Almost-Surely Terminating Asynchronous Byzantine Agreement Against General Adversaries with Optimal Resilience

Ashish Choudhury
International Institute of Information Technology
Bangalore, India
ashish.choudhury@iiitb.ac.in

ABSTRACT

In this work, we present an almost-surely terminating *asynchronous Byzantine agreement* (ABA) protocol for n parties. Our protocol requires $O(n^2)$ expected time and is secure against a *computationally-unbounded* malicious (Byzantine) adversary, characterized by a *non-threshold* adversary structure $\mathcal{Z}$, which enumerates all possible subsets of potentially corrupt parties. Our protocol has *optimal resilience* where $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}$ condition; i.e. union of *no* three subsets from $\mathcal{Z}$ covers all the n parties. To the best of our knowledge, this is the first almost-surely terminating ABA protocol with $\mathbb{Q}^{(3)}$ condition. Previously, almost-surely terminating ABA protocol is known with *non-optimal* resilience where $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}$ condition; i.e. union of *no* four subsets from $\mathcal{Z}$ covers all the n parties. To design our protocol, we present a *shunning asynchronous verifiable secret-sharing* (SAVSS) scheme with $\mathbb{Q}^{(3)}$ condition, which is of independent interest.

ACM Reference Format:
Ashish Choudhury. 2022. **Almost-Surely Terminating Asynchronous Byzantine Agreement Against General Adversaries with Optimal Resilience. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn**

1 INTRODUCTION

Byzantine agreement (BA) [31], also known as fault-tolerant distributed consensus, is a fundamental problem in secure distributed computing. Informally, a BA protocol allows a set $\mathcal{P}$ of n mutually distrusting parties with *private* input bits, to reach agreement on a common output bit, even if a subset of the parties are corrupted by a *computationally-unbounded* malicious (Byzantine) adversary, who can force the corrupt parties to behave arbitrarily during the protocol execution. BA protocols serve as a very important building block in secure *multiparty computation* (MPC) protocols [9, 36]. The BA problem has been widely studied over the last three decades and several fundamental results have been achieved, regarding the possibility and feasibility of BA protocols in various settings (see for instance [4, 30]). Recently, the BA problem has also received attention from several other research communities, after the advent of blockchain technologies (see for instance [24]).

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The traditional way of characterizing the adversary is through a *threshold*, by assuming that adversary can corrupt any subset of up to t parties. In this setting, BA is is achievable iff $t < n/3$ [31]. Hirt and Maurer [26] and later Fitzi and Maurer [23] generalized the threshold model by introducing the *general-adversary* model (also known as the *non-threshold* setting in the literature). In the non-threshold setting, the adversary is characterized by an *adversary structure* $\mathcal{Z} = \{Z_1, \ldots, Z_h\} \subset 2^{\mathcal{P}}$, which enumerates all possible subsets of potentially corrupt parties, such that adversary can select any subset from $\mathcal{Z}$ for corruption during the execution of a protocol.

In the general-adversary model, BA is achievable iff $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition [23, 26].[1] There are several well-known motivations for modelling the distrust in the system through a non-threshold adversary (see for instance [19, 22, 25, 27]). For example, it allows for more flexibility, compared to the threshold model, especially when $\mathcal{P}$ is not too large. To understand this, let $\mathcal{P} = \{P_1, \ldots, P_6\}$. Then in the threshold setting, any BA protocol can tolerate at most 1 corrupt party. However, in the non-threshold setting, one can design a BA protocol tolerating an adversary characterized by the adversary structure $\mathcal{Z} = \{\{P_1\}, \{P_2, P_4\}, \{P_3, P_5\}, \{P_3, P_6\}, \{P_2, P_5, P_6\}, \{P_4, P_5, P_6\}\}$, where the adversary can corrupt up to 3 parties, by corrupting the subset $\{P_2, P_5, P_6\}$ or $\{P_4, P_5, P_6\}$.

Our Motivation and Results: All the above results are in the *synchronous* communication model, where the parties are assumed to be synchronized through a *global* clock, implying strict upper bounds on the message delays. Consequently, protocol execution occurs as a sequence of communication *rounds*, where the parties are well aware of the beginning and end of each round. In a synchronous protocol, any expected message which does not get delivered within the known time bound can be attributed to a *corrupt* sender party. Unfortunately, guaranteeing such strict time-outs in extremely difficult in the real-world networks like the Internet, which are better modelled through *asynchronous* communication model [13]. In the asynchronous model, *no* timing assumptions are made and the messages can be arbitrarily, but finitely delayed. The only guarantee is that every sent message is eventually delivered, but the messages need not be delivered in the same order in which they were sent. To model the worst case scenario, adversary is given the full control of the message scheduling in an asynchronous network. Consequently, no party can distinguish between a *corrupt* sender party (who does not send any message) and a *slow honest* sender party (whose messages are arbitrarily delayed). As an implication of this, in any asynchronous protocol, no party can afford to wait to receive messages from all the parties, to avoid an

[1] Given a set of parties $\mathcal{P}' \subseteq \mathcal{P}$, we say that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(k)}(\mathcal{P}', \mathcal{Z})$ condition, if the union of any k subsets from $\mathcal{Z}$ *does not* cover the entire set $\mathcal{P}'$.

endless wait and hence at every step of the protocol, a party may afford to receive messages from only a subset of parties, ignoring communication from potentially slow *honest* parties. As a result of this, asynchronous protocols are more challenging to design than their synchronous counterparts.

Unlike synchronous protocols, asynchronous protocols are executed as a sequence of events, based on the order in which messages are delivered. Following the results in the synchronous setting, *asynchronous* BA (ABA) is possible iff $t < n/3$ in the threshold setting, while ABA in the non-threshold setting is possible only if $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Compared to the synchronous BA protocols, there are several inherent limitations of ABA protocols. The seminal FLP impossibility result [21] states that *any* (deterministic) ABA protocol *must have* non-terminating runs, where the honest parties (who are not under the control of the adversary) keep on running the protocol forever to obtain an output. A powerful paradigm to circumvent this impossibility result is to embrace randomness, pioneered by Rabin [35] and Ben-Or [7]. There are two categories of randomized ABA protocols. The first category is that of $(1-\epsilon)$*-terminating* ABA protocols [14, 33], where the honest parties *may not* terminate the protocol with probability ϵ, where $\epsilon > 0$ is some error parameter. The second category is that of *almost-surely terminating* ABA protocols [1, 5], where the honest parties terminate the protocol, asymptotically with probability 1. While the first category of ABA protocols are used in *statistically-secure* asynchronous MPC (AMPC) protocols [10, 16] where a negligible error is allowed in the security properties, the latter category of protocols are used in *perfectly-secure* AMPC protocols [6, 8, 18, 34] where all security properties are achieved *without* any error.

In this work, we focus on almost-surely terminating ABA protocols. The works of [1, 5] present efficient almost-surely terminating ABA protocols in the *threshold* setting with the *optimal* resilience of $t < n/3$. However, to the best of our knowledge, we are *not* aware of any almost-surely terminating ABA protocol in the *non-threshold* setting with *optimal resilience*. Motivated by this, we ask the following central question:

Does there exist an almost-surely terminating ABA protocol where the adversary structure $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition?

We answer the above question affirmatively by presenting an almost-surely terminating ABA protocol with the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Our protocol is efficient and requires $O(n^2)$ expected running time, where the expected computation and communication performed by the parties is polynomial in n.

Related Work: Not much work has been done in the domain of ABA against non-threshold adversaries. In [17], the authors presented an *almost-surely* terminating ABA protocol. Compared to our protocol, the expected running time of their protocol is a *constant*. However, their protocol has *non-optimal* resilience, where the underlying adversary structure $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition. In a technical report [29], the authors refer to an ABA protocol in [28] with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, to show that the condition $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ is sufficient for designing ABA protocol. However, the work of [28] is *not* available in the public domain and the exact details of the ABA protocol are *not* known. Also, it is *not* clear whether the protocol in [28] is $(1-\epsilon)$ or almost-surely terminating. Given the importance of ABA and the general-adversary model, our work fills the gap in the literature by presenting an almost-surely terminating ABA protocol with complete formal details and formal security proofs.

Technical Overview: To design our ABA protocol, we generalize the framework of [1, 5, 37] in the *threshold* setting (which is further based on [7, 20, 35]), to the general-adversary model. The framework reduces ABA to the design of an asynchronous *common-coin* (CC) protocol, which allows the honest parties to output a common random bit, with certain non-zero *success probability*. The design of CC protocol is further reduced to another well-known primitive called *asynchronous verifiable secret sharing* (AVSS) [8, 15]. Informally, an AVSS scheme consists of a sharing-phase protocol and a reconstruction-phase protocol. During the sharing-phase, there exists a designated *dealer* with some *private* input (called secret), which it shares among the parties, without revealing anything about the secret to the adversary. The "verifiability" of the scheme guarantees that even if dealer is *corrupt*, it has "consistently" shared some value among the parties. During the reconstruction-phase protocol, the parties robustly reconstruct the value shared during the sharing-phase, even if the corrupt parties (including a potentially corrupt dealer) behaves maliciously.

It is well-known that *perfectly-secure* (error-free) AVSS in the *threshold* setting necessarily requires $t < n/4$ [2, 8]. Hence to design a CC protocol with $t < n/3$, the work of [1] introduces a *weaker* variant of AVSS called *shunning* AVSS (SAVSS). Intuitively, depending upon the behaviour of the adversary, an SAVSS scheme either guarantees all the security properties of an AVSS or ensures that some *honest* party is able to *locally* detect and shun at least one *corrupt* party (also called as *local-conflict*) for all future communication. Once all corrupt parties are shunned by all honest parties, then there will be no further errors and hence SAVSS will provide all the security guarantees of AVSS. Based on their SAVSS, [1] designed a shunning-variant of CC called *shunning common-coin* (SCC) protocol, where either all honest parties output a common random bit with certain success probability or some local-conflict occurs.

To generalize the framework of [1], we present a perfectly-secure SAVSS scheme with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The scheme guarantees that if the properties of AVSS are not achieved, then at least one *new* local-conflict occurs. Consequently, it may take $O(n^2)$ "failed" SAVSS instances before all corrupt parties are shunned by all honest parties. By deploying our SAVSS we then design an SCC protocol against general adversaries with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, where the success probability is $\frac{1}{n}$. Finally, this SCC protocol when used in the (generalized) framework of [7, 35] leads to our ABA protocol with $O(n^2)$ expected running time.

Open Problems: In the *threshold* setting, almost-surely terminating ABA protocol with *optimal resilience* (i.e. $t < n/3$) and *best* running time is due to [5], where the expected running time is $O(n)$. We do not know how to generalize the protocol of [5] and get an almost-surely terminating ABA protocol with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition and $O(n)$ expected running time. Designing an almost-surely terminating ABA protocol with a *constant* expected running time and with *optimal resilience* even against threshold adversaries has been a long-standing open problem.

2 PRELIMINARIES

We assume the *secure-channel* model, where the parties in $\mathcal{P} = \{P_1, \ldots, P_n\}$ are connected by pair-wise private and authentic channels. The distrust is modelled by a centralized *malicious* (Byzantine) adversary Adv, characterized by an adversary structure $\mathcal{Z} = \{Z_1, \ldots, Z_h\} \subset 2^{\mathcal{P}}$. The adversary is *static*, who decides the set of corrupt parties $Z^\star \in \mathcal{Z}$, at the beginning of the execution of any protocol. Parties not under the control of Adv are called *honest*. Given $\mathcal{P}' \subseteq \mathcal{P}$, we say that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(k)}(\mathcal{P}', \mathcal{Z})$ condition, if for every $Z_{i_1}, \ldots, Z_{i_k} \in \mathcal{Z}$, the condition $\mathcal{P}' \not\subseteq Z_{i_1} \cup \ldots \cup Z_{i_k}$ holds [27]. We follow the asynchronous communication model of [13], where the channels among the parties are asynchronous and where the messages are arbitrarily yet finitely delayed, with the guarantee that every sent message is eventually delivered.

