Network Agnostic Perfectly Secure MPC Against General Adversaries

Ananya Appan[1]
University of Illinois at Urbana Champaign, USA

Anirudh Chandramouli[1]
Bar-Ilan University, Ramat Gan, Israel

Ashish Choudhury
International Institute of Information Technology, Bangalore, India

Abstract

In this work, we study *perfectly-secure multi-party computation* (MPC) against general (*non-threshold*) adversaries. Known protocols are secure against $\mathcal{Q}^{(3)}$ and $\mathcal{Q}^{(4)}$ adversary structures in a synchronous and an asynchronous network respectively. We address the existence of a *single* protocol which remains secure against $\mathcal{Q}^{(3)}$ and $\mathcal{Q}^{(4)}$ adversary structures in a *synchronous* and in an *asynchronous* network respectively, where the parties are *unaware* of the network type. We design the *first* such protocol against general adversaries. Our result generalizes the result of Appan, Chandramouli and Choudhury (PODC 2022), which presents such a protocol against *threshold* adversaries.

2012 ACM Subject Classification Security and privacy → Information-theoretic techniques; Theory of computation → Distributed algorithms; Theory of computation → Cryptographic protocols; Theory of computation → Communication complexity

Keywords and phrases Verifiable Secret Sharing, Byzantine Agreement, Perfect Security

Digital Object Identifier 10.4230/LIPIcs.DISC.2023.3

Related Version *Full Version*: https://arxiv.org/abs/2208.06223

Funding *Anirudh Chandramouli*: Supported by the Israel Science Foundation (grant No. 2439/20).
Ashish Choudhury: This research is an outcome of the R&D work undertaken in the project under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology, Government of India, being implemented by Digital India Corporation (formerly Media Lab Asia). The author is also thankful to the Electronics, IT & BT Government of Karnataka for supporting this work under the CIET project.

1 Introduction

Secure *multi-party computation* (MPC) [40, 27, 10] is one of the central pillars in modern cryptography. Informally, an MPC protocol allows a set of mutually distrusting parties, $\mathcal{P} = \{P_1, \ldots, P_n\}$, to securely perform any computation over their private inputs *without* revealing anything additional about their inputs. In any MPC protocol, the distrust is modelled by a centralized *adversary* $\mathcal{A}$, who can corrupt and control a subset of the parties during the protocol execution. We aim for *perfect security*, where $\mathcal{A}$ is a *computationally unbounded* byzantine adversary who can force the corrupt parties to behave *arbitrarily* during protocol execution and where all security guarantees are achieved *without* any error.

[1] Work done as a student at IIIT Bangalore

Traditionally, the *corruption capacity* of $\mathcal{A}$ is modelled through a publicly-known *threshold* t, where it is assumed that $\mathcal{A}$ can corrupt *any* subset of up to t parties [10, 15, 38]. A more generic and fine-grained form of corruption capacity is the *general-adversary* model (also known as the *non-threshold* setting) [28]. Here, $\mathcal{A}$ is characterized by a publicly-known monotone *adversary structure* $\mathcal{Z} \subset 2^{\mathcal{P}}$, which enumerates *all possible* subsets of potentially corrupt parties, where $\mathcal{A}$ can select any one subset from $\mathcal{Z}$ for corruption. Notice that a *threshold* adversary is a *special* type of *non-threshold* adversary, where $\mathcal{Z}$ consists of all subsets of $\mathcal{P}$ of size up to t. It is well-known that modelling $\mathcal{A}$ through $\mathcal{Z}$ allows for more flexibility, especially when $\mathcal{P}$ is small [28, 29]. The downside is that the complexity of the resultant protocols is polynomial in the size of $\mathcal{Z}$, which could be exponential in n in the worst case.

Traditionally, MPC protocols are designed assuming either a *synchronous* or *asynchronous* communication model. In a *synchronous* MPC (SMPC) protocol, the communication channels between the parties are assumed to be *synchronized*, and every message is assumed to be delivered within some *known* time Δ. Unfortunately, maintaining such time-outs in real-world networks like the Internet is extremely challenging. Asynchronous MPC (AMPC) protocols operate assuming an *asynchronous* communication network with eventual message delivery, where the messages can be arbitrarily, yet finitely delayed. Designing AMPC protocols is inherently *more challenging* when compared to SMPC protocols. This is because, due to the lack of an upper bound on message delays, parties won't know how long to wait for an expected message, since the corresponding sender party may be *corrupt* and may not send the message in the first place. Consequently, to avoid an endless wait, a party can consider messages from only a subset of parties for processing but, in the process, messages from potentially slow but honest parties may get ignored. In fact, in *any* AMPC protocol, it is *impossible* to ensure that the inputs of *all* honest parties are considered for computation, since the wait may turn out to be endless.

Against *threshold* adversaries, perfectly-secure SMPC and AMPC can tolerate up to $t_s < n/3$ [10] and $t_a < n/4$ [9] corrupt parties respectively. Following the notion of [29], given an adversary structure $\mathcal{Z}$ and a subset of parties $\mathcal{P}' \subseteq \mathcal{P}$, we say that $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(k)}(\mathcal{P}', \mathcal{Z})$ condition if the union of any k subsets from $\mathcal{Z}$ *does not* cover $\mathcal{P}'$. That is, for *every* $Z_{i_1}, Z_{i_2}, \ldots, Z_{i_k} \in \mathcal{Z}$, the following holds:

$$\mathcal{P}' \not\subseteq Z_{i_1} \cup \ldots \cup Z_{i_k}.$$

SMPC and AMPC against general adversaries is possible, provided the underlying adversary structure $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ [29] and $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition [32] respectively.

Our Motivation and Results. In an MPC protocol, it is usually *assumed* that the parties will be knowing if the underlying network is synchronous or asynchronous *beforehand.* Suppose that the parties are *not aware* of the network type. We aim to design a *single* MPC protocol that is capable of adapting to the exact timing behaviour of the underlying network while offering the best possible security guarantees in either network. We call such a protocol a *best-of-both-worlds* (BoBW) or a network-agnostic MPC protocol. Recently, [2] presented a BoBW *perfectly-secure* MPC protocol against *threshold* adversaries which could tolerate up to t_s and t_a corruptions in a *synchronous* and *asynchronous* network respectively, for any $t_a < t_s$ where $t_a < n/4$ and $t_s < n/3$, provided $3t_s + t_a < n$ holds. We aim to generalize this result against *general* adversaries, and ask the following question:

Let $\mathcal{A}$ be an adversary characterized by adversary structures $\mathcal{Z}_s$ and $\mathcal{Z}_a$ in a synchronous network and asynchronous network respectively, where $\mathcal{Z}_s \neq \mathcal{Z}_a$. Then, is there a BoBW perfectly-secure MPC protocol which is secure against $\mathcal{A}$, irrespective of the network type?

No prior work has addressed the above question. We present a BoBW perfectly-secure MPC protocol provided *all* the following conditions hold, which we refer to throughout as Con.[2]

▶ **Condition 1 ($\mathsf{Con}(\mathcal{Z}_s, \mathcal{Z}_a)$).** *$\mathcal{Z}_s$ and $\mathcal{Z}_a$ satisfy the following conditions.*

- *$\mathcal{Z}_s \neq \mathcal{Z}_a$, and $\mathcal{Z}_s, \mathcal{Z}_a$ satisfy the $\mathcal{Q}^{(3,1)}(\mathcal{P}, \mathcal{Z}_s, \mathcal{Z}_a)$ condition, meaning that the union of any 3 subsets from $\mathcal{Z}_s$ and any one subset from $\mathcal{Z}_a$, does not cover $\mathcal{P}$.*
- *Every subset in $\mathcal{Z}_a$ is a subset of some subset in $\mathcal{Z}_s$.*

The computation and communication complexity of our protocol is polynomial in n and $|\mathcal{Z}_s|$.

Significance of Our Result. We focus on the case where $\mathcal{Z}_s \neq \mathcal{Z}_a$ as, otherwise, the question is *trivial* to solve [3]. Let $\mathcal{P} = \{P_1, \ldots, P_8\}$, $\mathcal{Z}_s = \{\{P_1, P_2, P_3\}, \{P_2, P_3, P_4\}, \{P_3, P_4, P_5\}, \{P_4, P_5, P_6\}, \{P_7\}, \{P_8\}\}$ and $\mathcal{Z}_a = \{\{P_1, P_3\}, \{P_2, P_4\}, \{P_3, P_5\}, \{P_4, P_6\}\}$. Since $\mathcal{Z}_s$ and $\mathcal{Z}_a$ satisfy $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z}_s)$ and $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_a)$ conditions respectively, it follows that *existing* SMPC and AMPC protocols can tolerate $\mathcal{Z}_s$ and $\mathcal{Z}_a$ respectively. However, we show that *even* if the parties are *not aware* of the exact network type, then using our protocol, one can *still* achieve security against $\mathcal{Z}_s$ if the network is *synchronous* or against $\mathcal{Z}_a$ if the network is *asynchronous*. The above example demonstrates the flexibility offered by the non-threshold adversary model, in terms of tolerating *more* faults. In the *threshold* model, using the protocol of [2], one can tolerate up to $t_s = 2$ and $t_a = 1$ faults, in a *synchronous* and *asynchronous* network respectively. In the *non-threshold* model, our protocol can tolerate subsets of size larger than the maximum t_s and t_a allowed in a synchronous and asynchronous network.

We compare the communication complexity of our network-agnostic MPC protocol with the most efficient existing synchronous and asynchronous MPC protocols in Table 1.[4] Here, $(\mathbb{K}, +, \cdot)$ denotes the finite ring (or field) over which the computations are performed.

