Synchronous Consensus with Optimal Asynchronous Fallback Guarantees

Erica Blum[1], Jonathan Katz[2⋆], and Julian Loss[1,3⋆⋆]

[1] Dept. of Computer Science, University of Maryland, erblum@cs.umd.edu
[2] Dept. of Computer Science, George Mason University, jkatz2@gmail.com
[3] Ruhr University Bochum, julian.loss@rub.de

Abstract. Typically, protocols for Byzantine agreement (BA) are designed to run in either a *synchronous* network (where all messages are guaranteed to be delivered within some known time Δ from when they are sent) or an *asynchronous* network (where messages may be arbitrarily delayed). Protocols designed for synchronous networks are generally insecure if the network in which they run does not ensure synchrony; protocols designed for asynchronous networks are (of course) secure in a synchronous setting as well, but in that case tolerate a lower fraction of faults than would have been possible if synchrony had been assumed from the start.

Fix some number of parties n, and $0 < t_a < n/3 \leq t_s < n/2$. We ask whether it is possible (given a public-key infrastructure) to design a BA protocol that is resilient to (1) t_s corruptions when run in a synchronous network and (2) t_a faults even if the network happens to be asynchronous. We show matching feasibility and infeasibility results demonstrating that this is possible if and only if $t_a + 2 \cdot t_s < n$.

1 Introduction

Byzantine agreement (BA) [24, 35] is a classical problem in distributed computing. Roughly speaking, a BA protocol allows a group of n parties, each holding some initial input value, to agree on their outputs even in the presence of some threshold of corrupted parties. Such protocols are used widely in practice for ensuring consistency among a set of distributed processors [6,21,23,30], and have received renewed interest in the context of blockchain protocols. They also serve as a core building block for more complicated protocols, e.g., for secure multiparty computation. There is an extensive literature on Byzantine agreement, and many different models in which it can be studied. We focus here on the setting in which a *public-key infrastructure* (PKI) is available.

⋆ Portions of this work were done while at the University of Maryland.

⋆⋆ Work was done while at Ruhr University Bochum.

Typically, protocols for Byzantine agreement are designed and analyzed assuming either a *synchronous network*, where messages are guaranteed to be delivered within some known time bound Δ, or an *asynchronous network*, where messages can be delayed arbitrarily. Existing results precisely characterize when the problem can be solved in each case [5, 8, 10, 24, 35]: in a synchronous network, it is possible if and only if $t_s < n/2$ parties are corrupted, while in an asynchronous network it can be achieved only when there are $t_a < n/3$ corruptions. In each case, protocols tolerating the optimal threshold and running in expected *constant* rounds are known [5, 20].

In real-world deployments of Byzantine agreement, the network conditions in which a protocol are run may be unclear; for example, the network may generally be synchronous but intermittently experience congestion that prevents messages from being delivered in a timely fashion. This results in the following dilemma when deciding what protocol to use:

– Protocols designed for a synchronous network are, in general, insecure if the assumption of network synchrony fails.
– Protocols designed for an asynchronous network will (of course) be secure when the network is synchronous. But in this case the fraction of faults that can be tolerated is *lower* than what could have been tolerated if the protocol were designed for the synchronous setting.

Fix some thresholds t_a, t_s with $0 < t_a < n/3 \leq t_s < n/2$. We ask the following question: is it possible to design a BA protocol that is (1) resilient to any t_s (adaptive) corruptions when run in a synchronous network and also (2) resilient to t_a (adaptive) corruptions even if the network happens to be asynchronous? We completely resolve this question by showing matching feasibility and infeasibility results demonstrating that this is possible if and only if $t_a + 2 \cdot t_s < n$.

Positive result. The protocol achieving our positive result is constructed by combining two sub-protocols $\Pi_{\mathsf{SBA}}, \Pi_{\mathsf{ABA}}$ for Byzantine agreement, where Π_{SBA} is secure in a synchronous network and Π_{ABA} is secure in an asynchronous network. The key to our analysis is to separately analyze the validity, consistency, and liveness guarantees of these sub-protocols. Specifically, we design Π_{SBA} so that it also satisfies a certain validity guarantee *even when run in an asynchronous network*. We also design Π_{ABA} so that it achieves validity (in an asynchronous network) *even beyond* $n/3$ *corruptions*. We then use these properties to prove security of our main protocol, for different thresholds, when run in either a synchronous or asynchronous network.

Impossibility result. We also show that our positive result is *tight*, namely, that if $t_a + 2 \cdot t_s \geq n$ then there is no protocol that is simultaneously resilient to t_s corruptions when run in a synchronous network and also resilient to t_a faults in an asynchronous network. In fact, we show a result that is slightly stronger: it is not possible to achieve validity for t_s static faults in the synchronous setting while also achieving a weak notion of consistency for t_a static faults in an asynchronous network.