In our protocols, each P_i maintains a local *block-set* $\mathcal{B}_i$ across *all* protocol instances and a *wait-set* $\mathcal{W}_i$, which are initialized to $\emptyset$. Note that P_i maintains a *single* $\mathcal{B}_i$ set, where as a *separate* $\mathcal{W}_i$ set is maintained for each SAVSS instance. Party P_i includes P_j in $\mathcal{B}_i$ if during some protocol instance, x is expected from P_j, but instead $x' \neq x$ is received. Party P_i is said to be in *local-conflict* with P_j when $P_j \in \mathcal{B}_i$. Party P_i includes P_j in $\mathcal{W}_i$ corresponding to some SAVSS instance, if during that instance, P_i is expecting some message from P_j. While a party making an entry in $\mathcal{B}_i$ remains part of it until the end of the execution of the ABA protocol, any entry in a wait set is *temporary* and removed as and when the expected communication happens. Until the receipt of the desired communication from a party in $\mathcal{W}_i$, party P_i suspends (saves yet does not use) its future communication. Looking ahead, the way $\mathcal{W}_i$ and $\mathcal{B}_i$ sets are created and maintained, it will be guaranteed that *no honest* party is every included in the $\mathcal{B}_i$ set of any *honest* P_i. Moreover, if any *honest* P_j is ever included in the $\mathcal{W}_i$ set of any *honest* P_i, then P_i eventually removes P_j from P_i.

In our SAVSS scheme, the parties perform computations over an algebraic structure $(\mathbb{K}, +, \cdot)$, which is either a finite ring or a field with $|\mathbb{K}| \geq n$. Looking ahead, this is required to achieve the desired success probability in our SCC protocol.

2.1 Definitions

We now present the various definitions, as used in the paper.

Definition 2.1 (Shunning Asynchronous Verifiable Secret Sharing (SAVSS)). Let $(\Pi_{\mathsf{Sh}}, \Pi_{\mathsf{Rec}})$ be a pair of protocols for the parties in $\mathcal{P}$, where each P_i maintains a local $\mathcal{B}_i$ and $\mathcal{W}_i$ set, and for a special party *dealer* $P_{\mathsf{D}} \in \mathcal{P}$ that has a private input $s \in \mathbb{K}$ for Π_{Sh}. Then $(\Pi_{\mathsf{Sh}}, \Pi_{\mathsf{Rec}})$ is an SAVSS scheme if the following requirements hold for every possible Adv.

- **Output Computation**: **(a)**: If P_{D} is *honest* and all honest parties participate in Π_{Sh}, then each honest party eventually obtains an output in Π_{Sh}. **(b)**: If some honest party obtains an output in Π_{Sh}, then every other honest party eventually obtains an output in Π_{Sh}. **(c)**: If all honest parties participate in Π_{Rec}, then every honest party eventually obtains an output in Π_{Rec}.
- **Correctness**: If the honest parties compute an output during Π_{Sh}, then there exists some $\overline{s} \in \mathbb{K}$, where $\overline{s} = s$ for an *honest* P_{D}, such that *one* of the following holds:
  - All honest parties output $\overline{s}$ during Π_{Rec}; or
  - Some corrupt parties are included in the $\mathcal{B}$ sets of some honest parties.
- **Privacy**: If P_{D} is *honest*, then the view of Adv during Π_{Sh} is independent of s.

Definition 2.2 (Shunning Common Coin (SCC)). Let Π_{SCC} be a protocol for the parties in $\mathcal{P}$, with each P_i maintaining a local $\mathcal{B}_i$ and $\mathcal{W}_i$ set and where each party has some local random input and a binary output. Then Π_{SCC} is a p-SCC protocol for a given p where $0 < p < 1$, if all the following hold for every possible Adv.

- **Completion**: If all honest parties participate in Π_{SCC}, then every honest party eventually obtains an output.
- **Correctness**: One of the following holds.
  - For every $\sigma \in \{0, 1\}$, all honest parties output σ with probability at least p; or
  - Some corrupt parties are included in the $\mathcal{B}$ sets of some honest parties.

Remark 2.3 (On the Termination Guarantees of Our Sub-Protocols). For simplicity and for the ease of analysis, we *do not* put any *termination* criteria for the various sub-protocols (including SAVSS and SCC) used in our ABA protocol and the parties may keep on running these sub-protocol instances, even after obtaining an output. Looking ahead, the termination criteria of our ABA protocol will ensure that the parties terminate all sub-protocol instances upon obtaining their respective outputs in the ABA protocol.

Definition 2.4 (Almost-Surely Terminating Asynchronous Byzantine Agreement (ABA)). Let Π_{ABA} be a protocol for the parties in $\mathcal{P}$, where each P_i has a private input bit b_i and a possible output bit. Then, Π_{ABA} is an almost-surely terminating ABA protocol, if the following requirements hold for every possible Adv.

- **Termination**: If all honest parties participate in Π_{ABA}, then asymptotically with probability one, each honest P_i eventually terminates the protocol. That is,

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

  where the probability is over the random coins of the honest parties and the adversary in the protocol.
- **Agreement**: All honest parties output a common bit.
- **Validity**: If all honest parties have the same input bit σ, then all honest parties eventually output σ.

2.2 Existing Asynchronous Primitives

We use the following existing asynchronous primitives.

Asynchronous Reliable Broadcast (Acast): An Acast protocol allows a designated *sender* $P_S \in \mathcal{P}$ to identically send a message $m \in \{0, 1\}^\ell$ to all the parties. If P_S is *honest*, then all honest parties eventually output m. On the other hand, if P_S is *corrupt* and some honest party outputs an $m^\star \in \{0, 1\}^\ell$ (where $m^\star$ may be *different* from m), then every other honest party eventually outputs $m^\star$. In [29], a perfectly-secure Acast protocol is presented with a communication complexity of $O(n^2\ell)$ bits, provided the underlying adversary structure $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The protocol is obtained by generalizing the Bracha's Acast protocol against threshold adversaries with $t < n/3$ [12]. We will use the following terminologies while invoking the Acast protocol: we will say that "P_i *broadcasts* m" to mean that P_i acts as a sender and invokes an instance of the Acast protocol with input m and the parties participate in this instance. Similarly, we will say that "P_j *receives m from the broadcast of* P_i" to mean that party P_j outputs m in the corresponding instance of Acast.

Asynchronous Vote Protocol: In a voting protocol (also known as *gradecast*), every party has a single bit as input and each party's output can have *five* different forms:

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

A voting protocol ensures the following properties:

– If each honest party has the same input σ, then every honest party outputs $(\sigma, 2)$.
– If some honest party outputs $(\sigma, 2)$, then every other honest party outputs either $(\sigma, 2)$ or $(\sigma, 1)$.
– If some honest party outputs $(\sigma, 1)$ and no honest party outputs $(\sigma, 2)$, then each honest party outputs either $(\sigma, 1)$ or $(\Lambda, 0)$.

In [14], a voting protocol satisfying the above requirements is presented against threshold adversaries, provided $t < n/3$ holds. The protocol is generalized against non-threshold adversaries in [17], provided the underlying $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The protocol called Π_{Vote} incurs a communication of $O(n^5)$ bits.

3 SAVSS WITH $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ CONDITION

In this section, we present our SAVSS scheme (see Fig 1). The scheme is obtained by modifying the perfectly-secure AVSS scheme of [17] (which is further based on [32]) and considers a *sharing specification* $\mathbb{S} = \{S_1, \ldots, S_h\}$, where for $q = 1, \ldots, h$ the set $S_q = \mathcal{P} \setminus Z_q$ and where $\mathcal{Z} = \{Z_1, \ldots, Z_h\}$ is the underlying adversary structure. During the sharing-phase protocol Π_{Sh}, the dealer randomly picks a vector of shares $[s] = (s_1, \ldots, s_h)$ for its secret s, such that $s = s_1 + \ldots + s_h$ and distributes the share s_q, denoted by $[s]_q$, to all the parties in the set S_q. If the dealer is *honest*, then this does not reveal any information about s to the adversary, as adversary will be missing at least one share and so the probability distribution of the shares learnt the adversary will be independent of s. To verify whether a potentially *corrupt* dealer has distributed a common share $[s]_q$ to all the honest parties in the set S_q, the parties in S_q exchange the supposedly common share and publicly acknowledge if they received the same share. Based on these public acknowledgements, the parties check whether for each set $S_q \in \mathbb{S}$, there exists a subset of parties S'_q who have acknowledged the receipt of a common share from the dealer, such that S'_q is guaranteed to have at least one *honest* party. Notice that such a subset S'_q is bound to exist for an *honest* dealer, since for an *honest* dealer, all the *honest* parties in S_q will receive the same share from the dealer and the set of honest parties in each set S_q is *non-empty*, which follows from the fact that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. To ensure that all honest parties have an agreement on the subsets S'_q, the dealer is assigned the task of identifying the subsets S'_q and publicly declaring them.

To find the subsets S'_q, the dealer actually finds a *single* "core" subset of parties C where $\mathcal{P} \setminus C \in \mathcal{Z}$, such that corresponding to every $S_q \in \mathbb{S}$, there exists a subset $S'_q \subseteq (S_q \cap C)$, where $S_q \setminus S'_q \in \mathcal{Z}$ and where *all* the parties in S'_q have publicly acknowledged the receipt of a common share from the dealer. An *honest* dealer will eventually find such a core set C, as the set of *honest* parties in the system constitute a candidate C for an *honest* dealer. Upon finding a set C satisfying the above conditions, dealer publicly announces it and the parties "accept" the set C after verifying whether indeed the announced C satisfies the above properties.

During the reconstruction-phase protocol Π_{Rec}, the goal is to get the share $[s]_q$ corresponding to every set $S_q \in \mathbb{S}$, upon which the shared secret can be reconstructed by computing $[s]_1 + \ldots + [s]_h$. To get the share $[s]_q$ corresponding to S_q, *all* the parties in the set $(S_q \cap C)$ are asked to make the share received from the dealer public. However, due to the asynchronous communication, to avoid an indefinite wait, the parties *cannot* afford for *all* the parties in $(S_q \cap C)$ to make their version of the share $[s]_q$ public, as the potentially *corrupt* parties in $(S_q \cap C)$ may never make any share public. Consequently, as soon as *any* party from $(S_q \cap C)$ makes public its version of $[s]_q$, it is considered towards the reconstruction of s. However, this may lead to reconstruction of an *incorrect* s, as a potentially *corrupt* party from $(S_q \cap C)$ may make public an *incorrect* version of $[s]_q$. But this will lead to the honest party(ies) in S'_q getting into local-conflict with this corrupt party.

Each instance of SAVSS is associated with a unique id $\mathsf{sid} \in \mathbb{N}$. All messages communicated during the SAVSS instance sid are tagged with this id. However, for simplicity, we skip tagging every message explicitly with sid in the formal description of the scheme. During the protocol Π_{Sh}, once the core set C is agreed upon, the parties *locally* populate their respective $\mathcal{W}$ sets, anticipating the values they expect from the various parties during the protocol Π_{Rec}. At the beginning of each instance of the SAVSS, a corresponding *memory management* protocol Π_{MM} is also invoked, based on which each party locally decides whether to process a received message as per the SAVSS, delay it temporarily or block it permanently. Protocol Π_{MM} examines the messages produced by the various parties during the reconstruction phase and accordingly the $\mathcal{W}$ and $\mathcal{B}$ sets of the parties are updated. We stress that the parties keep executing the Π_{MM} protocol with id sid, even after obtaining their respective outputs in the Π_{Sh} and Π_{Rec} protocols with id sid. This ensures that if some message is pending from a party for instance sid, then its communication is ignored by the Π_{MM} protocol in any future instance $\mathsf{sid}' > \mathsf{sid}$.

