of ACS instead of ℓ instances, to identify a *single* set $\mathcal{R}_{l,m}$ for all the ℓ instances.

More explicitly, the set $\mathcal{Q}_{l,m}$ will be the *same* for $\mathsf{Mult}_1, \ldots, \mathsf{Mult}_\ell$ and each party in $\mathcal{Q}_{l,m}$ executes ℓ instances of Sh to $[\cdot]$-share the values $a_1^{(l)} \cdot b_1^{(m)}, \ldots, a_\ell^{(l)} \cdot b_\ell^{(m)}$ respectively. Here, $a_i^{(l)}$ and $b_i^{(m)}$ denote the l^{th} and m^{th} shares of a_i and b_i respectively. Next, the parties execute a *single* instance of ACS to identify a subset $\mathcal{R}_{l,m}$ of $\mathcal{Q}_{l,m}$, such that the ℓ instances of Sh of all the parties in $\mathcal{R}_{l,m}$ eventually terminate for all honest parties, where $\mathcal{Q}_{l,m} \setminus \mathcal{R}_{l,m} \subseteq Z$, for some $Z \in \mathcal{Z}$. The rest of the steps of Mult in the instances $\mathsf{Mult}_1, \ldots, \mathsf{Mult}_\ell$ are carried out as previously mentioned. With this, the number of ACS instances remains $|\mathcal{Z}|^2$ (one instance for each (l, m) across all the ℓ instances of Mult).

We call the modified protocol $\mathsf{Mult}(\{[a_i],[b_i]\}_{i\in\{1,\ldots,\ell\}})$. The protocol incurs a communication of $\mathcal{O}(\ell\cdot|\mathcal{Z}|^3\cdot\mathsf{poly}(n))$ elements from $\mathcal{K}$ over the pair-wise channels and a broadcast of $\mathcal{O}(\ell\cdot|\mathcal{Z}|^3\cdot\mathsf{poly}(n))$ bits, apart from $|\mathcal{Z}|^2$ instances of ACS.

6.2 Preprocessing Phase Protocol with Perfect Security

The preprocessing phase protocol PP is presented in Fig 6. The protocol outputs M number of $[\cdot]$-shared random multiplication triples over $\mathcal{K}$, such that the view of Adv is independent of these multiplication triples. The protocol consists of two stages. During the first stage, the parties generate M pairs of $[\cdot]$-shared random values and during the second stage, a $[\cdot]$-sharing of the product of each pair is computed. For the first stage, each party shares M pairs of random values, after which the parties identify a sufficiently large number of parties $\mathcal{R}$ whose sharing instances terminate, such that it is ensured that at least one party in $\mathcal{R}$ is *honest*. Since the honest parties in $\mathcal{R}$ share random values, summing up the pairs of values shared by *all* the parties in $\mathcal{R}$ results in random pairs.

Protocol PP

- **Generating random pairs of values**: The parties execute the following steps to generate a $[\cdot]$-sharing of M pairs of random values.
  - Each party $P_i\in\mathcal{P}$ picks M pairs of random values $(a_i^{(1)},b_i^{(1)}),\ldots,(a_i^{(M)},b_i^{(M)})$ over $\mathcal{K}$ and $[\cdot]$-shares these values by executing $2M$ instances of Sh.
  - The parties execute an instance $\mathsf{ACS}(\mathcal{P})$ of ACS to identify a set of parties $\mathcal{R}\subseteq\mathcal{P}$ where $\mathcal{P}\setminus\mathcal{R}\subseteq Z$ for some $Z\in\mathcal{Z}$, such that the $2M$ instances of Sh of all the parties in $\mathcal{R}$ eventually terminate for all the parties.
  - For $k=1,\ldots,M$, the parties set $[a_k]=\sum_{P_i\in\mathcal{R}}[a_i^{(k)}]$ and $[b_k]=\sum_{P_i\in\mathcal{R}}[b_i^{(k)}]$.
- **Computing the product of the pairs:** The parties compute $[c_1],\ldots,[c_M]$ by executing $\mathsf{Mult}(\{[a_k],[b_k]\}_{k\in\{1,\ldots,M\}})$ and terminate.