Table 1 Amortized communication complexity per multiplication of different perfectly-secure MPC protocols against general adversaries.

Setting	Reference	Condition	Communication Complexity (in bits)
Synchronous	[30]	$\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$	$\mathcal{O}(\|\mathcal{Z}\|^2 \cdot (n^5 \log \|\mathbb{K}\| + n^6) + \|\mathcal{Z}\| \cdot (n^7 \log \|\mathbb{K}\| + n^8))$
Asynchronous	[3]	$\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$	$\mathcal{O}(\|\mathcal{Z}\|^2 \cdot n^7 \log \|\mathbb{K}\| + \|\mathcal{Z}\| \cdot n^9 \log n)$
Network Agnostic	This work	$\mathsf{Con}(\mathcal{Z}_s, \mathcal{Z}_a)$	$\mathcal{O}(\|\mathcal{Z}_s\|^2 \cdot n^5 (\log \|\mathbb{K}\| + \log \|\mathcal{Z}_s\| + \log n))$

1.1 Technical Overview

Like in any generic MPC protocol [27, 10, 38], we assume that the underlying computation (which the parties want to perform securely) is modelled as some publicly-known function f, abstracted by some arithmetic circuit cir, over some algebraic structure $\mathbb{K}$, consisting of linear and non-linear (multiplication) gates. The problem of secure computation then reduces to secure *circuit-evaluation*, where the parties jointly and securely "evaluate" cir in a *secret-shared* fashion, such that all the values during the circuit-evaluation remain *verifiably secret-shared* and where the shares of the corrupt parties *fail* to reveal the exact

[2] Conditions Con imply that $\mathcal{Z}_s$ and $\mathcal{Z}_a$ satisfy the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z}_s)$ and $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_a)$ conditions respectively.
[3] If $\mathcal{Z}_s = \mathcal{Z}_a$, then AMPC is possible only if *even* $\mathcal{Z}_s$ satisfies the $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_s)$ condition. *Any* existing perfectly-secure AMPC protocol (with appropriate time-outs) [32, 18, 3] will work *even* in the synchronous network, with the guarantee that the inputs of *all honest* parties are considered for the computation
[4] Conventionally, the communication complexity of any generic MPC protocol is measured in terms of the number of bits communicated to evaluate a single multiplication gate in the underlying circuit.

underlying value. The secret-sharing used is typically *linear* [20], thus allowing the parties to evaluate the linear gates *locally* (non-interactively). On the other hand, non-linear gates are evaluated by deploying the standard Beaver's method [8] using random, secret-shared *multiplication-triples* which are generated in a circuit-independent *preprocessing phase*. Then, once all the gates are securely evaluated, the parties publicly reconstruct the secret-shared circuit-output. Apart from *verifiable secret-sharing* (VSS) [16], the parties also need to run instances of a *Byzantine agreement* (BA) protocol [37] to ensure that all the parties are on the "same page" during the various stages of the circuit-evaluation. The above framework for shared circuit-evaluation is defacto used in *all* generic perfectly-secure SMPC and AMPC protocols. Unfortunately, there are several obstacles while adapting the framework if the parties are *unaware* of the network type.

First Obstacle – A BoBW BA Protocol. Informally, a BA protocol [37] allows parties with private inputs to reach an agreement on a *common* output, even if a subset of the parties behave maliciously. In the *non-threshold* setting, one can design perfectly-secure BA protocol against $\mathcal{Q}^{(3)}$ adversary structures *irrespective* of the network type [25, 17]. However, the *termination* (also called *liveness*) guarantees are *different* for *synchronous* BA (SBA) and *asynchronous* BA (ABA) protocols. The (deterministic) SBA protocols ensure that all honest parties obtain their output after some fixed time (*guaranteed liveness*) [37]. On the other hand, to circumvent the FLP impossibility result [24], ABA protocols are *randomized* and provide *almost-surely liveness* [1, 7, 17], where the parties terminate the protocol asymptotically with a probability of 1. Known SBA protocols become insecure in an *asynchronous* network even if one expected message from an honest party gets arbitrarily delayed, while existing ABA protocols can provide *only* almost-surely liveness in a *synchronous* network.

The *first* obstacle is to get a BoBW BA protocol against non-threshold adversaries, which provides the security guarantees of SBA and ABA in a *synchronous* and an *asynchronous* network respectively. We present such a BA protocol which is secure against $\mathcal{Q}^{(3)}$ adversary structures. The protocol is obtained by generalizing the BoBW BA protocol of [2] which is secure against *threshold* adversaries and tolerates $t < n/3$ faults.

Second Obstacle – A BoBW VSS Protocol. In a VSS protocol, a designated *dealer* $\mathsf{D} \in \mathcal{P}$ has some private input s. The goal is to let D "verifiably" distribute shares of s such that the adversary does not learn anything additional about s, if D is *honest* (*privacy*). In a *synchronous* VSS (SVSS), every (honest) party obtains its shares after some *known* time-out (*correctness*). *Verifiability* guarantees that even a *corrupt* D shares some value "consistently" within the known time-out (*commitment* property). Perfectly-secure SVSS is possible, provided the underlying adversary structure $\mathcal{Z}_s$ satisfies $\mathcal{Q}^{(3)}$ condition [34, 30].

For an *asynchronous* VSS (AVSS) protocol, *correctness* guarantees that for an *honest* D, the secret s is eventually secret-shared. However, a *corrupt* D *may not* invoke the protocol in the first place, in which case the honest parties may not obtain any shares. Hence, the *commitment* property of AVSS guarantees that if D is *corrupt and* if some honest party computes a share (implying that D has invoked the protocol), then all honest parties eventually compute their shares. Perfectly-secure AVSS is possible, provided the underlying adversary structure $\mathcal{Z}_a$ satisfies the $\mathcal{Q}^{(4)}$ condition [18, 3].

Existing SVSS protocols become insecure in an asynchronous network, even if a single expected message from an *honest* party is *delayed*. On the other hand, existing AVSS protocols are insecure against $\mathcal{Q}^{(3)}$ adversary structures (which SVSS protocols can tolerate). Since, in our setting, the parties will *not* be knowing the exact network type, to maintain *privacy* during the shared circuit-evaluation, we need to ensure that each value remains secret-shared with respect to $\mathcal{Z}_s$ rather than *not* $\mathcal{Z}_a$, *even* if the network is *asynchronous*.[5] The *second* obstacle to perform shared circuit-evaluation in our setting is to get a perfectly-secure VSS protocol which is secure with respect to $\mathcal{Z}_s$ and $\mathcal{Z}_a$ in a *synchronous* and *asynchronous* network respectively and where *privacy always holds* with respect to $\mathcal{Z}_s$, *irrespective* of the network type. We are not aware of any VSS protocol with these guarantees. Hence, we present a BoBW VSS protocol satisfying the required properties.

Our BoBW VSS protocol is obtained by carefully and non-trivially "stitching" together the SVSS and AVSS protocols of [34] and [18] respectively. Both these protocols are further based on the classic *additive* secret-sharing protocol of Ito et al [31] (designed against passive adversaries). The secret is shared using a *sharing specification* $\mathbb{S}_{\mathcal{Z}}$ corresponding to a given adversary structure $\mathcal{Z}$, where $\mathbb{S}_{\mathcal{Z}}$ is the collection of "set-complements" of the subsets in $\mathcal{Z}$. That is, if $\mathcal{Z} = \{Z_1, \ldots, Z_{|\mathcal{Z}|}\}$, then $\mathbb{S}_{\mathcal{Z}} = (S_1, \ldots, S_{|\mathcal{Z}|})$ where $S_m = \mathcal{P} \setminus Z_m$, for $m = 1, \ldots, |\mathcal{Z}|$. The idea behind the secret-sharing of [31] is then to share a secret s through a *random* vector of shares $(s_1, \ldots, s_{|\mathcal{Z}|})$ which sum up to s, where all (honest) parties in the group S_m hold the share s_m. Since one of the subsets in $\mathbb{S}_{\mathcal{Z}}$ consists of *only honest* parties, it would be ensured that if D is *honest*, then the probability distribution of the shares learnt by the adversary is *independent* of s. The SVSS and AVSS protocols of [34] and [18] ensure that the underlying secret is indeed shared as per the above semantics, even in the presence of *malicious* corruptions, including a potentially corrupt D. We next briefly discuss these protocols individually and then give a high-level overview of how we combine them.