Scheme SAVSS

Protocol $\Pi_{\mathsf{Sh}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, P_{\mathsf{D}}, s)$

- **Distribution of Shares** – On having the input $s \in \mathbb{K}$, the dealer P_{D} executes the following steps.
  – Randomly select the shares $s_1, \ldots, s_h$, subject to the condition that $s_1 + \ldots + s_h = s$ holds.
  – For $q = 1, \ldots, h$, set $[s]_q = s_q$ and send $[s]_q$ to all the parties in the set S_q.
- **Pair-wise Consistency Check** – Each party $P_i \in \mathcal{P}$ (including P_{D}), executes the following steps.
  – Wait to receive a share s_{qi} from P_{D}, corresponding to every $S_q \in \mathbb{S}$ such that $P_i \in S_q$. Upon receiving, send the share s_{qi} to every party $P_j \in S_q$.
  – Broadcast an $\mathsf{OK}(i, j)$ message, if all the following hold.

1. P_i has received a share s_{qj} from P_j, corresponding to every $S_q \in \mathbb{S}$ such that $P_i, P_j \in S_q$;
2. The condition $s_{qi} = s_{qj}$ holds.

- **Constructing CORE Set and Public Announcement** — P_{D} executes the following steps.
  - Check if there exists a subset of parties $C \subseteq \mathcal{P}$, such that all the following hold. Upon finding such a C, broadcast it.

1. $\mathcal{P} \setminus C \in \mathcal{Z}$;
2. Corresponding to every $P_i, P_j \in C$, the messages $\mathsf{OK}(i, j)$ and $\mathsf{OK}(j, i)$ have been received from the broadcast of P_i and P_j respectively.
3. Corresponding to every $S_q \in \mathbb{S}$, there exists a subset $S'_q \subseteq (S_q \cap C)$, such that $S_q \setminus S'_q \in \mathcal{Z}$.

- **Verifying CORE Set and Populating Waiting Sets** – Each $P_i \in \mathcal{P}$ (including P_{D}) executes the following steps.
  - Wait to receive a set C from the broadcast of P_{D} and then *accept* and output C, if all the following hold.

1. $\mathcal{P} \setminus C \in \mathcal{Z}$;
2. Corresponding to every $P_i, P_j \in C$, the messages $\mathsf{OK}(i, j)$ and $\mathsf{OK}(j, i)$ have been received from the broadcast of P_i and P_j respectively.
3. Corresponding to every $S_q \in \mathbb{S}$, there exists a subset $S'_q \subseteq (S_q \cap C)$, such that $S_q \setminus S'_q \in \mathcal{Z}$.

  - If a set C is accepted, then populate the set $\mathcal{W}_{(i,\mathsf{sid})}$ as follows.
    - If $P_i = P_{\mathsf{D}}$, then for each $P_j \in C$ and each $S_q \in \mathbb{S}$ where $P_j \in S_q$, add the tuple $(q, P_j, [s]_q)$ to $\mathcal{W}_{(i,\mathsf{sid})}$. This is interpreted as P_{D} expects P_j to reveal the share $[s]_q$ on the behalf of the set S_q during the reconstruction protocol.
    - If $P_i \in C$, then corresponding to each $S_q \in \mathbb{S}$ where $P_i \in S_q$ and each $P_j \in S_q \cap C$, add the tuple (q, P_j, s_{qi}) to $\mathcal{W}_{(i,\mathsf{sid})}$. This is interpreted as P_i expects P_j to reveal the share s_{qi} on the behalf of the set S_q during the reconstruction protocol.
    - Else corresponding to every $P_j \in C$ and every $S_q \in \mathbb{S}$ where $P_j \in S_q$, add the tuple $(q, P_j, \star)$ to $\mathcal{W}_{(i,\mathsf{sid})}$. This is interpreted as P_i expects P_j to reveal some share on the behalf of the set S_q during the reconstruction protocol.

Protocol $\Pi_{\mathsf{Rec}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, P_{\mathsf{D}}, s)$

- **Making the Shares Public** – For $q = 1, \ldots, h$, each party $P_i \in C \cap S_q$ broadcasts the shares $\{s_{qi}\}_{P_i \in S_q \cap C}$.
- **Reconstructing the Secret** – Each $P_j \in \mathcal{P}$ outputs $s^\star = [s]^\star_1 + \ldots + [s]^\star_h$, where for $q = 1, \ldots, h$, party P_j computes $[s]^\star_q$ as follows.
  - If $P_j \in (C \cap S_q)$, then set $[s]^\star_q = s_{qj}$.
  - If $P_j \notin (C \cap S_q)$, then set $[s]^\star_q$ to the value s_{qi}, if s_{qi} is received from the broadcast of some $P_i \in C \cap S_q$.[a]

Protocol $\Pi_{\mathsf{MM}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, P_{\mathsf{D}}, s)$

The following code is executed by $P_i \in \mathcal{P}$, if a set C is accepted:

- **Initialization**: Initialize $\mathcal{W}_{(i,\mathsf{sid})}$ and $\mathcal{B}_i$ to $\emptyset$. The set $\mathcal{B}_i$ is a set that is initialized by the party P_i only once (when sid = 0) and dynamically updated during the various instances of Π_{MM}. The set $\mathcal{W}_{(i,\mathsf{sid})}$ is initialized in and maintained for SAVSS instance with id sid only.
- **Suspending Messages**: If any message is received from P_j during SAVSS with id sid, then block the message as per the following conditions.
  - If $P_j \in \mathcal{B}_i$, then discard the message.
  - If there exists a tuple of the form $(\star, P_j, \star)$ in the $\mathcal{W}_{(i,\mathsf{sid}')}$ set for any $\mathsf{sid}' < \mathsf{sid}$, then do not forward the message to SAVSS with id sid.
- **Filtering Parties from Waiting Lists**: Corresponding to every $P_j \in \mathcal{P}$, if $P_j \notin \mathcal{B}_i$ and if a share s_{qj} is received from the broadcast of P_j on the behalf of the set S_q during Π_{Rec} with id sid, then do the following, provided $P_j \in S_q \cap C$ and there exists a tuple of the form $(q, P_j, \mathsf{val}) \in \mathcal{W}_{(i,\mathsf{sid})}$.
  - If $\mathsf{val} = \star$, then remove (q, P_j, val) from $\mathcal{W}_{(i,\mathsf{sid})}$.
  - If $\mathsf{val} \neq \star$ and $\mathsf{val} = s_{qj}$, then remove (q, P_j, val) from $\mathcal{W}_{(i,\mathsf{sid})}$.
  - If $\mathsf{val} \neq \star$ and $\mathsf{val} \neq s_{qj}$, then add P_j to $\mathcal{B}_i$.

[a] If there are multiple such parties P_i from whose broadcast P_j receives some s_{qi}, then consider the first party P_i among them.

Figure 1: The SAVSS scheme with protocols for sharing, reconstruction and memory management with session id sid

We next prove the properties of our SAVSS.

Lemma 3.1 (**Properties of SAVSS Memory Management Protocol**). *The following hold for every honest $P_i \in \mathcal{P}$ during the protocol Π_{MM} with id* sid *for any* $\mathsf{sid} \in \mathbb{N}$, *irrespective of P_{D}.*

- *If P_j is included in $\mathcal{B}_i$ then P_j is corrupt.*
- *If P_j is honest, then any triplet of the form $(\star, P_j, \star)$ present in $\mathcal{W}_{(i,\mathsf{sid})}$ is eventually removed.*

Proof. If P_j is *honest*, then it eventually broadcasts all the messages it is supposed to broadcast during the protocol Π_{Rec} with id sid, which are eventually received by every honest P_i. Hence every triplet of the form $(\star, P_j, \star)$ will be eventually removed from $\mathcal{W}_{(i,\mathsf{sid})}$. Moreover, an *honest* P_j broadcasts the shares as received from P_{D} during the protocol Π_{Sh} with id sid. So the conditions for including P_j to $\mathcal{B}_i$ are never satisfied. □

Lemma 3.2. *For any* $\mathsf{sid} \in \mathbb{N}$, *the following hold during the* $(\Pi_{\mathsf{Sh}}, \Pi_{\mathsf{Rec}})$ *protocols with id* sid: **(a)**: *If P_{D} is honest and all honest parties participate in Π_{Sh}, then each honest party eventually outputs C.* **(b)**: *If some honest party outputs C in Π_{Sh}, then every other honest party eventually outputs C in Π_{Sh}.* **(c)**: *If all honest parties participate in Π_{Rec}, then every honest party eventually obtains an output in Π_{Rec}.*

Proof. We note that during Π_{Sh} and Π_{Rec}, the messages of every honest party is cleared by the Π_{MM} protocol and eventually delivered to the honest recipients, which follows from Lemma 3.1. We first consider an *honest* P_{D}. In this case, corresponding to every $S_q \in \mathbb{S}$, every *honest* party $P_i \in S_q$ receives the share s_{qi} from P_{D}, which will be the *same* as $[s]_q$. Consequently, every honest P_i eventually broadcasts an $\mathsf{OK}(i, j)$ message, corresponding to every *honest* P_j. Let $Z_c \in \mathcal{Z}$ be the set of *corrupt* parties and let $\mathcal{H} = \mathcal{P} \setminus Z_c$ be the set of *honest* parties. Next consider an arbitrary $S_q \in \mathbb{S}$. It is easy to see that $S_q \setminus (S_q \cap \mathcal{H}) \subseteq Z_c \in \mathcal{Z}$. It then follows that P_{D} eventually finds a candidate C set and broadcasts the same, since we have shown that the set $\mathcal{H}$ satisfies all the properties of a candidate C set. Consequently, every honest party will eventually receive a C set from the broadcast of P_{D}, accepts it and outputs C. This proves the first part of the lemma.

For the second part of the lemma, let P_h be the *first* honest party who outputs C during Π_{Sh}. This implies that P_h receives the set C from the broadcast of P_{D} and accepts it. It then follows that every other honest party also eventually receives C from the broadcast of P_{D} and the conditions for accepting C will eventually hold for those honest parties as well. Consequently, every other honest party eventually outputs C during Π_{Sh}.

For the third part, we first note that honest parties participate in Π_{Rec}, only after accepting C. We also note that corresponding to *every* $S_q \in \mathbb{S}$, there exists at least one *honest* party in the set $S_q \cap C$, that is $S_q \cap C \cap \mathcal{H} \neq \emptyset$. This is because as per the protocol conditions, $\mathcal{P} \setminus C \in \mathcal{Z}$, $S_q = \mathcal{P} \setminus Z_q$ and $\mathcal{H} = \mathcal{P} \setminus Z_c$, so if $S_q \cap C \cap \mathcal{H} = \emptyset$, then it implies that $\mathcal{Z}$ *does not* satisfy the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, which is a contradiction. Let P_j be an *honest* party in the set $S_q \cap C$. During Π_{Rec}, party P_j will broadcast the share s_{qj} received from P_{D} on the behalf of the set S_q, which is eventually delivered to every honest party. Consequently, every *honest* party P_i will eventually set the share $[s]^{\star}_q$ to some value and reconstructs some value $s^{\star}$. □

Lemma 3.3. *For any* sid $\in \mathbb{N}$*, the following holds during the* $(\Pi_{\mathsf{Sh}}, \Pi_{\mathsf{Rec}}, \Pi_{\mathsf{MM}})$ *protocols with id* sid*: If the honest parties output C during Π_{Sh}, then there exists a unique value $\overline{s}$, where $\overline{s} = s$ for an honest P_{D}, such that one of the following holds.*

- *All honest parties output $s^{\star} = \overline{s}$ during Π_{Rec}; or*
- *At least one new local-conflict occurs during Π_{MM} between an honest and a corrupt party.*

Proof. Let the honest parties output a set C during Π_{Sh}. This implies that the honest parties have accepted C, broadcasted by P_{D}. Now consider an arbitrary $S_q \in \mathbb{S}$. As shown in the proof of Lemma 3.2, there exists at least one *honest* party in the set $S_q \cap C$. We first note that *all honest* parties in the set $S_q \cap C$ receive the same share, say s_q, from P_{D}. This is because every honest party $P_k \in S_q \cap C$ has broadcasted an $\mathsf{OK}(k, l)$ message, corresponding to every honest $P_l \in S_q \cap C$, implying that the condition $s_{qk} = s_{ql}$ holds, where s_{qk} and s_{ql} are the shares corresponding to S_q, received by P_k and P_l respectively. We define

$$\overline{s} \stackrel{def}{=} \sum_{q=1,\ldots,h} s_q.$$

It is easy to see that if P_{D} is *honest*, then $\overline{s} = s$ holds.