Fig. 6: The Preprocessing Phase Protocol.

The properties of protocol PP are stated in Lemma 6.2.

Lemma 6.2. *Let the set of parties $\mathcal{P}$ satisfy the $\mathcal{Q}^{(4)}$ condition. Then in the protocol* PP*, the following holds.*

- **Termination***: All honest parties eventually terminate the protocol.*
- **Correctness***: The parties output M number of $[\cdot]$-shared random triples.*
- **Privacy***: The view of* Adv *is independent of the output multiplication triples.*
- **Communication Complexity***: The protocol incurs a communication of $\mathcal{O}(M\cdot|\mathcal{Z}|^3\cdot\mathsf{poly}(n))$ elements from $\mathcal{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(M\cdot|\mathcal{Z}|^3\cdot\mathsf{poly}(n))$ bits, along with $|\mathcal{Z}|^2$ instances of* ACS.

7 The AMPC Protocol

Once we have a preprocessing phase protocol, constructing an AMPC protocol is straightforward. The protocol consists of three phases. The first phase is the preprocessing phase, where the parties generate $[\cdot]$-sharings of M random multiplication triples by executing the protocol PP. The second phase is the input phase, where each party $[\cdot]$-shares its input by executing an instance of Sh. Due to the asynchronous nature of communication, the parties cannot afford to wait for the termination of Sh instances of all the parties and hence, they execute an instance of ACS to agree on a common subset of input providers $\mathcal{C}$ whose sharing instances eventually terminate, such that the remaining set of parties $\mathcal{P} \setminus \mathcal{C}$ is a part of some set in $\mathcal{Z}$. For these missing parties, a default $[\cdot]$-sharing of 0 is taken as their inputs. The third phase is the circuit-evaluation phase, where the parties jointly evaluate each gate in the circuit by maintaining the invariant that given the gate inputs, the parties compute the corresponding gate output in a $[\cdot]$-shared fashion. Maintaining the invariant is non-interactive for the addition gates, owing to the linearity property of $[\cdot]$-sharing. For the multiplication gates, the parties deploy the standard Beaver's circuit-randomization method. Finally, once the circuit output is available in a $[\cdot]$-shared fashion, the parties publicly reconstruct it by executing an instance of Rec. Since each honest party eventually invokes this instance of Rec, all honest parties eventually terminate the protocol. As the protocol is standard, we defer the formal details to the full version of the paper.

Theorem 7.1. *Let $f : \mathcal{K}^n \rightarrow \mathcal{K}$ be a publicly known function, expressed as an arithmetic circuit over $\mathcal{K}$ (which could be a ring or a field), consisting of M number of multiplication gates. Moreover, let Adv be a computationally unbounded adversary, characterized by an adversary structure $\mathcal{Z}$, such that the set of parties $\mathcal{P}$ satisfies the $\mathcal{Q}^{(4)}$ condition. Then, there exists a perfectly-secure AMPC protocol tolerating Adv. The protocol incurs a communication of $\mathcal{O}(M \cdot |\mathcal{Z}|^3 \cdot \mathsf{poly}(n))$ elements from $\mathcal{K}$ over the point-to-point channels and a broadcast of $\mathcal{O}(M \cdot |\mathcal{Z}|^3 \cdot \mathsf{poly}(n))$ bits, along with $|\mathcal{Z}|^2 + 1$ instances of ACS.*

8 Open Problems

Our work leaves several open problems. The first is to improve the communication complexity of our protocol, both in terms of the dependency on $|\mathcal{Z}|$, as well as in terms of the involved $\mathsf{poly}(n)$ factor. In this work, we considered perfect security. One could also explore statistical and computational security in the asynchronous communication model, tolerating a generalized adversary.

Acknowledgement: We sincerely thank the anonymous reviewers of INDOCRYPT 2020 for several helpful remarks and comments.