- **SVSS Against $\mathcal{Q}^{(3)}$ Adversary Structures [34]:** Consider an arbitrary adversary structure $\mathcal{Z}_s$ satisfying the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z}_s)$ condition, and let $\mathbb{S}_{\mathcal{Z}_s} = (S_1, \ldots, S_{|\mathcal{Z}_s|})$ be the corresponding sharing specification. The protocol is executed as a sequence of *phases*. To share s, during the *first* phase, D picks a random vector of shares $(s_1, \ldots, s_{|\mathcal{Z}_s|})$, such that $s = s_1 + \ldots + s_{|\mathcal{Z}_s|}$. Then all parties in every group $S_m \in \mathbb{S}_{\mathcal{Z}_s}$ are given share s_m by D. To check whether a potentially *corrupt* D has given the same share to all the (honest) parties in S_m, the parties in S_m perform a *pairwise consistency* check of their supposedly common share during the *second* phase, and publicly broadcast the results during the *third* phase, using a *synchronous* reliable broadcast protocol. If any party in S_m publicly complains about an inconsistency, then during the *fourth* phase, D makes public the share s_m corresponding to S_m by broadcasting it. This *does not* violate the privacy for an *honest* D, since a complaint for inconsistency from S_m implies that S_m has at least one *corrupt* party and so, the adversary will already know s_m. If D *does not* "resolve" any complaint during the fourth phase (implying D is *corrupt*), then D is *publicly discarded*, and everyone takes a default sharing of 0 on the behalf of D. Clearly, the protocol ensures that by the end of the *fourth* phase, *all honest* parties in S_m have the *same* share, and the sum of these shares across all the S_m sets is the value shared by D.
- **AVSS Against $\mathcal{Q}^{(4)}$ Adversary Structures [18]:** Consider an arbitrary adversary structure $\mathcal{Z}_a$ satisfying the $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_a)$ condition, and let $\mathbb{S}_{\mathcal{Z}_a} = (S_1, \ldots, S_{|\mathcal{Z}_a|})$ be the corresponding sharing specification. The AVSS protocol of [18] closely follows the SVSS protocol of [34]. However, the phases are *no longer* synchronized. Moreover, during the pairwise consistency phase, the parties *cannot* afford to wait to know the status of the consistency checks between all pairs of parties, since potentially *corrupt* parties may *never* respond. Instead, corresponding to every S_m, the parties check for the existence of a set

[5] Since we are assuming that every subset in $\mathcal{Z}_a$ is a subset of some subset in $\mathcal{Z}_s$, privacy will be maintained *irrespective* of the network type if each value remains secret-shared with respect to $\mathcal{Z}_s$.

of "core" parties $\mathcal{C}_m \subseteq S_m$, with $S_m \setminus \mathcal{C}_m \in \mathcal{Z}_a$, which publicly confirmed that they are pairwise consistent. To ensure that all the parties agree on the core sets, D is assigned the task of identifying the core sets and broadcasting them (where the broadcast now happens through an *asynchronous* reliable broadcast protocol). The protocol proceeds *only* upon the receipt of core sets from D and their verification. While an *honest* D will eventually find and broadcast valid core sets, a *corrupt* D may *not* do so, in which case the parties obtain no shares. Once the core sets are identified and verified, it is guaranteed that all the (honest) parties in each core set $\mathcal{C}_m$ have received the same share from D. The goal is then to ensure that even the (honest) parties "outside" $\mathcal{C}_m$ (namely, the parties in $S_m \setminus \mathcal{C}_m$) get this common share. Since $\mathcal{Z}_a$ satisfies the $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_a)$ condition, the "majority" of the parties in $\mathcal{C}_m$ are *honest* [6]. Hence, the parties in $S_m \setminus \mathcal{C}_m$ can "extract" the common share held by the parties in $\mathcal{C}_m$, by applying the "majority rule" on the shares received from the parties in $\mathcal{C}_m$, during the pairwise consistency tests.

- **Our BoBW VSS Protocol:** In our VSS protocol, the parties first start executing the steps of the above SVSS protocol, *assuming* a *synchronous* network, where all the instances of broadcast happen by executing an instance of a BoBW reliable broadcast protocol Π_{BC}, designed as part of our BoBW BA protocol. Let T_{BC} be the time taken by the protocol Π_{BC} to produce the output in a *synchronous* network. If indeed the network is *synchronous*, then within time $2\Delta + T_{\mathsf{BC}}$, the results of pairwise consistency tests should be publicly available, where Δ is the upper bound on message delay in a *synchronous* network. Moreover, if any inconsistency is reported, then within the time $2\Delta + 2T_{\mathsf{BC}}$, the dealer D should have resolved all those inconsistencies by making the "disputed" shares public. However, unlike the SVSS protocol, the parties *cannot* afford to discard D if it fails to resolve any inconsistency within time $2\Delta + 2T_{\mathsf{BC}}$. This is because the network could be *asynchronous*, and D's responses may be arbitrarily *delayed*, even if D is *honest*. A bigger challenge is that in an *asynchronous* network, some honest parties, say $\mathcal{H}_1$, *might* be seeing the inconsistencies being reported within local time $2\Delta + T_{\mathsf{BC}}$, *as well as* D's responses within the local time $2\Delta + 2T_{\mathsf{BC}}$. And there might be another set of honest parties, say $\mathcal{H}_2$, who *might not* be seeing these inconsistencies and D's responses within these timeouts. This may result in the parties in $\mathcal{H}_1$ considering the shares made public by D, while the parties in $\mathcal{H}_2$ may think that the network is *asynchronous* and wait for the core sets of parties to be made public by D (as done in the AVSS). However, this gives a *corrupt* D an opportunity to *violate* the *commitment* property in an *asynchronous* network. In more detail, consider a set S_m for which pairwise *inconsistency* is reported, and for which D also finds a set of core parties $\mathcal{C}_m$. Then, it might be possible that the parties in $\mathcal{C}_m$ have received the common share s_m from D, but in response to the inconsistencies reported for S_m, D broadcasts the share s'_m, where $s'_m \neq s_m$. This will lead to a situation where the parties in $\mathcal{H}_1$ consider s'_m as the share for the group S_m after the timeout of $2\Delta + 2T_{\mathsf{BC}}$. On the other hand, the parties in $\mathcal{H}_2$ *may not* see the inconsistencies and s'_m within the timeout of $2\Delta + 2T_{\mathsf{BC}}$, but eventually see $\mathcal{C}_m$ and extract the share s_m corresponding to S_m.
  To deal with the above challenge, apart from resolving the inconsistencies reported for *any* set S_m, the dealer D *also* finds and broadcasts a core set of parties $\mathcal{C}_m$, who have confirmed receiving the same share from D corresponding to *all* the sets S_m, such that

[6] Since the $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z}_a)$ condition is satisfied, the conditions $\mathcal{Q}^{(3)}(S_m, \mathcal{Z}_a)$ and, consequently, $\mathcal{Q}^{(2)}(\mathcal{C}_m, \mathcal{Z}_a)$ are also satisfied. Thus, the $\mathcal{Q}^{(1)}(\mathcal{C}_m \setminus Z^\star, \mathcal{Z}_a)$ condition is satisfied, where $Z^\star$ is the actual set of corrupt parties, implying that the set of honest parties form a "majority".

$S_m \setminus \mathcal{C}_m \in \mathcal{Z}_s$. *Additionally*, if there is any inconsistency reported for S_m, then *apart* from D, *every* party in S_m *also* makes public its version of the share corresponding to S_m received from D. Now, at time $2\Delta + 2T_{\mathsf{BC}}$, the parties check if D has broadcasted a core set $\mathcal{C}_m$ for each S_m. Moreover, if any inconsistency has been reported corresponding to S_m, the parties check if "sufficiently many" parties from $\mathcal{C}_m$ have made public the same share which D made public. This *prevents* a *corrupt* D from making public a share that is *different* from the share which it distributed to the parties in $\mathcal{C}_m$.

If the network is *asynchronous*, then different parties may have *different* "opinions" regarding whether D has broadcasted "valid" core sets $\mathcal{C}_m$. Hence, at time $2\Delta + 2T_{\mathsf{BC}}$, the parties run an instance of our BoBW BA protocol to decide what the case is. If the parties find that D has broadcasted valid core sets $\mathcal{C}_m$ corresponding to each S_m, then the parties in S_m proceed to compute their share as follows: if D has made public the share for S_m in response to any inconsistency, then it is taken as the share for S_m. If no share has been made public for S_m, then the parties check if "sufficiently many" parties have reported the same share during the pairwise consistency test within time 2Δ, which we show should have happened if the network is *synchronous*, and if the parties maintain sufficient timeouts. If none of these conditions holds, then the parties proceed to filter out the common share, held by the parties in $\mathcal{C}_m$, through the "majority rule".

On the other hand, if the parties find that D has *not* made public core sets within time $2\Delta + 2T_{\mathsf{BC}}$, then either the network is *asynchronous* or D is *corrupt*. So the parties resort to the steps used in AVSS. Namely, D finds and broadcasts a set of core parties $\mathcal{E}_m$ corresponding to each S_m, where $S_m \setminus \mathcal{E}_m \in \mathcal{Z}_a$. [7] Then, the parties filter out the common share, held by the parties in $\mathcal{E}_m$, through majority rule (see Section 4 for details).

Best-of-Both-Worlds Secure Multiplication. Apart from BoBW VSS and BA, another key component in our MPC protocol is a BoBW multiplication protocol against general adversaries. This is again obtained by carefully stitching together the synchronous and asynchronous multiplication protocol of [34] and [18] respectively. The protocol takes as input secret-shared a and b, both shared with respect to $\mathcal{Z}_s$, and securely outputs a secret-sharing of $a \cdot b$ with respect to $\mathcal{Z}_s$, *irrespective* of the network type. Let $(a_1, \ldots, a_{|\mathcal{Z}_s|})$ and $(b_1, \ldots, b_{|\mathcal{Z}_s|})$ be the vector of shares, corresponding to a and b respectively. The idea here is to securely generate a secret-sharing of each of the summands $a_l \cdot b_m$, where $l, m \in \{1, \ldots, |\mathcal{Z}_s|\}$. The linearity property (see Definition 2) of the secret-sharing then guarantees that a secret-sharing of $a \cdot b$ can be obtained from the secret-sharing of the summands $a_l \cdot b_m$.