Next consider an arbitrary *honest* party P_j. From Lemma 3.2, party P_j eventually reconstructs some value $s^{\star}$ during Π_{Rec}. Let $s^{\star} \neq \overline{s}$. This implies that there exists at least one $S_q \in \mathbb{S}$, such that during Π_{Rec}, party P_j has set $[s]^{\star}_q$ to s_{qi} where s_{qi} is received from the broadcast of some $P_i \in S_q \cap C$ and $s_{qi} \neq s_q$. Note that P_i is *corrupt*, as every *honest* party in $S_q \cap C$ broadcasts the share s_q during Π_{Rec}. We also note that the share s_{qi} broadcasted by P_i is eventually received by *all honest* parties in $S_q \cap C$. Now consider an arbitrary *honest* party $P_k \in S_q \cap C$. From the protocol steps, during Π_{Sh}, party P_k broadcasts the message $\mathsf{OK}(k, i)$ corresponding to P_i and adds the tuple (q, P_i, s_q) to $\mathcal{W}_{(k,\mathsf{sid})}$. Since during Π_{Rec} party P_i broadcasts $s_{qi} \neq s_q$, it follows that during the protocol Π_{MM}, the local-conflict (P_k, P_i) occurs and party P_k adds P_i to $\mathcal{B}_k$.

Finally, to complete the proof we need to show that P_i is *not* included in $\mathcal{B}_k$ during any instance of Π_{MM} with sid$'$, where sid$'$ < sid. On contrary, if P_i is included in $\mathcal{B}_k$ during Π_{MM} with sid$'$, where sid$'$ < sid, then P_k will never broadcast the $\mathsf{OK}(k, i)$ message during the instance of Π_{Sh} with id sid, which is a contradiction. This is because if $P_i \in \mathcal{B}_k$, then all the messages from P_i are discarded by P_k during the instance of Π_{Sh} with id sid, due to the protocol Π_{MM} with id sid. □

Lemma 3.4. *For any* sid $\in \mathbb{N}$*, if P_{D} is honest, then the view of the adversary is independent of the input s of P_{D} during Π_{Sh} with id* sid.

Proof. Let P_{D} be *honest*. In the protocol, P_{D} randomly selects the shares $s_1, \ldots, s_h$, subject to the condition that $s_1 + \ldots + s_h = s$ holds. Moreover, since each $S_q = \mathcal{P} \setminus Z_q$ where $Z_q \in \mathcal{Z}$, it follows that the probability distribution of the shares learnt by the adversary during Π_{Sh}, will be independent of the input s of P_{D}. More specifically, let $Z_c \in \mathcal{Z}$ be the set of *corrupt* parties. Then throughout Π_{Sh}, the adversary does not not learn anything about the share s_c, which is available only to the parties in $S_c = \mathcal{P} \setminus Z_c$. This is because the share s_c is distributed by P_{D} to the parties in S_c over pair-wise secure channels. Moreover, during the pair-wise consistency tests, the parties in S_c exchange the share s_c only among themselves. It now follows that for every candidate value of s from the point of view of the adversary, there is a corresponding unique s_c, which will be consistent with the shares seen by the adversary during Π_{Sh}. Now since the share s_c is chosen randomly, it follows that the view of the adversary during Π_{Sh} remains independent of s. □

Lemma 3.5. *Protocol Π_{Sh} incurs a communication of $O(|\mathcal{Z}| \cdot n^2 \log |\mathbb{K}| + n^4 \log n)$ bits, while protocol Π_{Rec} incurs a communication of $O(|\mathcal{Z}| \cdot n^3 \log |\mathbb{K}|)$ bits.*

Proof. During Π_{Sh}, the dealer P_{D} needs to send a share s_q of size $\log |\mathbb{K}|$ bits to each set $S_q \in \mathbb{S}$, consisting of $O(n)$ parties. This requires a communication of $O(|\mathbb{S}| \cdot n \log |\mathbb{K}|)$ bits. During the pair-wise consistency test, every party in S_q sends its share to every other party in S_q, incurring a communication of $O(|\mathbb{S}| \cdot n^2 \log |\mathbb{K}|)$ bits. There are $O(n^2)$ $\mathsf{OK}(\star, \star)$ messages which are broadcasted, where each message is of size $2 \log n$ bits, encoding the identity of 2 parties. Moreover, P_{D} broadcasts a C set, encoded using $O(n \log n)$ bits. During Π_{Rec}, a party may need to broadcast up to $O(|\mathbb{S}|)$ shares. The communication complexity now follows from the communication complexity of the Acast protocol and the fact that $|\mathbb{S}| = |\mathcal{Z}|$. □

4 SCC WITH $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ CONDITION

The SCC protocol is presented in Fig 3. The protocol is almost the same as the CC protocol of [17] with the following *difference*: while the CC protocol of [17] uses their perfectly-secure AVSS with $\mathbb{Q}^{(4)}(\mathcal{P}, \mathcal{Z})$ condition, we *replace* the instances of AVSS with our SAVSS scheme with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition (presented in the last section), thus leading to an SCC with $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition.

The SCC protocol Π_{SCC} consists of two *stages*. In the first stage, a uniformly random, yet *unknown* value $\mathsf{Coin}_i \in \{0, \ldots, n-1\}$ is "attached" to every party P_i. Then, once it is ensured that "sufficiently many" number of parties in a set FS (called *final set*) 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 the reconstructed values. However, due to the asynchronous communication, each (honest) party may have a *different* FS set and hence, a potentially different output. To circumvent this problem, the protocol ensures that there is a *non-empty* overlap among the FS sets of all the (honest) parties. Ensuring this common overlap is the crux of the protocol.

During the first stage, each party acts as a dealer and shares a random value from $\mathbb{K}$ on the behalf of each party, by invoking an instance of Π_{Sh}. To ensure that each $\mathsf{Coin}_i \in \{0, \ldots, n-1\}$, the parties set $\mathbb{K}$ to either a finite ring or a field, where $|\mathbb{K}| \geq n$. Each party P_i creates a dynamic set of *accepted dealers* $\mathcal{AD}_i$, which includes all the dealers in whose Π_{Sh} instances, P_i computes an output. The properties of Π_{Sh} guarantee that these dealers are eventually included in the set of accepted dealer of every other honest party as well. Party P_i then waits for "sufficiently many" 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). 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. Note that the value of Coin_i will *not* be known to anyone at this point (including P_i), as the value(s) shared by the honest dealer(s) in the set AD_i on the behalf of P_i is(are) not yet known, owing to the *privacy* property of Π_{Sh}.

On receiving AD_j from any P_j, party P_i verifies if the set is "valid" by checking if the Π_{Sh} instances of dealers in AD_j has produced an output for P_i as well; i.e. ,$\mathsf{AD}_j \subseteq \mathcal{AD}_i$ holds. Once the validity of AD_j is confirmed, P_i "partially accepts" P_j and includes it in its set of *partially-accepted parties* $\mathcal{PAP}_i$. Additionally, P_i publicly "approves" the partial-acceptance of P_j, by broadcasting an OK message for P_j (this implicitly means P_i's approval for the yet unknown, but well-defined value Coin_j). Notice that partial-acceptance of P_j by P_i implies that every other honest party will also eventually partially accept P_j. This is because every other honest party will eventually receive the set AD_j and will find it valid.

Party P_i then waits for the approval of AD_j from a set of parties S_j *including itself*, such that $\mathcal{P} \setminus S_j \in \mathcal{Z}$ holds, thus guaranteeing that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(2)}(S_j, \mathcal{Z})$ condition. Once $\mathcal{P} \setminus S_j \in \mathcal{Z}$ holds, P_j is shifted from $\mathcal{PAP}_i$ to a dynamic set of *accepted parties* $\mathcal{AP}_i$. Looking ahead, ensuring that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(2)}(S_j, \mathcal{Z})$ condition, *coupled* with maintaining separate $\mathcal{PAP}_i$ and $\mathcal{AP}_i$ sets, plays a crucial rule to ensure a *non-empty* overlap among the FS sets of honest parties. 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 OK messages) for AD_j are publicly broadcasted. Party P_i keeps on expanding its set of accepted parties $\mathcal{AP}_i$ until $\mathcal{P} \setminus \mathcal{AP}_i \in \mathcal{Z}$ holds, which happens eventually. Once ensured, P_i publicly announces it with a ready message and the corresponding $\mathcal{AP}_i$ set, denoted by AP_i, along with its set of partially-accepted parties $\mathcal{PAP}_i$.[2]

On receiving the ready message and $\mathsf{AP}_j, \mathcal{PAP}_j$ sets from any party P_j, party P_i verifies the "validity" of AP_j and $\mathcal{PAP}_j$. For this, P_i checks if the parties in AP_j are accepted by P_i as well; i.e. if $\mathsf{AP}_j \subseteq \mathcal{AP}_i$ holds. And if the partially-accepted parties in $\mathcal{PAP}_j$ are either accepted or partially-accepted by P_i; i.e. if $\mathcal{PAP}_j \subseteq (\mathcal{AP}_i \cup \mathcal{PAP}_i)$ 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 and partially-accepted parties (and hence Coin values). 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 Flag_i to 1, marking the beginning of the second stage. Let SP_i denote the set of supportive parties $\mathcal{SP}_i$ when Flag_i is set to 1.

The second stage involves publicly reconstructing the unknown Coin values which were accepted and partially-accepted by P_i till it sets Flag_i to 1. Let FS_i be the set of accepted and partially-accepted parties when Flag_i is set to 1. This implies that the union of the AP_j and $\mathcal{PAP}_j$ sets of *all* the parties in SP_i is a subset of FS_i, as each $(\mathsf{AP}_j \cup \mathcal{PAP}_j) \subseteq (\mathcal{AP}_i \cup \mathcal{PAP}_i)$. 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 Π_{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, party P_i sets its output to 0, else, it outputs 1.

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 who broadcasts a ready message and show that the set AP_i will be the common overlap (see Lemma 4.3).[3] For the ease of understanding, we pictorially depict the various sets computed in the protocol Π_{SCC} in Fig 2. Notice that the parties may end up reconstructing a *different* coin value $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$, corresponding to $P_k \in \mathsf{FS}_i$, which will end up affecting the required success probability of the protocol Π_{SCC}. However, in this case, at least one *new* local-conflict occurs between an honest and a corrupt party.

Protocol $\Pi_{\mathsf{SCC}}(\mathcal{P}, \mathcal{Z})$

1. **Initialization**: Initialize a set of *accepted dealers* $\mathcal{AD}_i$, a set of *partially-accepted parties* $\mathcal{PAP}_i$, a set of *accepted parties* $\mathcal{AP}_i$ and a set of *supportive parties* $\mathcal{SP}_i$ to $\emptyset$. Additionally, initialize a Boolean variable $\mathsf{Flag}_i = 0$.
2. **Sharing Secrets on Behalf of Others**:
   - For $j = 1, \ldots, n$, choose a random secret $s_{ij} \in \mathbb{K}$ on the behalf of P_j, and as a dealer, invoke an instance $\Pi_{\mathsf{Sh}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, P_i, s_{ij})$ of Π_{Sh} with id (sid, P_i, P_j). Let this instance be denoted as $\Pi_{\mathsf{Sh}}^{(ij)}$.
   - Participate in $\Pi_{\mathsf{Sh}}^{(jk)}$, corresponding to every $P_j, P_k \in \mathcal{P}$.
3. **Populating the Set of Accepted Dealers**:
   - Add party P_j to $\mathcal{AD}_i$, if an output is computed in the instances $\Pi_{\mathsf{Sh}}^{(j1)}, \ldots, \Pi_{\mathsf{Sh}}^{(jn)}$.
   - If $(\mathcal{P} \setminus \mathcal{AD}_i) \in \mathcal{Z}$, 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
   $$\mathsf{Coin}_i \stackrel{def}{=} \sum_{P_j \in \mathsf{AD}_i} s_{ji} \mod n.$$
   We say that the *coin* Coin_i *is attached to party* P_i.[a]
4. **Validating the Set of Accepted Dealers**:
   - If the message $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ is received from the broadcast of P_j, then broadcast a message $\mathsf{OK}(P_i, P_j)$, if the dealers

[2]We note that the sets $\mathcal{AP}_i$ were *not* built in the CC protocol of [17]. However, these sets are *essential* even in their CC protocol, as otherwise a non-empty overlap among the FS sets of honest parties is *not* guaranteed in their CC protocol.