References

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

2. L. Bangalore, A. Choudhury, and A. Patra. The Power of Shunning: Efficient Asynchronous Byzantine Agreement Revisited. *J. ACM*, 67(3):14:1–14:59, 2020.
3. D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991.
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 *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230. Springer Verlag, 2008.
6. M. Ben-Or. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *PODC*, pages 27–30. ACM, 1983.
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. G. Bracha. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC*, pages 154–162. ACM, 1984.
11. R. Canetti. *Studies in Secure Multiparty Computation and Applications.* PhD thesis, Weizmann Institute, Israel, 1995.
12. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
13. R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC*, pages 42–51, 1993.
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. A. Choudhury. Brief Announcement: Almost-surely Terminating Asynchronous Byzantine Agreement Protocols with a Constant Expected Running Time. In *PODC*, pages 169–171. ACM, 2020.
16. A. Choudhury, M. Hirt, and A. Patra. Asynchronous Multiparty Computation with Linear Communication Complexity. In Y. Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2013.

17. Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous mpc with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, ICDCN '15, New York, NY, USA, 2015. Association for Computing Machinery.
18. S. Coretti, J. A. Garay, M. Hirt, and V. Zikas. Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions. In *ASIACRYPT*, volume 10032 of *Lecture Notes in Computer Science*, pages 998–1021, 2016.
19. 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.
20. R. Cramer, I. Damgård, and U. M. Maurer. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In B. Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer Verlag, 2000.
21. V. Dani, V. King, M. Movahedi, and J. Saia. Quorums Quicken Queries: Efficient Asynchronous Secure Multiparty Computation. In *ICDCN*, LNCS 8314, pages 242–256. Springer Verlag, 2014.
22. M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.
23. Matthias Fitzi, Martin Hirt, and Ueli Maurer. General adversaries in unconditional multi-party computation. In Kwok Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *Advances in Cryptology — ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 232–246. Springer-Verlag, November 1999.
24. O. Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
25. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In A. V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.
26. V. Goyal, Y. Liu, and Y. Song. Communication-Efficient Unconditional MPC with Guaranteed Output Delivery. In *CRYPTO*, volume 11693 of *Lecture Notes in Computer Science*, pages 85–114. Springer, 2019.
27. 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.
28. Martin Hirt. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. PhD thesis, ETH Zurich, September 2001. Reprint as vol. 3 of *ETH Series in Information Security and Cryptography*, ISBN 3-89649-747-2, Hartung-Gorre Verlag, Konstanz, 2001.
29. Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '97, page 25–34, New York, NY, USA, 1997. Association for Computing Machinery.

30. Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, April 2000. Extended abstract in *Proc. 16th of ACM PODC '97*.
31. Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multiparty computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 473–485, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
32. Martin Hirt and Daniel Tschudi. Efficient general-adversary multi-party computation. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2013.
33. M. Karchmer and A. Wigderson. On span programs. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 102–111, 1993.
34. M. V. N. Ashwin Kumar, K. Srinathan, and C. Pandu Rangan. Asynchronous perfectly secure computation tolerating generalized adversaries. In Lynn Batten and Jennifer Seberry, editors, *Information Security and Privacy*, pages 497–511, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.
35. K. Kursawe and F. C. Freiling. Byzantine Fault Tolerance on General Hybrid Adversary Structures. Technical Report, RWTH Aachen, 2005.
36. Joshua Lampkins and Rafail Ostrovsky. Communication-efficient MPC for general adversary structures. *IACR Cryptol. ePrint Arch.*, 2013:640, 2013.
37. N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.
38. Ueli M. Maurer. Secure multi-party computation made simple. In Stelvio Cimato, Clemente Galdi, and Giuseppe 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 14–28. Springer, 2002.
39. A. Patra, A. Choudhary, T. Rabin, and C. Pandu Rangan. The Round Complexity of Verifiable Secret Sharing Revisited. In S. Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 487–504. Springer, 2009.
40. A. Patra, A. Choudhury, and C. Pandu Rangan. Asynchronous Byzantine Agreement with Optimal Resilience. *Distributed Computing*, 27(2):111–146, 2014.
41. Michael O. Rabin. Randomized Byzantine Generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409, 1983.
42. 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.
43. A. Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.
44. Adam Smith and Anton Stiglic. Multiparty computation unconditionally secure against Q^2 adversary structures. *CoRR*, cs.CR/9902010, 1999.
45. 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.
46. A. C. Yao. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.