To generate a secret-sharing of $a_l \cdot b_m$, the parties do the following: let $\mathcal{I}_{l,m}$ be the set of parties who have both a_l and b_m. Since $\mathcal{Q}^{(3,1)}(\mathcal{P}, \mathcal{Z}_s, \mathcal{Z}_a)$ condition is satisfied, *irrespective* of the network type, $\mathcal{I}_{l,m}$ *will* have *at least* one honest party. Each party in $\mathcal{I}_{l,m}$ is asked to independently secret-share $a_l \cdot b_m$ through an instance of our BoBW VSS protocol. To avoid an endless wait, the parties *cannot* afford for *all* the parties in $\mathcal{I}_{l,m}$ to secret-share their "versions" of $a_l \cdot b_m$, even if the network would have been *synchronous*. Hence the parties run instances of our BoBW BA to agree on a common subset of parties $\mathcal{R}_{l,m}$ from $\mathcal{I}_{l,m}$, where $\mathcal{I}_{l,m} \setminus \mathcal{R}_{l,m} \in \mathcal{Z}_s$, who have shared some version of $a_l \cdot b_m$ through VSS instances. However, we take special care to ensure that *irrespective* of the network type, the set $\mathcal{R}_{l,m}$ has at least one *honest* party from $\mathcal{I}_{l,m}$, who has indeed shared the summand $a_l \cdot b_m$. Note that achieving this goal is *not* a challenge for the *synchronous* multiplication protocol of [34], since

[7] $\mathcal{E}_m$ (not to be confused with $\mathcal{C}_m$) is the core set of parties corresponding to S_m which D finds in case it is unable to find and make public valid core sets $\mathcal{C}_m$ "on time" for each S_m.

$\mathcal{R}_{l,m} = \mathcal{I}_{l,m}$ holds.[8] Similarly, the goal is *easily* achievable in the *asynchronous* multiplication protocol of [18].[9] To ensure that the $\mathcal{R}_{l,m}$ has at least one *honest* party, we carefully run instances of our BoBW BA and decide the timeouts of the parties in these BA instances (see Section 5 for the exact details). Once the set $\mathcal{R}_{l,m}$ is decided, the parties then check if *all* the parties in $\mathcal{R}_{l,m}$ have shared the same version of $a_l \cdot b_m$. If all the versions are the same, then any one of these is taken as a secret-sharing of $a_l \cdot b_m$. Else at least one party from $\mathcal{R}_{l,m}$ has behaved maliciously and so the parties publicly reconstruct the shares a_l and b_m and compute a default secret-sharing of $a_l \cdot b_m$.[10]

Comparison of Our Results with [2]. Even though our BoBW BA protocol is an easy generalization of the BoBW BA protocol of [2] against *threshold* adversaries, our VSS protocol and the multiplication protocol are relatively simpler and based on completely different ideas. For instance, the BoBW VSS protocol of [2] is based on the properties of symmetric bivariate polynomials of degree t_s in two variables over a finite field, where the underlying secret is embedded in the constant term of the polynomial and the share for each party is a distinct univariate polynomial, lying on the bivariate polynomial (this is a two-dimensional extension of the classical Shamir's secret-sharing [39]). The bivariate polynomials help to verify whether a potentially *corrupt* D has distributed shares consistently. However, verifying the same in the BoBW setting is quite challenging. As a result, the VSS protocol of [2] is quite involved and is further based on a "weaker" primitive, called *weak polynomial-sharing* (WPS) [36, 5], which ensures that if the dealer is *corrupt*, then *only* a subset of the *honest* parties receive their designated shares.[11] On the contrary, our BoBW VSS protocol is much *simpler* and *not* based on any WPS protocol. Intuitively this is because the "sharing-semantics" of the underlying secret-sharing is *different* for VSS against the threshold and non-threshold adversaries. While the former is based on polynomial interpolation, the latter deploys additive secret-sharing. Consequently, there is more "redundancy" available to verify whether D has consistently shared its secret, compared to bivariate polynomials, since each candidate share is now available with multiple parties. To the best of our knowledge, the idea of designing VSS based on WPS has been used *only* against *threshold* adversaries and it is *not* known whether the idea can be generalized against *non-threshold* adversaries.

Similarly, the multiplication protocol of [2] is quite involved and based on the framework of [19], which further involves a lot of subprotocols and deploys properties of polynomial evaluation and interpolation over finite fields. In contrast, our multiplication protocol is relatively simpler and straightforward and *does not* involve multiple sub-protocols.

1.2 Other Related Work

All existing works in the domain of BoBW protocols focus only on *threshold* adversaries. The works of [12, 14, 21] show that the condition $2t_s + t_a < n$ is necessary and sufficient for BoBW *cryptographically-secure* BA and MPC, tolerating *computationally bounded* adversaries. Using

---

[8] In a synchronous network, a and b are secret-shared with respect to a set $\mathcal{Z}$ satisfying $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. This ensures that $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(1)}(\mathcal{I}_{l,m}, \mathcal{Z})$ condition and hence contains at least one honest party. Moreover, in a *synchronous* network, the VSS instances of *all* the parties in $\mathcal{I}_{l,m}$ get over within a known time bound and hence $\mathcal{R}_{l,m} = \mathcal{I}_{l,m}$ holds.

[9] In an asynchronous network, a and b are secret-shared with respect to a set $\mathcal{Z}$ satisfying $\mathcal{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. This ensures that $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(2)}(\mathcal{I}_{l,m}, \mathcal{Z})$ condition. Consequently, $\mathcal{I}_{l,m} \setminus \mathcal{R}_{l.m} \in \mathcal{Z}$ will hold, implying that $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(1)}(\mathcal{R}_{l,m}, \mathcal{Z})$ condition and $\mathcal{R}_{l,m}$ contains at least one honest party.

[10] The vector of shares $(s, 0, \ldots, 0)$ can be considered as a default sharing of a publicly known value s.

[11] It is *not* known how to *directly* design a BoBW VSS protocol, *without* deploying any WPS.

the same condition, [13] presents a BoBW *cryptographically-secure* atomic broadcast protocol. The work of [35] studies Byzantine fault tolerance and state machine replication protocols for multiple thresholds, including t_s and t_a. The work of [26] presents a BoBW protocol for the task of approximate agreement using the condition $2t_s + t_a < n$. The same condition has been used to design a BoBW distributed key-generation (DKG) protocol in [6]. A recent work [22] has studied the problem of perfectly-secure message transmission (PSMT) [23] over *incomplete* graphs, in the BoBW setting. Along with the results of [2], they note that BoBW perfectly-secure MPC over incomplete networks is possible as long as $3t_s + t_a < n$ *and* $t_s + 2t_a < N$, where N is the connectivity of the graph modelling the underlying network.

1.3 Open Problems

We do not know whether the conditions Con are indeed necessary for any BoBW perfectly-secure MPC protocol. In fact, it is not known whether the corresponding condition $3t_s+t_a < n$ is necessary for any BoBW perfectly-secure MPC against *threshold* adversaries. We conjecture that these conditions are indeed necessary for the respective adversarial model, for any BoBW perfectly-secure MPC. The main aim of this work (and [2]) is to show the feasibility of BoBW perfectly-secure MPC against general adversaries over complete networks. We do not know if an equivalent result for MPC over incomplete networks can be shown as in [22]. Improving the efficiency of these protocols is also left for future work.

2 Preliminaries and Definitions

The parties in $\mathcal{P}$ are assumed to be connected by pair-wise secure channels. The underlying communication network can be either synchronous or asynchronous, with parties being *unaware* about the exact type. In a *synchronous* network, every sent message is delivered within a *known* time Δ. In an *asynchronous* network, messages can be delayed arbitrarily, but finitely, with every message sent being delivered *eventually*. The distrust among $\mathcal{P}$ is modelled by a *malicious* (byzantine) adversary $\mathcal{A}$, who can corrupt a subset of the parties in $\mathcal{P}$ and force them to behave in any arbitrary fashion during the execution of a protocol. For simplicity, we assume the adversary to be *static*, it decides the set of corrupt parties at the beginning of the protocol execution. However, our protocols can be proved secure even against a more powerful *adaptive* adversary that can decide the set of corrupt parties at run time.

Adversary $\mathcal{A}$ can corrupt any one subset of parties from $\mathcal{Z}_s$ and $\mathcal{Z}_a$ in *synchronous* and *asynchronous* networks respectively. The adversary structures are *monotone*, implying that if $Z \in \mathcal{Z}_s$ ($Z \in \mathcal{Z}_a$ resp.), then every subset of Z also belongs to $\mathcal{Z}_s$ (resp. $\mathcal{Z}_a$). We say that $\mathcal{Z}_s$ and $\mathcal{Z}_a$ satisfy the $\mathcal{Q}^{(k,k')}(\mathcal{P}, \mathcal{Z}_s, \mathcal{Z}_a)$ condition if the union of any k subsets from $\mathcal{Z}_s$ and any k' subsets from $\mathcal{Z}_a$, *does not* cover $\mathcal{P}$. That is, for every $Z_{i_1}, \ldots, Z_{i_k} \in \mathcal{Z}_s$ and every $Z_{j_1}, \ldots, Z_{j_{k'}} \in \mathcal{Z}_a$, the condition $\mathcal{P} \not\subseteq Z_{i_1} \cup \ldots \cup Z_{i_k} \cup Z_{j_1} \cup \ldots \cup Z_{j_{k'}}$ holds.

In our VSS and MPC protocols, all computations are done over a finite algebraic structure $(\mathbb{K}, +, \cdot)$, which could be a ring or a field. Without loss of generality, we assume that each P_i has an input $x_i \in \mathbb{K}$, and the parties want to securely compute a function $f : \mathbb{K}^n \to \mathbb{K}$, represented by an arithmetic circuit cir over $\mathbb{K}$, consisting of linear and non-linear (multiplication) gates, where cir has c_M multiplication gates and a multiplicative depth of D_M.

Termination Guarantees of Our Sub-Protocols. As done in [2], for simplicity, we will *not* be specifying any *termination* criteria for our sub-protocols. The parties will keep on participating in these sub-protocol instances even *after* computing their outputs. The termination criteria of our MPC protocol will ensure the termination of *all* underlying sub-protocol instances. We will be using an existing *randomized* ABA protocol [17] which ensures that the honest parties (eventually) obtain their respective output *almost-surely* with probability 1, where the probability is over the random coins of the honest parties and adversary in the protocol. The property of almost-surely obtaining an output carries over to the "higher" level protocols, where ABA is used as a building block.