[3]This lemma is borrowed from [17]. However, the proof provided in [17] is flawed, as they *do not* consider the $\mathcal{PAP}$ sets in their protocol. The proof presented in this article, can be considered as the correct proof of the lemma for their CC protocol.

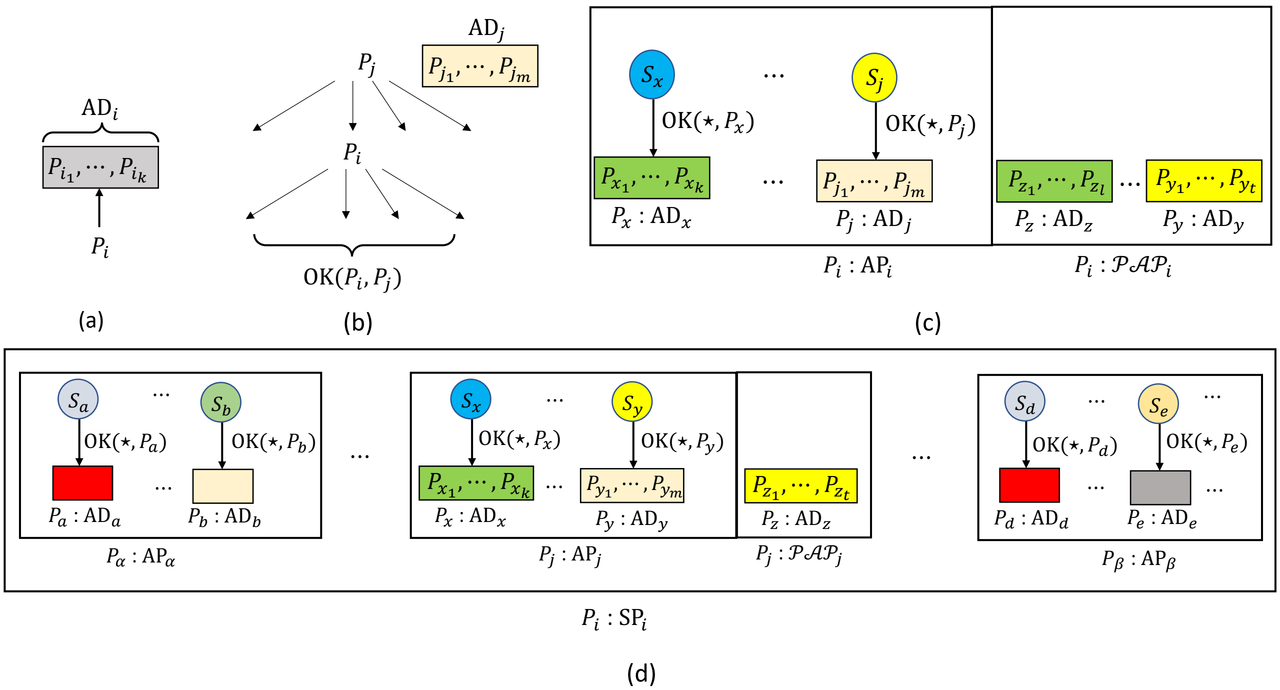

Figure 2: Pictorial depiction of the various sets computed in the protocol Π_{SCC}. The set AD_i in figure (a) denotes the set of dealers attached to P_i; the values shared by the parties in AD_i on the behalf of P_i define the value Coin_i. Figure (b) denotes P_j broadcasting AD_j. Upon receiving AD_j from P_j, party P_i approves AD_j through the $\mathsf{OK}(P_i, P_j)$ message. Figure (c) denotes the set of accepted parties AP_i and the set of partially-accepted parties $\mathcal{PAP}_i$ for P_i. For every $P_j \in \mathsf{AP}_i$, there is a corresponding set S_j, who has approved AD_j and hence Coin_j. Finally, figure (d) denotes the set SP_i.

attached to P_j are accepted by P_i, i.e. $\mathsf{AD}_j \subseteq \mathcal{AD}_i$ holds. Moreover, include P_j to $\mathcal{PAP}_i$.

5. **Populating the Set of Accepted Parties**:
   - Shift P_j from $\mathcal{PAP}_i$ to $\mathcal{AP}_i$, if $\mathsf{OK}(\star, P_j)$ is received from the broadcast of a set of parties S_j *including* P_i, such that $(\mathcal{P} \setminus S_j) \in \mathcal{Z}$.
   - If $(\mathcal{P} \setminus \mathcal{AP}_i) \in \mathcal{Z}$, then assign $\mathsf{AP}_i = \mathcal{AP}_i$ and broadcast the message $(\mathsf{ready}, P_i, \mathsf{AP}_i, \mathcal{PAP}_i)$.
6. **Populating the Set of Supportive Parties**:
   - Consider P_j to be *supportive* and include it in $\mathcal{SP}_i$, if P_i receives the message $(\mathsf{ready}, P_j, \mathsf{AP}_j, \mathcal{PAP}_j)$ from the broadcast of P_j and each party in AP_j is accepted by P_i and each party in $\mathcal{PAP}_j$ is either accepted or partially-accepted by P_i, i.e. $\mathsf{AP}_j \subseteq \mathcal{AP}_i$ and $\mathcal{PAP}_j \subseteq (\mathcal{AP}_i \cup \mathcal{PAP}_i)$ holds.
   - If $\mathcal{P} \setminus \mathcal{SP}_i \in \mathcal{Z}$, then set $\mathsf{Flag}_i = 1$. Let SP_i and FS_i be the contents of $\mathcal{SP}_i$ and $(\mathcal{AP}_i \cup \mathcal{PAP}_i)$ respectively, when $\mathsf{Flag}_i = 1$.[b]
7. **Reconstructing the Coin Values**:
   - If $\mathsf{Flag}_i = 1$, then reconstruct the value of the coin attached to each party in FS_i as follows:
     - Start participating in the instances $\Pi_{\mathsf{Rec}}(\mathcal{P}, \mathcal{Z}, \mathbb{S}, P_j, s_{jk})$ with id (sid, P_j, P_k) corresponding to each $P_j \in \mathsf{AD}_k$, such that $P_k \in \mathsf{FS}_i$, Denote this instance of Π_{Rec} as $\Pi_{\mathsf{Rec}}^{(jk)}$ and let r_{jk} be the corresponding output (Some parties may be included in the set $\mathcal{AP}_i \cup \mathcal{PAP}_i$ after Flag_i is set to 1. In that case, start participating in the corresponding Π_{Rec} instances).
     - For every $P_k \in \mathsf{FS}_i$, compute
$$\mathsf{Coin}'_k = \sum_{P_j \in \mathsf{AD}_k} r_{jk} \mod n.$$
       We say that the value Coin'_k is *associated to* P_k.
8. **Output computation**:
   - If the values associated to all the parties in FS_i are computed, then do the following.
     - If there exists any $P_k \in \mathsf{FS}_i$ where $\mathsf{Coin}'_k = 0$, 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 $\bigcup_{P_j \in \mathsf{SP}_i} (\mathsf{AP}_j \cup \mathcal{PAP}_j) \subseteq \mathsf{FS}_i$ holds, as $(\mathsf{AP}_j \cup \mathcal{PAP}_j) \subseteq (\mathcal{AP}_i \cup \mathcal{PAP}_i)$.

Figure 3: The shunning common-coin protocol for session id sid. The above code is executed by every $P_i \in \mathcal{P}$

The properties of Π_{SCC} are stated in the following lemmas and theorem, which are proved in Appendix A due to space constraints.

Lemma 4.1. *If each honest party participates in Π_{SCC} with id sid, then each honest party eventually computes an output.*

Lemma 4.2. *During Π_{SCC} with id sid, if any honest party receives the message $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$ from the broadcast of any party P_k, then a unique value Coin_k is fixed such that all the following hold:*

– *The coin* Coin_k *is attached to* P_k.
– *The value* Coin_k *is distributed uniformly over* $\{0,\ldots,n-1\}$ *and is independent of the coins attached to the other parties.*
– *If any honest party associates* $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$ *to* P_k*, then at least one new local-conflict occurs between an honest and a corrupt party.*

Lemma 4.3. *In* Π_{SCC} *with id* sid*, once some honest party sets* Flag *to 1, then there exists a set, say* $\mathcal{M}$*, such that all the following hold:*
– $\mathcal{P} \setminus \mathcal{M} \in \mathcal{Z}$.
– *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.
– *Whenever any honest party* P_i *sets its* $\mathsf{Flag}_i = 1$*, the condition* $\mathcal{M} \subseteq \mathsf{FS}_i$ *holds.*

Lemma 4.4. *In* Π_{SCC} *with id* sid*, one of the following holds.*
– *For every possible* $\sigma \in \{0,1\}$*, with probability at least* $\frac{1}{n}$*, all the honest parties output* σ*; otherwise*
– *At least one new local-conflict occurs between an honest and a corrupt party.*

Lemma 4.5. *Protocol* Π_{SCC} *incurs a communication of* $O(|\mathcal{Z}| \cdot n^5 \log |\mathbb{K}| + n^6 \log n)$ *bits.*

Theorem 4.6. *Let* Adv *be a computationally unbounded adversary, characterized by an adversary structure* $\mathcal{Z}$*, satisfying the* $\mathbb{Q}^{(3)}(\mathcal{P},\mathcal{Z})$ *condition and where* $|\mathcal{P}| = n \geq 3$*. Then for every possible* $\mathsf{sid} \in \mathbb{N}$*, protocol* Π_{SCC} *is a* $\frac{1}{n}$*-SCC protocol, incurring a communication of* $O(|\mathcal{Z}| \cdot n^5 \log |\mathbb{K}| + n^6 \log n)$ *bits.*

5 ABA WITH $\mathbb{Q}^{(3)}(\mathcal{P},\mathcal{Z})$ CONDITION

In this section, we show how to "combine" protocols Π_{Vote} and Π_{SCC} to get the protocol Π_{ABA} (see Fig 4), by generalizing the blueprint of [7, 20, 35] against general adversaries. For an easy description of the blueprint (against *threshold* adversaries), we refer to [3, 11]. The protocol consists of several iterations, where each iteration consists of two instances of Π_{Vote} protocol and one instance of Π_{SCC}, which are carefully "stitched" together.

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

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

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

The *second* case is when the honest parties start the protocol with *different* input bits. In this case, the protocol tries to take the help of Π_{SCC} to ensure that all honest parties reach an iteration with a common input bit for that iteration. Once such an iteration is reached, this *second* case gets "transformed" to the *first* case and hence all honest parties will eventually output that common bit. In more detail, in each iteration k, it will be ensured that either every honest party have the same input bit for the second instance of Π_{Vote} with probability at least $\frac{1}{n} \cdot \frac{1}{2} = \frac{1}{2n}$ or else one new local-conflict occurs. This is because the input for second instance of Π_{Vote} is either the output bit of the first instance of Π_{Vote} or the output of Π_{SCC}, both of which are *independent* of each other. Hence if the output of Π_{SCC} is same for all the parties with probability $\frac{1}{n}$, then with probability $\frac{1}{n} \cdot \frac{1}{2}$, this bit will be the same as output bit from the first instance of Π_{Vote}. If in any iteration k, it is *guaranteed* that all honest parties have the same inputs for the second instance of Π_{Vote}, then the parties will obtain a common output and with highest grade from the second instance of Π_{Vote}. And then from the next iteration onward, all parties will stick to that common bit and eventually output that common bit.

We show that it requires $O(n^2)$ number of iterations in *expectation* before a "good" iteration is reached where it is *guaranteed* that all honest parties have the same input for the second instance of Π_{Vote}. Intuitively, this is because there can be $O(n^2)$ number of "bad" iterations in which the honest parties may have different outputs from the corresponding instances of Π_{SCC}. This follows from the fact that the corrupt parties may deviate from the protocol instructions during the instances of Π_{SCC}. There can be at most $O(n^2)$ local-conflicts which may occur *overall* during various "failed" instances of Π_{SCC} (where a failed instance means that different honest parties obtain different outputs) and only after all these local-conflicts are identified, the parties may start witnessing "clean" instances of Π_{SCC} where all honest parties shun communication from all corrupt parties and where it is ensured that all honest parties obtain the same output bit with probability $\frac{1}{n}$. Once all the bad iterations are over and all potential local-conflicts are identified, in each subsequent iteration, all honest parties will then have the same output from Π_{SCC} (and hence, same input for the second instance of Π_{Vote}) with probability at least $\frac{1}{2n}$. Consequently, it will take $\Theta(n^2)$ expected number of such iterations before the parties reach a good iteration where it is guaranteed that all honest parties have the same inputs for the second instance of Π_{Vote}.

Protocol Π_{ABA}

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

– **Initialization**: Set $b = b_i$, sid = 0, Committed = False and $k = 1$. Then do the following.

1. Set sid = sid + 1 and participate in an instance of Π_{Vote} protocol with id sid and input b.
2. Once an output (b, g) is received from the instance of Π_{Vote} with id sid, participate in an instance of Π_{SCC} with id sid. Let Coin_k denote the output received during Π_{SCC} with id sid.
3. If $g < 2$, then set $b = \mathsf{Coin}_k$.
4. Set sid = sid + 1 and participate in an instance of Π_{Vote} protocol with id sid and input b and let (b', g') be the output received. If $g' > 0$, then set $b = b'$.
5. If $g' = 2$ and Committed = False, then set Committed = True and send (ready, b) to all the parties.
6. Set $k = k + 1$ and repeat from 1.

– **Output Computation and Termination**:
  – If (ready, b) is received from a set of parties $\mathcal{R} \notin \mathcal{Z}$, then send (ready, b) to all the parties.
  – If (ready, b) is received from a set of parties $\mathcal{T}$ such that $\mathcal{P} \setminus \mathcal{T} \in \mathcal{Z}$, then output b and terminate.

Figure 4: The ABA protocol from Π_{Vote} and Π_{SCC}. The above code is executed by every $P_i \in \mathcal{P}$

The properties of Π_{ABA} are stated in the following lemmas and Theorem, which are proved in Appendix B due to space constraints.

LEMMA 5.1. *In protocol Π_{ABA}, if all honest parties have the same input bit σ, then all honest parties eventually output σ.*

LEMMA 5.2. *In Π_{ABA}, if some honest party terminates with output σ, then every other honest party eventually terminates with output σ.*

LEMMA 5.3. *In protocol Π_{ABA}, if all honest parties initiate iteration k, then one of the following holds:*

– *With probability at least $\frac{1}{2n}$, all honest parties have the same updated bit b at the end of iteration k; or*
– *At least one new local-conflict occurs between an honest and a corrupt party.*

COROLLARY 5.4. *Let $\mathcal{I}$ denote the set of iterations k in Π_{ABA}, such that all the honest parties have the same updated bit after iteration k with probability less than $\frac{1}{2n}$. Then $|\mathcal{I}| = O(n^2)$.*

LEMMA 5.5. *In protocol Π_{ABA}, if for every iteration k, all the honest parties have the same updated bit at the end of iteration k with probability at least $\frac{1}{2n}$, then the protocol requires expected $O(n^2)$ iterations to terminate.*

LEMMA 5.6. *Protocol Π_{ABA} terminates for the honest parties in $O(n^2)$ expected running time.*

THEOREM 5.7. *Let Adv be a computationally unbounded adversary, characterized by an adversary structure $\mathcal{Z}$, such that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then protocol Π_{ABA} is an almost-surely terminating ABA protocol with expected running time of $\mathsf{R} = O(n^2)$. The protocol incurs an expected communication complexity of $O(\mathsf{R} \cdot (|\mathcal{Z}| \cdot n^5 \log|\mathbb{K}| + n^6 \log n))$ bits.*

REFERENCES

[1] I. Abraham, D. Dolev, and J. Y. Halpern. 2008. An Almost-surely Terminating Polynomial Protocol for Asynchronous Byzantine Agreement with Optimal Resilience. In *PODC*. ACM, 405–414.
[2] I. Abraham, D. Dolev, and G. Stern. 2020. Revisiting Asynchronous Fault Tolerant Computation with Optimal Resilience. In *PODC*. ACM, 139–148.
[3] A. Appan, A. Chandramouli, and A. Choudhury. 2022. Perfectly-Secure Synchronous MPC with Asynchronous Fallback Guarantees. *To appear in PODC. IACR Cryptol. ePrint Arch.* (2022), 109.
[4] Hagit Attiya and Jennifer Welch. 2004. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Vol. 19. John Wiley & Sons.
[5] L. Bangalore, A. Choudhury, and A. Patra. 2020. The Power of Shunning: Efficient Asynchronous Byzantine Agreement Revisited. *J. ACM* 67, 3 (2020), 14:1–14:59.
[6] Z. Beerliová-Trubíniová and M. Hirt. 2007. Simple and Efficient Perfectly-Secure Asynchronous MPC. In *ASIACRYPT (LNCS, Vol. 4833)*. Springer Verlag, 376–392.
[7] M. Ben-Or. 1983. Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols. In *PODC*. ACM, 27–30.
[8] M. Ben-Or, R. Canetti, and O. Goldreich. 1993. Asynchronous Secure Computation. In *STOC*. ACM, 52–61.
[9] M. Ben-Or, S. Goldwasser, and A. Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*. ACM, 1–10.
[10] M. Ben-Or, B. Kelmer, and T. Rabin. 1994. Asynchronous Secure Computations with Optimal Resilience (Extended Abstract). In *PODC*. ACM, 183–192.
[11] E. Blum, J. Katz, and J. Loss. 2019. Synchronous Consensus with Optimal Asynchronous Fallback Guarantees. In *TCC (LNCS, Vol. 11891)*. Springer, 131–150.
[12] G. Bracha. 1984. An Asynchronous [(n-1)/3]-Resilient Consensus Protocol. In *PODC*. ACM, 154–162.
[13] R. Canetti. 1995. *Studies in Secure Multiparty Computation and Applications*. Ph.D. Dissertation. Weizmann Institute, Israel.
[14] R. Canetti and T. Rabin. 1993. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *STOC*. 42–51.
[15] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. 1985. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *FOCS*. IEEE Computer Society, 383–395.
[16] A. Choudhury. 2020. Improving the Efficiency of Optimally-Resilient Statistically-Secure Asynchronous Multi-party Computation. In *INDOCRYPT (LNCS, Vol. 12578)*. Springer, 810–831.
[17] A. Choudhury and N. Pappu. 2020. Perfectly-Secure Asynchronous MPC for General Adversaries. In *INDOCRYPT (LNCS, Vol. 12578)*. Springer, 786–809.
[18] A. Choudhury and A. Patra. 2017. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Trans. Information Theory* 63, 1 (2017), 428–468.
[19] R. Cramer, I. Damgård, and U. M. Maurer. 2000. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In *EUROCRYPT (LNCS, Vol. 1807)*. Springer Verlag, 316–334.
[20] Paul Feldman and Silvio Micali. 1988. Optimal Algorithms for Byzantine Agreement. In *STOC*. ACM, 148–161.
[21] M. J. Fischer, N. A. Lynch, and M. Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382.
[22] Matthias Fitzi. 2002. *Generalized Communication and Security Models in Byzantine Agreement*. Ph.D. Dissertation.
[23] M. Fitzi and U. M. Maurer. 1998. Efficient Byzantine Agreement Secure Against General Adversaries. In *DISC (LNCS, Vol. 1499)*. Springer, 134–148.
[24] J. A. Garay and A. Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *CT-RSA (LNCS, Vol. 12006)*. Springer, 284–318.
[25] Martin Hirt. 2001. *Multi-Party Computation: Efficient Protocols, General Adversaries, and Voting*. Ph.D. Dissertation. ETH Zurich.
[26] Martin Hirt and Ueli Maurer. 1997. Complete Characterization of Adversaries Tolerable in Secure Multi-Party Computation. In *PODC*. ACM, 25–34.
[27] M. Hirt and U. Maurer. 2000. Player Simulation and General Adversary Structures in Perfect Multiparty Computation. *Journal of Cryptology* 13, 1 (2000), 31–60.
[28] K. Kursawe. 2001. *Distributed Trust*. Ph.D. Dissertation. Saarland University.
[29] K. Kursawe and F. C. Freiling. 2005. Byzantine Fault Tolerance on General Hybrid Adversary Structures. Technical Report, RWTH Aachen.
[30] Nancy A Lynch. 1996. *Distributed algorithms*. Morgan Kaufmann.
[31] R. Shostak M. Pease and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *Journal of the ACM (JACM)* 27, 2 (1980), 228–234.
[32] U. M. Maurer. 2002. Secure Multi-party Computation Made Simple. In *SCN (LNCS, Vol. 2576)*. Springer, 14–28.
[33] A. Patra, A. Choudhury, and C. Pandu Rangan. 2014. Asynchronous Byzantine Agreement with Optimal Resilience. *Distributed Computing* 27, 2 (2014), 111–146.
[34] A. Patra, A. Choudhury, and C. Pandu Rangan. 2015. Efficient Asynchronous Verifiable Secret Sharing and Multiparty Computation. *J. Cryptology* 28, 1 (2015), 49–109.
[35] Michael O. Rabin. 1983. Randomized Byzantine Generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November*

1983. 403–409.
[36] T. Rabin and M. Ben-Or. 1989. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *STOC*. ACM, 73–85.
[37] C. Wang. 2015. Asynchronous Byzantine Agreement with Optimal Resilience and Linear Complexity. *CoRR* abs/1507.06165 (2015).

A PROPERTIES OF THE PROTOCOL Π_{SCC}

In this section, we prove the properties of the protocol Π_{SCC}.

Lemma 4.1. *If each honest party participates in* Π_{SCC} *with id* sid*, then each honest party eventually computes an output.*

Proof. We begin by showing that if all *honest* parties participate in the protocol Π_{SCC}, then each *honest* party P_i eventually sets $\mathsf{Flag}_i = 1$. Firstly, each *honest* party P_j invokes its n instances of Π_{Sh} as a dealer. These instances are guaranteed to eventually produce an output for each *honest* party, which follows from Lemma 3.2. Hence, P_i eventually finds a set of accepted-dealers AD_i such that $\mathcal{P} \setminus \mathsf{AD}_i \in \mathcal{Z}$, as the set of honest parties constitute a potential AD_i set. This implies that P_i eventually broadcasts a message $(\mathtt{Attach}, \mathsf{AD}_i, P_i)$. Consequently, P_i eventually receives the message $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ from the broadcast of every *honest* party P_j. Also, $\mathsf{AD}_j \subseteq \mathcal{AD}_i$ eventually holds, which again follows from Lemma 3.2. Consequently, P_i eventually broadcasts a message $\mathsf{OK}(P_i, P_j)$ for every honest party P_j. Therefore, P_i eventually includes every honest party in the set of accepted-parties $\mathcal{AP}_i$, and eventually $\mathcal{P} \setminus \mathcal{AP}_i \in \mathcal{Z}$ holds. Hence, P_i eventually broadcasts a message $(\mathsf{ready}, P_i, \mathsf{AP}_i)$. Consequently, P_i eventually receives the message $(\mathsf{ready}, P_j, \mathsf{AP}_j)$ from the broadcast of every other honest party P_j. Also, $\mathsf{AP}_j \subseteq \mathcal{AP}_i$ eventually holds, which follows from Lemma 3.2. Therefore, P_i's set of supportive parties $\mathcal{SP}_i$ eventually satisfies the condition $\mathcal{P} \setminus \mathcal{SP}_i \in \mathcal{Z}$, after which P_i sets $\mathsf{Flag}_i = 1$.