We next discuss the syntax and semantics of the secret-sharing used in our VSS.

▶ **Definition 2** ([34]). *Let $\mathbb{S} = (S_1, \ldots, S_{|\mathbb{S}|})$ be a set called the sharing specification where, for $m = 1, \ldots, |\mathbb{S}|$, each $S_m \subseteq \mathcal{P}$. Then a value $s \in \mathbb{K}$ is said to be secret-shared with respect to $\mathbb{S}$ if there exist shares $s_1, \ldots, s_{|\mathbb{S}|} \in \mathbb{K}$ such that $s = s_1 + \ldots + s_{|\mathbb{S}|}$ and, for $m = 1, \ldots, |\mathbb{S}|$, the share s_m is available to every (honest) party in S_m.*

A secret-sharing of s is denoted by $[s]$, where $[s]_m$ denotes the m^{th} share. The above secret-sharing is *linear* as $[c_1 s_1 + c_2 s_2] = c_1[s_1] + c_2[s_2]$ holds for publicly-known $c_1, c_2 \in \mathbb{K}$. Hence, the parties can *non-interactively* compute any linear function over secret-shared inputs. For our protocols, we will consider the sharing specification $\mathbb{S} = \{S_m : S_m = \mathcal{P} \setminus Z_m \text{ and } Z_m \in \mathcal{Z}_s\}$.

2.1 Existing Asynchronous Primitives

Asynchronous Reliable Broadcast (Acast). An Acast protocol allows a designated *sender* $\mathsf{S} \in \mathcal{P}$ to send its input $m \in \{0,1\}^\ell$ *identically* to all the parties, even if S is potentially corrupt. An Acast protocol Π_{ACast} is presented in [33], provided $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The protocol also provides certain guarantees in a *synchronous* network, as stated in Lemma 8 (Appendix A). The protocol, along with the proof of Lemma 8 and various terminologies associated with Π_{ACast} are available in the full version of this paper [4].

Public Reconstruction of a Secret-Shared Value. Let $s \in \mathbb{K}$ be a value, which is secret-shared with respect to $\mathbb{S} = \{S_m : S_m = \mathcal{P} \setminus Z_m \text{ and } Z_m \in \mathcal{Z}_s\}$. To publicly reconstruct s, we use the reconstruction protocol $\Pi_{\mathsf{Rec}}(s, \mathbb{S})$ of [34]. In a *synchronous* network, the protocol will take Δ time, while in an *asynchronous* network, the parties eventually output s. The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}_s| \cdot n^2 \log |\mathbb{K}|)$ bits; see [4] for the details.

3 Best-of-Both-Worlds Byzantine Agreement (BA)

We begin with the definition of BA, which is adapted from [14, 2].

▶ **Definition 3** (BA). *Let Π be a protocol for $\mathcal{P}$ where every P_i has input $b_i \in \{0,1\}$ and a possible output from $\{0,1,\perp\}$. Let $\mathcal{A}$ be an adversary, characterized by adversary structure $\mathcal{Z}$, where $\mathcal{A}$ can corrupt any set of parties from $\mathcal{Z}$ during the execution of Π.*

- **$\mathcal{Z}$-Guaranteed Liveness:** *All honest parties obtain an output.*
- **$\mathcal{Z}$-Almost-Surely Liveness:** *Almost-surely, all honest parties obtain some output.*
- **$\mathcal{Z}$-Validity:** *If all honest parties input b, every honest party with an output outputs b.*
- **$\mathcal{Z}$-Weak Validity:** *If all honest parties input b, every honest party with an output outputs b or $\perp$.*
- **$\mathcal{Z}$-Consistency:** *All honest parties with an output output the same value (may be $\perp$).*
- **$\mathcal{Z}$-Weak Consistency:** *All honest parties with an output output a common $v \in \{0,1,\perp\}$.*

A $\mathcal{Z}$-perfectly-secure synchronous BA (SBA) protocol Π *has $\mathcal{Z}$-guaranteed liveness, $\mathcal{Z}$-validity, and $\mathcal{Z}$-consistency in a synchronous network. A $\mathcal{Z}$-perfectly-secure asynchronous BA (ABA)* Π *has $\mathcal{Z}$-almost-surely liveness, $\mathcal{Z}$-validity and $\mathcal{Z}$-consistency in an asynchronous network.*[12]

To design our BoBW BA protocol, we will need a special broadcast protocol. Hence, we next review the definition of broadcast, adapted from [14, 2].

▶ **Definition 4 (Broadcast).** *Let Π be a protocol where a sender $\mathsf{S} \in \mathcal{P}$ has input $m \in \{0,1\}^\ell$, and parties obtain an output. Let $\mathcal{A}$ be an adversary characterized by adversary structure $\mathcal{Z}$.*

- **$\mathcal{Z}$-Liveness:** *All honest parties obtain some output.*
- **$\mathcal{Z}$-Validity:** *If S is honest, then every honest party with an output outputs m.*
- **$\mathcal{Z}$-Weak Validity:** *If S is honest, every honest party with an output outputs m or $\perp$.*
- **$\mathcal{Z}$-Consistency:** *If S is corrupt, every honest party with an output outputs a common value.*
- **$\mathcal{Z}$-Weak Consistency:** *If S is corrupt, every honest party with an output outputs a common $m^\star \in \{0,1\}^\ell$ or $\perp$.*

Π *is a $\mathcal{Z}$-perfectly-secure broadcast protocol if it has $\mathcal{Z}$-Liveness, $\mathcal{Z}$-Validity, and $\mathcal{Z}$-Consistency.*[13]

We give an overview of how to generalize the BoBW BA protocol of [2] and defer to the full version of the paper [4] for the details. The protocol is based on three components.

Component I: SBA with Asynchronous Guaranteed Liveness. We require a $\mathcal{Z}$-perfectly-secure SBA protocol Π_{SBA} with $\mathcal{Q}^{(3)}(\mathcal{P},\mathcal{Z})$ condition, which *also* provides $\mathcal{Z}$-guaranteed liveness in an *asynchronous* network. We design a candidate for Π_{SBA} by generalizing the simple SBA protocol of [11], which was designed to tolerate $t < n/3$ corruptions. The protocol requires at most $3n$ rounds in a *synchronous* network and hence, within time $T_{\mathsf{SBA}} \stackrel{def}{=} 3n \cdot \Delta$, all honest parties will get an output in a *synchronous* network. The protocol incurs a communication of $\mathcal{O}(n^3\ell)$ bits if the inputs of the parties are of size ℓ bits. To achieve $\mathcal{Z}$-guaranteed liveness in an *asynchronous* network, the parties can run Π_{SBA} till time T_{SBA}, and then output $\perp$ if no "valid" output is computed as per the protocol at the time T_{SBA}; see the full version of this paper [4] for the details.

Component II: ABA with Synchronous Guarantees. We deploy the ABA protocol Π_{ABA} of [17], where $\mathcal{Z}$ satisfies the $\mathcal{Q}^{(3)}(\mathcal{P},\mathcal{Z})$ condition and where each party has an input bit. The protocol has the following liveness guarantees in an *asynchronous* network.

- If the inputs of all *honest* parties are the same, then Π_{ABA} achieves $\mathcal{Z}$-guaranteed liveness. Else, Π_{ABA} achieves $\mathcal{Z}$-almost-surely liveness.

Protocol Π_{ABA} also achieves $\mathcal{Z}$-validity, $\mathcal{Z}$-consistency, and the following liveness guarantees in a *synchronous* network.

- If all *honest* parties have the same input, then Π_{ABA} achieves $\mathcal{Z}$-guaranteed liveness, and all honest parties obtain output within time $T_{\mathsf{ABA}} = k \cdot \Delta$, for some known constant k.
- Else, Π_{ABA} achieves $\mathcal{Z}$-almost-surely liveness and requires $\mathcal{O}(\mathrm{poly}(n) \cdot \Delta)$ expected time.

[12] The *weak validity* and *weak consistency* properties are defined here for the sake of completeness. Looking ahead, our BoBW BA protocol will be using BA protocol(s) with these "weaker" properties.

[13] Similar to BA, the weak validity and consistency properties are defined here for the sake of completeness, since we will be designing a broadcast protocol with these weaker properties in our BoBW BA protocol.

Irrespective of the network type, Π_{ABA} incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^5 \log |\mathbb{F}| + n^6 \log n)$ bits, if all honest parties have the same input bit. Else, it incurs an expected communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^7 \log |\mathbb{F}| + n^8 \log n)$ bits. Here $\mathbb{F}$ is a finite field such that $|\mathbb{F}| > n$ holds.

Component III: Synchronous Broadcast with Asynchronous Guarantees. We assume the existence of a broadcast protocol Π_{BC}, which is a $\mathcal{Z}$-perfectly-secure broadcast protocol in a *synchronous* network, and which also provides $\mathcal{Z}$-Liveness, $\mathcal{Z}$-Weak Validity and $\mathcal{Z}$-Weak Consistency in an *asynchronous* network. We present a candidate for Π_{BC} by generalizing the broadcast protocol of [2] with similar guarantees. The protocol incurs a communication of $\mathcal{O}(n^3\ell)$ bits, where S participates with input $m \in \{0,1\}^\ell$. The idea is to carefully "stitch" together protocol Π_{ACast} with the protocol Π_{SBA}. In the protocol, all *honest* parties have some output at the (local) time $T_{\mathsf{BC}} = 3\Delta + T_{\mathsf{SBA}}$. Depending upon the network type and corruption status of S, the output is -

- *Synchronous Network and Honest* S*:* m for *all* honest parties.
- *Synchronous Network and Corrupt* S*:* a common $m^\star \in \{0,1\}^\ell \cup \{\perp\}$ for *all* honest parties.
- *Asynchronous Network and Honest* S*:* either m or $\perp$ for each honest party.
- *Asynchronous Network and Corrupt* S*:* a common $m^\star \in \{0,1\}^\ell$ or $\perp$ for each honest party.

Protocol Π_{BC} also gives the parties who output $\perp$ at time T_{BC} an option to switch their output to some ℓ-bit string if the parties keep running the protocol beyond time T_{BC} and if certain "conditions" are satisfied for those parties. We stress that this switching provision is *only* for those who output $\perp$ at time T_{BC}. While this provision is not "useful" and not used while designing BA, it comes in handy when Π_{BC} is used to broadcast values in our VSS protocol. Notice that the output-switching provision will *not* lead to a violation of consistency and hence honest parties will *not* end up with different ℓ-bit outputs. Following the terminology of [2], we call the process of computing output at time T_{BC} and beyond time T_{BC} as the *regular mode* and *fallback mode* of Π_{BC} respectively. We refer to Appendix A for the terminologies associated with the protocol Π_{BC}.

Π_{BC}+Π_{ABA} $\Rightarrow$ BoBW BA. We combine protocols Π_{BC} and Π_{ABA} to get Π_{BA} by generalizing the idea used in [2] against *threshold* adversaries. In the protocol, every party first broadcasts its input bit (for the BA protocol) through an instance of Π_{BC}. If the network is *synchronous*, then all honest parties should have received the inputs of all the (honest) sender parties from the corresponding broadcast instances through regular mode by time T_{BC}. Consequently, at time T_{BC}, the parties decide an output for *all* the n instances of Π_{BC}. Based on these outputs, the parties decide their respective inputs for the Π_{ABA} protocol. Specifically, if "sufficiently many" outputs from the Π_{BC} instances are found to be *same*, then the parties consider this output value as their input for the Π_{ABA} instance. Else, they stick to their original inputs. The overall output for Π_{BA} is then set to be the output from Π_{ABA}. For the formal description of Π_{BA} and the proof of Theorem 5, see [4].

▶ **Theorem 5.** *Let $\mathcal{Z}$ satisfy the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then Π_{BA} achieves the following.*

- *In a synchronous network, the protocol is a $\mathcal{Z}$-perfectly-secure SBA protocol, where all honest parties obtain an output within time $T_{\mathsf{BA}} = T_{\mathsf{BC}} + T_{\mathsf{ABA}}$. The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^5 \log |\mathbb{F}| + n^6 \log n)$ bits.*
- *In an asynchronous network, the protocol is a $\mathcal{Z}$-perfectly-secure ABA protocol, with an expected communication of $\mathcal{O}(|\mathcal{Z}| \cdot n^7 \log |\mathbb{F}| + n^8 \log n)$ bits.*

Protocol $\Pi_{\mathsf{VSS}}(\mathsf{D}, s, \mathbb{S} = (S_1, \ldots, S_{|\mathcal{Z}_s|}))$

- **Phase I – Share Distribution:** D randomly selects $s^{(1)}, \ldots, s^{(|\mathcal{Z}_s|)} \in \mathbb{K}$ such that $s = s^{(1)} + \ldots + s^{(|\mathcal{Z}_s|)}$. For $m = 1, \ldots, |\mathcal{Z}_s|$, it then sends $s^{(m)}$ to every party in the set S_m.
- **Phase II – Pairwise Checks:** For $m = 1, \ldots, |\mathcal{Z}_s|$, each $P_i \in S_m$ does the following.
  - On receiving $s_i^{(m)}$ from D, wait till the local time is a multiple of Δ. Send $s_i^{(m)}$ to each $P_j \in S_m$.
  - On receiving $s_j^{(m)}$ from any $P_j \in S_m$, wait till the local time is a multiple of Δ. Do the following.
    - If a share $s_i^{(m)}$ corresponding to S_m has been received from D, then, broadcast $\texttt{OK}(m, i, j)$ if $s_i^{(m)} = s_j^{(m)}$ holds. Else, broadcast $\texttt{NOK}(m, i)$.
    - If $s_j^{(m)}$ and $s_k^{(m)}$ have been received from any P_j and P_k respectively, belonging to S_m such that $s_j^{(m)} \neq s_k^{(m)}$, then broadcast $\texttt{NOK}(m, i)$.
- **Local Computation – Constructing Consistency Graphs:** Each $P_i \in \mathcal{P}$ constructs undirected *consistency graphs* $G_i^{(1)}, \ldots, G_i^{(|\mathcal{Z}_s|)}$, where $G_i^{(m)}$ is over the parties in S_m and where the edge (P_j, P_k) is included in $G_i^{(m)}$ if P_i has received $\texttt{OK}(m, j, k)$ and $\texttt{OK}(m, k, j)$ from the broadcast of P_j and P_k respectively, either through regular or fallback mode.
- **Phase III – Resolving Complaints and Broadcasting Core Sets Based On $\mathcal{Z}_s$:** Each $P_i \in \mathcal{P}$ (including D) does the following at time $2\Delta + T_{\mathsf{BC}}$.
  - If $\texttt{NOK}(m, j)$ is received from the broadcast of any $P_j \in S_m$ through regular-mode corresponding to any $m \in \{1, \ldots, |\mathcal{Z}_s|\}$, then do the following:
    - **If** $P_i = \mathsf{D}$**:** Broadcast $\texttt{Resolve}(m, s^{(m)})$.
    - **If** $P_i \neq \mathsf{D}$**:** Broadcast $\texttt{Resolve}(m, s_i^{(m)})$, if $P_i \in S_m$ and P_i has received $s_i^{(m)}$ from D.
  - **(If** $P_i = \mathsf{D}$**):** For $m = 1, \ldots, |\mathcal{Z}_s|$, check if there exists a $\mathcal{C}_m \subseteq S_m$ which constitutes a clique in graph $G_{\mathsf{D}}^{(m)}$, such that $S_m \setminus \mathcal{C}_m \in \mathcal{Z}_s$. If $\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{Z}_s|}$ are found, then broadcast them.
- **Local Computation – Verifying and Accepting Core sets:** Each party $P_i \in \mathcal{P}$ (including D) does the following at time $2\Delta + 2T_{\mathsf{BC}}$.
  - If $\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{Z}_s|}$ are received from the broadcast of D through regular mode, *accept* these if:
    - For $m = 1, \ldots, |\mathcal{Z}_s|$, the set $\mathcal{C}_m$ constitutes a clique in the consistency graph $G_i^{(m)}$ at time $2\Delta + T_{\mathsf{BC}}$. In addition, $S_m \setminus \mathcal{C}_m \in \mathcal{Z}_s$.
    - For $m = 1, \ldots, |\mathcal{Z}_s|$, if $\texttt{NOK}(m, j)$ was received from the broadcast of any $P_j \in S_m$ through regular mode at time $2\Delta + T_{\mathsf{BC}}$, then the following must hold at time $2\Delta + 2T_{\mathsf{BC}}$.
      - $\texttt{Resolve}(m, s^{(m)})$ is received from the broadcast of D through regular-mode.
      - $\texttt{Resolve}(m, s^{(m)})$ is received from the broadcast of a set of parties $\mathcal{C}'_m$ through regular-mode, where $\mathcal{C}'_m \subseteq \mathcal{C}_m$, and $\mathcal{C}_m \setminus \mathcal{C}'_m \in \mathcal{Z}_s$.
- **Phase IV – Deciding Whether Core Sets Based on $\mathcal{Z}_s$ have Been Accepted by Any Honest Party:** At time $2\Delta + 2T_{\mathsf{BC}}$, each $P_i \in \mathcal{P}$ participates in an instance of Π_{BA} with input $b_i = 1$ if it has accepted sets $\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{Z}_s|}$, else, with input $b_i = 0$, and waits for time T_{BA}.

Figure 1 Best-of-both-worlds VSS protocol: Part I.

4 Best-of-Both-Worlds VSS Protocol

The goal of our BoBW VSS protocol (Fig 1 and Fig 2) is to enable a *dealer* $\mathsf{D} \in \mathcal{P}$ to "verifiably" generate a secret-sharing of its private input $s \in \mathbb{K}$ with respect to the specification $\mathbb{S} = \{S_m : S_m = \mathcal{P} \setminus Z_m \text{ and } Z_m \in \mathcal{Z}_s\}$, *irrespective* of the network type. An overview of the protocol has been given in Section 1. In the protocol, broadcast is instantiated through Π_{BC} with respect to $\mathcal{Z}_s$ (see the terminologies associated with Π_{BC} in Appendix A).

Theorem 6 states the properties of Π_{VSS} and is proven in the full version of the paper [4].

▶ **Theorem 6.** *Protocol* Π_{VSS} *achieves the following.*

Protocol $\Pi_{\mathsf{VSS}}(\mathsf{D}, s, \mathbb{S} = (S_1, \ldots, S_{|\mathcal{Z}_s|}))$ Contd ...