We now show that if any *honest* P_i sets $\mathsf{Flag}_i = 1$, then P_i eventually computes an output in Π_{SCC}. For this to be true, party P_i needs to be able to compute the values Coin_k attached with all the parties $P_k \in \mathsf{FS}_i$. Party P_i can compute Coin_k, if the Π_{Rec} instances $\Pi_{\mathsf{Rec}}^{(jk)}$ produce some output for P_i, corresponding to each $P_j \in \mathsf{AD}_k$. We show that the Π_{Rec} instances $\Pi_{\mathsf{Rec}}^{(jk)}$ eventually produce an output. This follows from the properties of Π_{Rec} (Lemma 3.2) and the fact that every party $P_k \in \mathsf{FS}_i$ is eventually accepted or partially-accepted by every other honest party P_ℓ (i.e., $P_k \in \mathcal{AP}_\ell \cup \mathcal{PAP}_\ell$) and consequently $\Pi_{\mathsf{Rec}}^{(jk)}$ is eventually initiated by every honest party.

Hence, if each honest party participates in protocol Π_{SCC}, then each honest party eventually computes an output in Π_{SCC}. ◻

We next prove the properties of the Coin values, attached with various parties.

Lemma 4.2. *During* Π_{SCC} *with id* sid*, if any honest party receives the message* $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$ *from the broadcast of any party* P_k*, then a unique value* Coin_k *is fixed such that all the following hold:*

- *The coin* Coin_k *is attached to* P_k*.*
- *The value* Coin_k *is distributed uniformly over* $\{0, \ldots, n-1\}$ *and is independent of the coins attached to the other parties.*
- *If any honest party associates* $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$ *to* P_k*, then at least one new local-conflict occurs between an honest and a corrupt party.*

Proof. Let P_i be an *honest* party, who receives the message $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$ from the broadcast of P_k. From the properties of broadcast, every honest party eventually receives the same message $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$ from the broadcast of P_k. Let $\{s_{jk}\}_{P_j \in \mathsf{AD}_k}$ denote the set of values shared by $P_j \in \mathsf{AD}_k$ (as a dealer), on the behalf of P_k, during the instance $\Pi_{\mathsf{Sh}}^{(jk)}$. We define

$$\mathsf{Coin}_k \overset{def}{=} \sum_{P_j \in \mathsf{AD}_k} s_{jk} \mod n.$$

Since all the honest parties receive the same set AD_k, the value of Coin_k will be common from the point of view of all honest parties.

For the second property, we note that the parties start executing the instances $\{\Pi_{\mathsf{Rec}}^{(jk)}\}_{P_j \in \mathsf{AD}_k}$ only after receiving the broadcasted message $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$. This implies that the set AD_k is fixed, before any instance in $\{\Pi_{\mathsf{Rec}}^{(jk)}\}_{P_j \in \mathsf{AD}_k}$ is invoked. The set AD_k consists of at least one *honest* party, say P_j, as $\mathcal{P} \setminus \mathsf{AD}_k \in \mathcal{Z}$ and $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. The privacy property of Π_{Sh} (see Lemma 3.4) ensures that the view of the adversary during the instance of $\Pi_{\mathsf{Sh}}^{(jk)}$ is independent of the secret s_{jk} shared by P_j. Now since the secrets shared by the *honest* parties during Π_{SCC} are mutually independent and uniformly selected from $\mathbb{K}$, it follows that Coin_k is uniformly and independently distributed over $\{0, \ldots, n-1\}$.

For the third property, let P_h be an *honest* party, who associates Coin'_k to P_k, such that $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$. From the protocol steps, P_h associates Coin'_k by participating in the instances $\Pi_{\mathsf{Rec}}^{(jk)}$ and computing the values $\{r_{jk}\}_{P_j \in \mathsf{AD}_k}$. Since $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$, it follows that $\{s_{jk}\}_{P_j \in \mathsf{AD}_k} \neq \{r_{jk}\}_{P_j \in \mathsf{AD}_k}$. Hence, there exists *at least one* $P_j \in \mathsf{AD}_k$, such that the value s_{jk} during the instance $\Pi_{\mathsf{Sh}}^{(jk)}$ is *different* from the value r_{jk} computed during the instance $\Pi_{\mathsf{Rec}}^{(jk)}$. The proof now follows from Lemma 3.3. ◻

We next prove the crucial non-empty overlap property among the FS of all the honest parties, which will further lead to a *non-zero* success probability for the protocol Π_{SCC}.

Lemma 4.3. *In protocol* Π_{SCC} *with id* sid*, once some honest party sets its* Flag *to 1, then there exists a set, say* $\mathcal{M}$*, such that all the following hold:*

- $\mathcal{P} \setminus \mathcal{M} \in \mathcal{Z}$*.*
- *For each* $P_j \in \mathcal{M}$*, some honest party receives the message* $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ *from the broadcast of* P_j*.*
- *Whenever any honest party* P_i *sets its* $\mathsf{Flag}_i = 1$*, the condition* $\mathcal{M} \subseteq \mathsf{FS}_i$ *holds.*

Proof. Let P_f be the *first honest* party who broadcasts a ready message (from the proof of Lemma 4.1, such a party P_f exists). This implies that P_f finds a set of accepted parties AP_f (and probably, an additional set of parties $\mathcal{PAP}_f$), such that $\mathcal{P} \setminus \mathsf{AP}_f \in \mathcal{Z}$. Moreover, for each $P_j \in \mathsf{AP}_f$, party P_f receives $\mathsf{OK}(\star, P_j)$ messages from a set of parties S_j, including P_f, such that $\mathcal{P} \setminus S_j \in \mathcal{Z}$. We define $\mathcal{M} \overset{def}{=} \mathsf{AP}_f$ and show that AP_f satisfies all the properties of $\mathcal{M}$, as stated in the lemma.

The first property holds as $\mathcal{P} \setminus \mathsf{AP}_f \in \mathcal{Z}$ holds, before any *honest* party P_i sets $\mathsf{Flag}_i = 1$. This follows from the fact that for an *honest*

P_i to set $\mathsf{Flag}_i = 1$, it must receive a ready message from at least *one honest* party, and P_f is assumed to be the *first* honest party that broadcasts a ready message. And P_f broadcasts the ready message only when $\mathcal{P} \setminus \mathsf{AP}_f \in \mathcal{Z}$ holds.

We now consider the second property. For each $P_j \in \mathsf{AP}_f$, the messages $\mathsf{OK}(\star, P_j)$ are received from the broadcasts of parties in the set S_j, which also includes P_f. Now P_f broadcasts $\mathsf{OK}(P_f, P_j)$ only after receiving $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ from the broadcast of P_j. Since P_f is assumed to be an *honest* party, the second property follows.

We now consider the third property. Let P_i be an arbitrary *honest* party who sets Flag_i to 1. We want to show that $\mathsf{AP}_f \subseteq \mathsf{FS}_i$ holds. For this, consider an arbitrary party $P_j \in \mathsf{AP}_f$; we show that P_j belongs to FS_i as well. Since $P_j \in \mathsf{AP}_f$, it implies that P_f has received the messages $\mathsf{OK}(\star, P_j)$ from a set of parties S_j, such that $\mathcal{P} \setminus S_j \in \mathcal{Z}$ holds. We also note that from the definition of SP_i, the condition $\mathcal{P} \setminus \mathsf{SP}_i \in \mathcal{Z}$ holds. Since $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, it follows that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(1)}(S_j \cap \mathsf{SP}_i, \mathcal{Z})$ condition and hence there exists at least one *honest* party, say P_h, such that $P_h \in S_j \cap \mathsf{SP}_i$ holds. Now $P_h \in S_j$ implies that P_h must have received the message $(\mathtt{Attach}, \mathsf{AD}_j, P_j)$ from the broadcast of P_j and after verification, have broadcasted an $\mathsf{OK}(P_h, P_j)$ message and included P_j to $\mathcal{PAP}_h$. Moreover, since P_f is assumed to be the *first* honest party to have broadcasted a ready message, it follows that P_j will be present in $\mathcal{PAP}_h \cup \mathsf{AP}_h$, when P_h broadcasts the message $(\text{ready}, P_h, \mathsf{AP}_h, \mathcal{PAP}_h)$. Since $P_h \in \mathsf{SP}_i$, we have $(\mathsf{AP}_h \cup \mathcal{PAP}_h) \subseteq \mathsf{FS}_i$. Consequently, $P_j \in \mathsf{FS}_i$. This proves the third property. ◻

Lemma 4.4. *In protocol Π_{SCC} with id* sid*, one of the following holds.*

- *For every possible $\sigma \in \{0, 1\}$, with probability at least $\frac{1}{n}$, all the honest parties output σ; otherwise*
- *At least one new local-conflict occurs between an honest and a corrupt party.*

Proof. Let P_i be an arbitrary *honest* party. In the protocol, P_i sets its output bit based on the values *associated* with the parties in FS_i. Moreover, for every $P_k \in \mathsf{FS}_i$, party P_i receives the message $(\mathtt{Attach}, \mathsf{AD}_k, P_k)$ from the broadcast of P_k. This further guarantees that a uniformly random and independently distributed value $\mathsf{Coin}_k \in \{0, \ldots, n-1\}$ is fixed, which is *attached* to P_k (Lemma 4.2). Moreover, let Coin'_k be the value, *associated* to P_k by P_i. Furthermore, let $\mathcal{M}$ be the set of parties as discussed in Lemma 4.3. From the same lemma, it holds that $\mathcal{M} \subseteq \mathsf{FS}_i$. Now there are two possible cases.

1. **Case I: For every $P_k \in \mathsf{FS}_i$, party P_i associates $\mathsf{Coin}'_k = \mathsf{Coin}_k$ to P_k.** If $\mathsf{Coin}_k = 0$ holds for some $P_k \in \mathcal{M}$, then party P_i outputs 0 in Π_{SCC}. The probability that for at least one party $P_k \in \mathcal{M}$, the attached value $\mathsf{Coin}_k = 0$ is $1 - (1 - \frac{1}{n})^{|\mathcal{M}|} \geq \frac{1}{n}$.
   On the other hand, if $\mathsf{Coin}_k \neq 0$ for every party $P_k \in \mathsf{FS}_i$, then P_i outputs $\sigma = 1$. The probability of this event is at least $(1 - \frac{1}{n})^n \geq e^{-1} \geq 0.36$. And for any $n \geq 3$, we have $0.36 > \frac{1}{n}$. This proves the first part of the lemma.
2. **Case II: There exists at least one $P_k \in \mathsf{FS}_i$, such that party P_i associates $\mathsf{Coin}'_k \neq \mathsf{Coin}_k$ to P_k.** In this case, from the third part of Lemma 4.2, at least one new local-conflict occurs. This proves the second part of the lemma.

◻

Lemma 4.5. *Protocol Π_{SCC} incurs a communication of $O(|\mathcal{Z}| \cdot n^5 \log |\mathbb{K}| + n^6 \log n)$ bits.*

Proof. The proof simply follows from the communication complexity of Π_{Sh} and Π_{Rec} (Lemma 3.5) and the fact that there are $O(n^2)$ instances of Π_{Sh} and Π_{Rec} involved in the protocol. ◻

The proof of Theorem 4.6 now simply follows from Lemma 4.1-4.5.

Theorem 4.6. *Let* Adv *be a computationally unbounded adversary, characterized by an adversary structure $\mathcal{Z}$, satisfying the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition and where $|\mathcal{P}| = n \geq 3$. Then for every possible* $\mathsf{sid} \in \mathbb{N}$, *protocol Π_{SCC} is a $\frac{1}{n}$-SCC protocol, incurring a communication of $O(|\mathcal{Z}| \cdot n^5 \log |\mathbb{K}| + n^6 \log n)$ bits.*

B PROPERTIES OF THE PROTOCOL Π_{ABA}

In this section, we prove the properties of the protocol Π_{ABA}. We first start with the proof of the validity property.