- **Local Computation – Computing Shares Through Core Sets Based on $\mathcal{Z}_s$:** If the output of Π_{BA} is 1, then each party $P_i \in \mathcal{P}$ does the following.
  - If $\mathcal{C}_1, \ldots, \mathcal{C}_{|\mathcal{Z}_s|}$ are not received yet, then wait to receive them from the broadcast of D. Then for $m = 1, \ldots, |\mathcal{Z}_s|$, compute the share $s_i^{(m)}$ corresponding to S_m as follows, if $P_i \in S_m$.
    - If at time $2\Delta + 2T_{\mathsf{BC}}$, `Resolve`$(m, s^{(m)})$ was received from D's broadcast and from a set of parties $\mathcal{C}'_m \subseteq \mathcal{C}_m$ through regular-mode, where $\mathcal{C}_m \setminus \mathcal{C}'_m \in \mathcal{Z}_s$, then output $s_i^{(m)} = s^{(m)}$.
    - Else, if a common value, say $s^{(m)}$, was received from a set of parties $\mathcal{C}''_m \subseteq \mathcal{C}_m$ at time 2Δ where $\mathcal{C}_m \setminus \mathcal{C}''_m \in \mathcal{Z}_s$, then output $s_i^{(m)} = s^{(m)}$.
    - Else wait till there exists a subset of parties $\mathcal{C}'''_m \subseteq \mathcal{C}_m$ where $\mathcal{C}_m \setminus \mathcal{C}'''_m \in \mathcal{Z}_a$, such that a common value, say $s^{(m)}$, is received from all the parties in $\mathcal{C}'''_m$. Output $s_i^{(m)} = s^{(m)}$.
- **Phase V – Broadcasting Core Sets Based on $\mathcal{Z}_a$:** If the output of Π_{BA} is 0, then for $m = 1, \ldots, |\mathcal{Z}_s|$, dealer D does the following in its graph $G_{\mathsf{D}}^{(m)}$.
  - Check if there exists a subset of parties $\mathcal{E}_m \subseteq S_m$, which constitutes a clique in the graph $G_{\mathsf{D}}^{(m)}$, such that $S_m \setminus \mathcal{E}_m \in \mathcal{Z}_a$. Upon finding $\mathcal{E}_1, \ldots, \mathcal{E}_{|\mathcal{Z}_s|}$, broadcast them.
- **Local Computation – Computing Shares Through Core Sets Based on $\mathcal{Z}_a$:** If the output of Π_{BA} is 0, then each party $P_i \in \mathcal{P}$ does the following.
  - Participate in any instance of Π_{BC} invoked by D for broadcasting $\mathcal{E}_1, \ldots, \mathcal{E}_{|\mathcal{Z}_s|}$, only after time $2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$. Wait till $\mathcal{E}_1, \ldots, \mathcal{E}_{|\mathcal{Z}_s|}$ are received from the broadcast of D. Upon receiving, *accept* these sets if each set $\mathcal{E}_m$ constitutes a clique in the graph $G_i^{(m)}$ and $S_m \setminus \mathcal{E}_m \in \mathcal{Z}_a$. Upon accepting, compute the share $s_i^{(m)}$ corresponding to every S_m where $P_i \in S_m$ as follows.
    - If $P_i \in \mathcal{E}_m$, then output $s_i^{(m)}$ received from D.
    - Else, wait till there exists a subset $\mathcal{E}'_m \subseteq \mathcal{E}_m$, where $\mathcal{E}_m \setminus \mathcal{E}'_m \in \mathcal{Z}_s$, such that there exists a common value, say $s^{(m)}$, received from all the parties in $\mathcal{E}'_m$. Output $s_i^{(m)} = s^{(m)}$.

Figure 2 Best-of-both-worlds VSS protocol: Part II.

If D is honest, then the following hold.

- **$\mathcal{Z}_s$-correctness:** *In a synchronous network, s is secret-shared with respect to $\mathbb{S}$ at time $T_{\mathsf{VSS}} = 2\Delta + 2T_{\mathsf{BC}} + T_{\mathsf{BA}}$.*
- **$\mathcal{Z}_a$-correctness:** *In an asynchronous network, almost-surely, s is eventually secret-shared with respect to $\mathbb{S}$.*
- **Privacy:** *Adversary's view remains independent of s in any network.*

If D is corrupt, either no honest party obtains an output or there exists an $s^\star \in \mathbb{K}$, such that:

- **$\mathcal{Z}_a$-commitment:** *In an asynchronous network, almost-surely, $s^\star$ is eventually secret-shared with respect to $\mathbb{S}$.*
- **$\mathcal{Z}_s$-commitment:** *In a synchronous network, $s^\star$ is shared with respect to $\mathbb{S}$, such that:*
  - *If any honest party outputs its shares at time T_{VSS}, then all honest parties output their shares at time T_{VSS}.*
  - *If any honest party outputs its shares at time $T > T_{\mathsf{VSS}}$, then every honest party outputs its shares by time $T + 2\Delta$.*

Communication Complexity: *The protocol incurs a communication of $\mathcal{O}(|\mathcal{Z}_s| \cdot n^4 (\log |\mathbb{K}| + \log |\mathcal{Z}_s| + \log n) + n^5 \log n)$ bits, and invokes one instance of Π_{BA}.*

Π_{VSS} for L Secrets. We describe how D can share L secrets with just one instance of Π_{BA} in Appendix B.

5 The Preprocessing Phase Protocol

Our preprocessing phase allows the parties to generate secret-sharing of c_M multiplication-triples, which are random for the adversary and is based on two sub-protocols.[14]

Agreement on a Common Subset (ACS). In protocol Π_{ACS}, there exists a set $\mathcal{P}' \subseteq \mathcal{P}$ such that it will be *guaranteed* that $\mathcal{Z}_s$ and $\mathcal{Z}_a$ *either* satisfy the $\mathcal{Q}^{(1,1)}(\mathcal{P}', \mathcal{Z}_s, \mathcal{Z}_a)$ condition *or* $\mathcal{Q}^{(3,1)}(\mathcal{P}', \mathcal{Z}_s, \mathcal{Z}_a)$ condition [15]. Moreover, each party in $\mathcal{P}'$ will have L values, which it would like to secret-share using Π_{VSS}. As *corrupt* dealers might *not* invoke their instances of Π_{VSS}, the parties can compute outputs from *only* a subset of Π_{VSS} instances corresponding to parties $\mathcal{P}' \setminus Z$, for some $Z \in \mathcal{Z}_s$ (*even* in a *synchronous* network). However, in an *asynchronous* network, *different* parties may compute outputs from Π_{VSS} instances of *different* subsets of $\mathcal{P}' \setminus Z$ parties, corresponding to a *different* $Z \in \mathcal{Z}_s$. Protocol Π_{ACS} allows parties to agree on a *common* subset $\mathcal{CS}$ of parties, where $\mathcal{P}' \setminus \mathcal{CS} \in \mathcal{Z}_s$, such that *all* honest parties will be able to compute their outputs corresponding to the Π_{VSS} instances of the parties in $\mathcal{CS}$. Moreover, in a *synchronous* network, *all honest* parties from $\mathcal{P}'$ are guaranteed to be present in $\mathcal{CS}$.[16] Protocol Π_{ACS} is obtained by generalizing the ACS protocol of [2], which was designed for *threshold* adversaries. The idea is to run n instances of our BA protocol Π_{BA}, one for each party, and decide which of these Π_{VSS} instances will produce an output for everyone. However, we need to take special care to ensure that all honest parties are going to make it to $\mathcal{CS}$ in a *synchronous* network; see the full version of the paper [4] for the details.

The Multiplication Protocol. Protocol Π_{Mult} takes as input secret-shared pairs of values $\{([a^{(\ell)}], [b^{(\ell)}])\}_{\ell=1,\ldots,L}$, and securely generates $\{[c^{(\ell)}]\}_{\ell=1,\ldots,L}$, where $c^{(\ell)} = a^{(\ell)} \cdot b^{(\ell)}$. For simplicity, we discuss the idea when $L = 1$ (a brief overview of the protocol has already been presented in Section 1). Let $[a]$ and $[b]$ be the inputs to the protocol and the goal is to compute $[a \cdot b]$. The parties securely compute secret-shared summands $[a]_l \cdot [b]_m$ and then $[a \cdot b]$ can be computed locally from secret-shared summands $[a]_l \cdot [b]_m$, owing to the linearity property. A secret-sharing of the summand $[a]_l \cdot [b]_m$ is computed as follows: let $\mathcal{I}_{l,m} = S_l \cap S_m$. Then, *irrespective* of the network type, $\mathcal{I}_{l,m}$ is *bound* to have *at least* one honest party, since $\mathcal{Z}_s$ and $\mathcal{Z}_a$ satisfy the $\mathcal{Q}^{(1,1)}(\mathcal{I}_{l,m}, \mathcal{Z}_s, \mathcal{Z}_a)$ condition. Each party in $\mathcal{I}_{l,m}$ is asked to independently secret-share the summand $[a]_l \cdot [b]_m$ through an instance of Π_{VSS}. To avoid an indefinite wait, the parties agree on a common subset of parties $\mathcal{R}_{l,m}$ from $\mathcal{I}_{l,m}$, where $\mathcal{I}_{l,m} \setminus \mathcal{R}_{l,m} \in \mathcal{Z}_s$, who have shared some summand, such that $\mathcal{R}_{l,m}$ has at least one *honest* party, *irrespective* of the network type. For this, the parties execute an instance of the Π_{ACS} protocol. To check if any cheating has occurred, the parties check whether *all* the parties in $\mathcal{R}_{l,m}$ have shared the same "version" of the summand $[a]_l \cdot [b]_m$. Protocol Π_{Mult} and its properties are available in the full version of this paper [4].

The Preprocessing Phase Protocol. Protocol $\Pi_{\mathsf{PreProcessing}}$ has two stages. In the *first* stage, the parties securely generate secret-sharing of c_M pairs of random values, by running an instance of Π_{ACS}, where the input for each party will be c_M pairs of random values. In the *second* stage, a secret-sharing of the product of each pair is computed by executing Π_{Mult}. Protocol $\Pi_{\mathsf{PreProcessing}}$ and its properties are available in the full version of this paper [4].