Lemma 5.1. *In protocol Π_{ABA}, if all honest parties have the same input bit σ, then all honest parties eventually output σ.*

Proof. Let $Z_c \in \mathcal{Z}$ be the set of corrupt parties. If every honest party has the same input bit σ, then from the properties of the protocol Π_{Vote}, all honest parties eventually output $(b, g) = (\sigma, 2)$ at the end of the first as well as second instance of the Π_{Vote} protocol during the first iteration. Consequently, every *honest* party eventually sends a (ready, σ) message to all the parties and only the parties in Z_c may send a $(\text{ready}, \overline{\sigma})$ message. It now follows easily from the steps of the output computation stage that *no* honest party ever sends a $(\text{ready}, \overline{\sigma})$ message and all honest parties eventually output σ. ◻

We next prove the agreement property.

Lemma 5.2. *In protocol Π_{ABA}, if some honest party terminates with output bit σ, then every other honest party eventually terminates with output σ.*

Proof. We first show that if any *honest* party broadcasts a (ready, σ) message for any $\sigma \in \{0, 1\}$ during any iteration k, then no honest party broadcasts a $(\text{ready}, \overline{\sigma})$ message during iteration k or in the subsequent iterations. For this, let P_i be an *honest* party who broadcasts a (ready, σ) message during iteration k. This implies that P_i outputs $(b, g) = (\sigma, 2)$ in the second instance of the Π_{Vote} protocol during iteration k and sets Committed to True. Then, from the properties of the protocol Π_{Vote}, every other honest party outputs either $(\sigma, 2)$ or $(\sigma, 1)$ in the second instance of the Π_{Vote} protocol during iteration k. Consequently, no other honest party broadcasts the $(\text{ready}, \overline{\sigma})$ message during iteration k. Also, from the protocol steps, all honest parties update their input to σ for the next iteration. This further implies that all honest parties will continue to input σ to each subsequent invocations of Π_{Vote}, ignoring the output of Π_{SCC}, for as long as they continue running. Consequently, no honest party ever sends a $(\mathsf{ready}, \overline{\sigma})$ message.

Now let some *honest* party, say P_h, terminates Π_{ABA} with output σ during iteration k. This implies that P_h receives the (ready, σ) message from a set of parties $\mathcal{T}$, such that $\mathcal{P} \setminus \mathcal{T} \in \mathcal{Z}$. Let $Z_c \in \mathcal{Z}$ be the set of *corrupt* parties. Since all the parties in $\mathcal{T} \setminus Z_c$ are *honest*, the (ready, σ) messages of all the parties in $\mathcal{T} \setminus Z_c$ are eventually delivered to every honest party, during iteration k. Moreover, as shown above, no honest party ever broadcasts a $(\mathsf{ready}, \overline{\sigma})$ message. Furthermore, since $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition, it follows that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(1)}(\mathcal{T} \setminus Z_c, \mathcal{Z})$ condition and consequently $\mathcal{T} \setminus Z_c \notin \mathcal{Z}$. Now based on all these, we conclude that every honest party (including P_h) eventually broadcasts a (ready, σ) message during iteration k, which are eventually delivered to every honest party. Consequently, every honest party eventually receives sufficiently many number of (ready, σ) messages and terminates with output σ. □

We next prove that at the end of each iteration, the updated value of all honest parties will be the same with probability at least $\frac{1}{2n}$, or at least one *new* local-conflict occurs.

Lemma 5.3. *In protocol Π_{ABA}, if all honest parties initiate iteration k, then one of the following holds:*

- *With probability at least $\frac{1}{2n}$, all honest parties have the same updated bit b at the end of iteration k; or*
- *At least one new local-conflict occurs between an honest and a corrupt party.*

Proof. We first note that if P_i is *honest* and if at all any local conflict (P_i, P_j) occurs during iteration k of Π_{ABA}, then the conflict is different from any local conflict of the form $(P_i, \star)$, which could have occurred during any iteration k' of Π_{ABA}, where $k' < k$. On contrary, let the local conflict (P_i, P_j) occurs both during iteration k' as well as k. Since the conflict occurs during the iteration k', it follows that during the instance of Π_{SCC} with id k', party P_i includes P_j to the set $\mathcal{B}_i$ as part of one of the underlying memory management protocols. As a result, any communication from party P_j is completely ignored by P_i in any instance of Π_{SCC} protocol with id k, where $k > k'$. Consequently, the local conflict (P_i, P_j) *does not* re-occur during iteration k, which is a contradiction.

Now to prove the lemma statement, we consider an event Agree, which denotes that all honest parties have the same input for the second instance of Π_{Vote} during iteration k. If the event Agree occurs, then from the properties of Π_{Vote}, all honest parties will have the same updated bit at the end of iteration k. We show that either the event Agree occurs during iteration k with probability at least $\frac{1}{2n}$, or else at least one new local-conflict occurs. For this, we consider two different possible cases with respect to the output from the first instance of Π_{Vote} during iteration k.

- *Case I: No honest party obtains an output $(b, 2)$ for any $b \in \{0, 1\}$ during the first instance of* Π_{Vote}. In this case, all honest parties set the output from the instance of Π_{SCC} with id k as the input for the second instance of Π_{Vote}. From Lemma 4.4, either all honest parties will have the same output bit from the instance of Π_{SCC} with probability at least $\frac{1}{n} > \frac{1}{2n}$, or at least one new local-conflict occurs.
- *Case II: Some honest party obtains an output $(b, 2)$ during the first instance of* Π_{Vote}. In this case, the properties of Π_{Vote} ensure that all honest parties obtain the output $(b, 2)$ or $(b, 1)$ from the first instance of Π_{Vote}. Moreover, from the protocol steps, the output of the instance of Π_{SCC} with id k is *not* revealed, until the first honest party generates an output from the first instance of Π_{Vote} during iteration k. Consequently, the output bit b from the first instance of Π_{Vote} is *independent* of the output of Π_{SCC}. From Lemma 4.4, either all honest parties will have the same output bit from the instance of Π_{SCC} with probability at least $\frac{1}{n}$, or at least one new local-conflict occurs. If all honest parties have the same output Coin_k from the instance of Π_{SCC} with probability at least $\frac{1}{n}$, then the probability that $\mathsf{Coin}_k = b$ holds is at least $\frac{1}{2} \cdot \frac{1}{n} = \frac{1}{2n}$ and all honest parties will have the same input for the second instance of Π_{Vote}.

□

As a corollary of Lemma 5.3, we can conclude that there can be $O(n^2)$ iterations where the honest parties have the same updated value at the end with probability *strictly less* than $\frac{1}{2n}$. This is because there are $O(n^2)$ *different* local-conflicts which can occur throughout Π_{ABA}. From Lemma 5.3, if the honest parties do not have the same updated value at the end of an iteration with probability $\frac{1}{2n}$ or more, then at least one *new* local-conflict occurs. Hence there can be $O(n^2)$ such iterations.

Corollary 5.4. *Let $\mathcal{I}$ denote the set of iterations k in Π_{ABA}, such that all the honest parties have the same updated bit after iteration k with probability less than $\frac{1}{2n}$. Then $|\mathcal{I}| = O(n^2)$.*

We next derive the expected number of iterations required in the protocol Π_{ABA} for the honest parties to produce an output. We begin with the simpler case, where we assume that at the end of *each* iteration of Π_{ABA}, all honest parties have the *same* updated modified bit. Later, we will derive the expected number of iterations required when this is *not* the case.

Lemma 5.5. *In protocol Π_{ABA}, if for every iteration k, all the honest parties have the same updated bit at the end of iteration k with probability at least $\frac{1}{2n}$, then the protocol requires expected $O(n^2)$ iterations to terminate.*

Proof. In order to prove the lemma, we need to derive the expected number of iterations, until all the honest parties have the same input during the second instance of Π_{Vote} of an iteration. This is because once all the honest parties have the same input during the second instance of Π_{Vote} of an iteration, then all honest parties will set Committed to True at the end of that iteration and start broadcasting a ready message, followed by terminating the protocol. Let τ be the random variable which counts the number of iterations until all honest parties have the same input during the second instance of Π_{Vote} in an iteration. Then the probability that

$\tau = k$ is given as:

$$\begin{aligned}\Pr(\tau = k) = \Pr(\tau \neq 1) \cdot \Pr(\tau \neq 2 \mid \tau \neq 1) \cdot \ldots \cdot \\ \Pr(\tau \neq (k-1) \mid \tau \neq 1 \cap \ldots \cap \tau \neq (k-2)) \\ \cdot \Pr(\tau = k \mid \tau \neq 1 \cap \ldots \cap \tau \neq (k-1)).\end{aligned}$$

Now as per the lemma condition, every multiplicand on the right hand side in the above equation, except the last one, is upper bounded by $(1 - \frac{1}{2n})$ and the last multiplicand is upper bounded by $\frac{1}{2n}$. Hence, we get

$$\Pr(\tau = k) \leq (1 - \frac{1}{2n})^{k-1}(\frac{1}{2n}).$$

Now the expected value $E(\tau)$ of τ is computed as follows:

$$\begin{aligned}E(\tau) &= \sum_{k=0}^{\infty} \tau \cdot \Pr(\tau = k) \\ &\leq \sum_{k=0}^{\infty} k(1 - \frac{1}{2n})^{k-1}(\frac{1}{2n}) \\ &= \frac{1}{2n} \sum_{k=0}^{\infty} k(1 - \frac{1}{2n})^{k-1} \\ &= \frac{1}{1 - (1 - \frac{1}{2n})} + \frac{1 - \frac{1}{2n}}{\left(1 - (1 - \frac{1}{2n})\right)^2} \\ &= 2n + 4n^2 - 2n = 4n^2\end{aligned}$$

The expression for $E(\tau)$ is a sum of AGP up to infinite terms, which is given by $\frac{a}{1-r} + \frac{dr}{(1-r)^2}$, where $a = 1$, $r = 1 - \frac{1}{2n}$ and $d = 1$. Hence, we have $E(\tau) \leq 4n^2$. □

We next derive the expected number of iterations required in the protocol Π_{ABA}. This automatically gives the expected running time of Π_{ABA}, as each iteration in Π_{ABA} requires a constant time.

Lemma 5.6. *Protocol Π_{ABA} terminates for the honest parties in $O(n^2)$ expected running time.*

Proof. From Corollary 5.4, there can be $O(n^2)$ iterations in Π_{ABA}, where at the end of the iteration, the updated bits of the honest parties are different with probability more than $\frac{1}{2n}$. After this, in each iteration of Π_{ABA}, the honest parties will have the same updated bit at the end of iteration, except with probability at most $\frac{1}{2n}$ and as a result, Π_{ABA} will require expected $O(n^2)$ iterations to terminate (follows from Lemma 5.5). As each iterations in Π_{ABA} requires a constant time, it follows that Π_{ABA} terminates for the honest parties in $O(n^2)$ expected running time. □

The proof of Theorem 5.7 now follows from Lemmas 5.1-5.6. The communication complexity follows from the fact that two instances of Π_{Vote} and one instance of Π_{SCC} is executed in each iteration in Π_{ABA} and the expected number of such iterations is $O(n^2)$.

Theorem 5.7. *Let* Adv *be a computationally unbounded adversary, characterized by an adversary structure $\mathcal{Z}$, such that $\mathcal{Z}$ satisfies the $\mathbb{Q}^{(3)}(\mathcal{P}, \mathcal{Z})$ condition. Then protocol Π_{ABA} is an almost-surely terminating ABA protocol with expected running time of $\mathsf{R} = O(n^2)$. The protocol incurs an expected communication complexity of $O(\mathsf{R} \cdot (|\mathcal{Z}| \cdot n^5 \log|\mathbb{K}| + n^6 \log n))$ bits.*