---

[14] $([a], [b], [c])$ constitutes a multiplication triple, where $a, b \in \mathbb{K}$ and $c = a \cdot b$ holds.

[15] In our *preprocessing phase* protocol, $\mathcal{P}'$ will be $S_l \cap S_m$ corresponding to some $S_l, S_m \in \mathbb{S}$ and hence, the $\mathcal{P}'^{(1,1)}(\mathcal{Q}, \mathcal{Z}_s, \mathcal{Z}_a)$ condition will be satisfied. In our MPC protocol, $\mathcal{P}'$ will be $\mathcal{P}$ and hence the $\mathcal{Q}^{(3,1)}(\mathcal{P}', \mathcal{Z}_s, \mathcal{Z}_a)$ condition will be satisfied.

[16] This property will be crucial in a *synchronous* network.

6 Best-of-Both-Worlds Circuit-Evaluation Protocol

Protocol Π_{CirEval} for the circuit-evaluation consists of four phases. In the *first* phase, the parties generate secret-sharing of c_M random multiplication-triples through $\Pi_{\mathsf{PreProcessing}}$. Additionally, they invoke Π_{ACS} to generate secret-sharing of their respective inputs for the publicly known function f and agree on a *common* subset of parties $\mathcal{CS}$, where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}_s$, such that the inputs of the parties in $\mathcal{CS}$ are secret-shared. The inputs of the remaining parties are set to 0. Note that in a *synchronous* network, *all honest* parties will be in $\mathcal{CS}$. In the *second* phase, the parties securely evaluate each gate in the circuit in a secret-shared fashion, after which the parties *publicly* reconstruct the secret-shared output in the *third* phase. The *last* phase is the *termination phase*, where the parties wait till "sufficiently many" parties have obtained the same output, after which they "safely" take that output and terminate the protocol (and all the underlying sub-protocols).

Π_{CirEval} and the proof of Theorem 7 are available in the full version of this paper [4].

▶ **Theorem 7.** *Let $\mathcal{A}$ be an adversary, characterized by adversary structures $\mathcal{Z}_s$ and $\mathcal{Z}_a$ in a synchronous and asynchronous network respectively, satisfying the conditions* Con *(see Condition 1 in Section 1). Moreover, let $f : \mathbb{K}^n \rightarrow \mathbb{K}$ be a function represented by an arithmetic circuit* cir *over $\mathbb{K}$, consisting of c_M number of multiplication gates, with a multiplicative depth of D_M, with each party having an input $x_i \in \mathbb{K}$. Then, Π_{CirEval} incurs a communication cost of $\mathcal{O}(c_M \cdot |\mathcal{Z}_s|^3 \cdot n^5(\log|\mathbb{K}| + \log|\mathcal{Z}_s| + \log n) + |\mathcal{Z}_s|^2 \cdot n^6 \log n)$ bits, invokes $\mathcal{O}(|\mathcal{Z}_s|^2 \cdot n)$ instances of Π_{BA}, and achieves the following for some $\mathcal{CS} \subseteq \mathcal{P}$.*

- *In a synchronous network, all honest parties output $y = f(x_1, \ldots, x_n)$ at time $(30n + D_M + 6k + 38) \cdot \Delta$, where $x_j = 0$ for every $P_j \notin \mathcal{CS}$, such that $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}_s$, and every honest party is present in $\mathcal{CS}$; here k is a constant determined by the protocol Π_{ABA}.*
- *In an asynchronous network, almost-surely, the honest parties eventually output $y = f(x_1, \ldots, x_n)$ where $x_j = 0$ for every $P_j \notin \mathcal{CS}$ and where $\mathcal{P} \setminus \mathcal{CS} \in \mathcal{Z}_s$.*
- *The view of $\mathcal{A}$ remains independent of the inputs of the honest parties in $\mathcal{CS}$.*

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

3 A. Appan, A. Chandramouli, and A. Choudhury. Revisiting the Efficiency of Asynchronous Multi Party Computation Against General Adversaries. In *INDOCRYPT*, volume 13774 of *Lecture Notes in Computer Science*, pages 223–248. Springer International Publishing, 2022.

4 Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly secure synchronous mpc with asynchronous fallback guarantees against general adversaries. Cryptology ePrint Archive, Paper 2022/1047, 2022. URL: `https://eprint.iacr.org/2022/1047`.

5 B. Applebaum, E. Kachlon, and A. Patra. The Round Complexity of Perfect MPC with Active Security and Optimal Resiliency. In *FOCS*, pages 1277–1284. IEEE, 2020.

6 R. Bacho, D. Collins, C. Liu-Zhang, and J. Loss. Network-Agnostic Security Comes for Free in DKG and MPC. Cryptology ePrint Archive, Paper 2022/1369, 2022.

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

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

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

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

11 P. Berman, J. A. Garay, and K. J. Perry. Towards Optimal Distributed Consensus (Extended Abstract). In *FOCS*, pages 410–415. IEEE Computer Society, 1989.

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

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

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

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

16 B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395. IEEE Computer Society, 1985.

17 A. Choudhury. Almost-Surely Terminating Asynchronous Byzantine Agreement Against General Adversaries with Optimal Resilience. In *ICDCN*, pages 167–176. ACM, 2023.

18 A. Choudhury and N. Pappu. Perfectly-Secure Asynchronous MPC for General Adversaries (Extended Abstract). In *INDOCRYPT*, volume 12578 of *Lecture Notes in Computer Science*, pages 786–809. Springer, 2020.

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

20 R. Cramer, I. Damgård, and U. M. Maurer. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer Verlag, 2000.

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

22 G. Deligios and C. Liu-Zhang. Synchronous Perfectly Secure Message Transmission with Optimal Asynchronous Fallback Guarantees. *IACR Cryptol. ePrint Arch.*, page 1397, 2022.

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

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

25 M. Fitzi and U. M. Maurer. Efficient Byzantine Agreement Secure Against General Adversaries. In *DISC*, volume 1499 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1998.

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

27 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.

28 Martin Hirt and Ueli Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC*, pages 25–34. ACM, 1997.

29 Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000.

30 Martin Hirt and Daniel Tschudi. Efficient general-adversary multi-party computation. In *ASIACRYPT*, volume 8270 of *Lecture Notes in Computer Science*, pages 181–200. Springer, 2013.

31 M. Ito, A. Saito, and T. Nishizeki. Secret Sharing Schemes Realizing General Access Structures). In *Global Telecommunication Conference, Globecom*, pages 99–102. IEEE Computer Society, 1987.

32 M. V. N. Ashwin Kumar, K. Srinathan, and C. Pandu Rangan. Asynchronous Perfectly Secure Computation Tolerating Generalized Adversaries. In *ACISP*, volume 2384 of *Lecture Notes in Computer Science*, pages 497–512. Springer, 2002.

33 K. Kursawe and F. C. Freiling. Byzantine Fault Tolerance on General Hybrid Adversary Structures. Technical Report, RWTH Aachen, 2005.

34 U. M. Maurer. Secure Multi-party Computation Made Simple. In *SCN*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2002.

35 A. Momose and L. Ren. Multi-Threshold Byzantine Fault Tolerance. In *CCS*, pages 1686–1699. ACM, 2021.

36 A. Patra and D. Ravi. On the Power of Hybrid Networks in Multi-Party Computation. *IEEE Trans. Information Theory*, 64(6):4207–4227, 2018.

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

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

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

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

A Broadcast Protocols

A.1 Acast

The properties satsfied by protocol Π_{ACast} [33] in a synchronous and an asynchronous network are given in Lemma 8.

▶ **Lemma 8.** *Let $\mathcal{A}$ be an adversary characterized by an adversary structure $\mathcal{Z}$ satisfying the $\mathcal{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then, for a sender S with input m, Π_{ACast} achieves the following in an asynchronous network.*

- **$\mathcal{Z}$-Liveness:** *If S is honest, then all honest parties eventually have an output.*
- **$\mathcal{Z}$-Validity:** *If S is honest, then each honest P_i with an output, outputs m.*
- **$\mathcal{Z}$-Consistency:** *If S is corrupt and some honest P_i outputs $m^\star$, then all honest parties eventually output $m^\star$.*

Π_{ACast} achieves the following in a synchronous network.

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

Communication Complexity: *$\mathcal{O}(n^2\ell)$ bits are communicated by the parties in total.*

A.2 Terminologies Associated with Π_{BC}

▶ **Terminology 9 (Terminologies for Π_{BC}).** *We say that P_i broadcasts m to mean that P_i invokes an instance of Π_{BC} as S with input m, and the parties participate in this instance. Similarly, we say that P_j receives m from the broadcast of P_i through regular-mode (resp. fallback-mode), to mean that P_j has the output m at time T_{BC} (resp. after time T_{BC}) during the instance of Π_{BC}.*

B VSS for sharing L secrets

To share L secrets, D can invoke L instances of Π_{VSS}. However, instead of computing and broadcasting $L \cdot |\mathcal{Z}_s|$ core sets, it can compute and broadcast only $|\mathcal{Z}_s|$ core sets, on the behalf of *all* the L instances of Π_{VSS}. The parties will need to execute a *single* instance of Π_{BA} to decide whether D has broadcasted valid core sets. The resultant protocol will incur a communication of $\mathcal{O}(L \cdot |\mathcal{Z}_s| \cdot n^4(\log|\mathbb{K}| + \log|\mathcal{Z}_s| + \log n) + n^5 \log n)$ bits and invokes one instance of Π_{BA}. To avoid repetition, we do not provide the formal details.