1.1 Related Work

The question of designing protocols that remain secure when run in various network conditions is natural, and so it is somewhat surprising that it has only recently begun to draw attention in the literature. Recent work by Malkhi et al. [28] is most closely related to our own. Among other things, they consider protocols with certain guarantees when run in synchronous or *partially synchronous* networks. In contrast, we consider the case of synchronous or *fully asynchronous* networks. Liu et al. [25] design a protocol that is resilient to a minority of malicious corruptions in a synchronous network, and a minority of *fail-stop* faults in an asynchronous network. Our work can be viewed as extending theirs to consider malicious corruptions in both settings. Guo, Pass, and Shi [16] consider a model motivated by eclipse attacks [18] on blockchain protocols, whereby an attacker temporarily disconnects some subset S of honest parties from the rest of the network S', e.g., by delaying or dropping messages between S and S'. Parties in S may not be able to reach agreement with honest parties in S'; nevertheless, as observed by Guo et al., it may be possible to provide certain guarantees for the parties in S' if their network is well-behaved (i.e., synchrony continues to hold for messages sent between parties in S'). Guo et al. gave BA protocols tolerating the optimal corruption threshold in this model, and Abraham et al. [2] extended their work to achieve similar guarantees for state-machine replication. The main difference between these works and ours is that they continue to assume synchrony in part of the network, and their protocols fail completely if the all communication channels in the network may be asynchronous.

Kursawe [22] shows a protocol for asynchronous BA that reaches agreement more quickly in case the network is synchronous. In contrast to our work, that protocol does not achieve better fault tolerance (and, in particular, cannot tolerate $n/3$ or more faults) in the synchronous case.

Other recent work has looked at designing protocols for synchronous BA that achieve good *responsiveness* when the network latency is low. That is, these protocols ensure that if the actual message-delivery time is

$\delta < \Delta$ then the time to reach agreement is proportional to δ rather than the upper bound Δ. This problem was considered by Pass and Shi [31,32], who gave protocols that rely on a leader and are therefore not adaptively secure, as well as by Loss and Moran [27], who avoid the use of a leader. The work of Loss and Moran was extended by Liu-Zhang et al. [26] to the case of general secure computation. None of these works provides security in case the synchrony assumption fails altogether.

Several prior works [3, 7, 12, 34] consider a model in which synchrony is assumed to be available for some (known) limited period of time, and asynchronous afterward. Fitzi et al. [11] and Loss and Moran [27] study trade-offs between the validity, consistency, and liveness properties of BA that inspired our asynchronous BA protocol in Section 4 and our lower bound in Section 6.

1.2 Paper Organization

We introduce our model as well as definitions for Byzantine agreement and related tasks in Section 2. In Sections 3 and 4 we describe two protocols for Byzantine agreement and prove various properties about them. Those protocols are used, in turn, as sub-protocols of our main protocol in Section 5 that achieves security (for different thresholds) in both synchronous and asynchronous networks. Finally, in Section 6 we show that the bounds we achieve are tight.

2 Model and Definitions

Throughout, we consider a network of n parties $P_1, \ldots, P_n$ who may communicate over point-to-point authenticated channels. We also assume that the parties have established a public-key infrastructure in advance of the protocol execution. This means that all parties hold the same vector $(pk_1, \ldots, pk_n)$ of public keys for a digital signature scheme, where each honest party P_i holds the honestly generated secret key sk_i associated with pk_i. (Malicious parties may choose their keys arbitrarily.) A *valid* signature σ on m from P_i is one for which $\mathsf{Verify}_{pk_i}(m, \sigma) = 1$. We make the standard convention of treating signatures as idealized objects; i.e., throughout our analysis, signatures are assumed to be perfectly unforgeable. When the signature scheme used is existentially unforgeable under chosen-message attacks we thus obtain security against computationally bounded adversaries, with a negligible probability of failure. We implicitly assume that parties use domain separation when signing (e.g., via unique session IDs) to ensure that signatures generated for one purpose will be considered invalid if used in another context.

When we say a protocol tolerates t corrupted parties we always mean that it is secure against an adversary who may *adaptively* corrupt up to t parties during execution of the protocol and coordinate the actions of those parties as they deviate from the protocol in an arbitrary manner. An honest party is one who is not corrupted by the end of the protocol. We stress that our claims about adaptive security are only with respect to the "property-based" definitions we give here; we do not consider adaptive security with respect to a simulation-based definition [13, 19].

We are interested in protocols running in one of two possible settings. When a protocol is run in a *synchronous* network, we assume all messages are delivered within a known time bound Δ after they are sent. We allow the adversary to arbitrarily schedule delivery of messages subject to this bound, which implies in particular that we consider a *rushing* adversary who may obtain messages sent to it before sending messages of its own. In the synchronous case, we also assume all parties begin running the protocol at the same time, and all parties have local clocks that progress at the same rate. When we refer to a protocol running in an *asynchronous* network, we allow the adversary to arbitrarily schedule delivery of messages without any upper bound on their delivery time. We do, however, require that all messages that are sent are eventually delivered. Importantly, honest parties do not know *a priori* which type of network the protocol is running in.

We may view executions in a synchronous network as proceeding in a series of *rounds*, where execution begins at time 0 and the rth round refers to the period of time from $(r-1)\cdot\Delta$ to $r\cdot\Delta$. When we say a party receives a message in round r we mean that it receives a message in that time interval; when we say it sends a message in round r we means it sends that message at the beginning of that round, i.e., at time $(r-1)\cdot\Delta$. Thus, in a synchronous network all messages sent in round r are received in round r (but in an asynchronous network this need not be the case).

We assume a *coin-flip mechanism* $\mathsf{CoinFlip}$ available as an atomic primitive. This can be viewed as an ideal functionality, parameterized by a value t, that upon receiving input k from $t+1$ parties generates an unbiased coin $\mathsf{Coin}_k \in \{0,1\}$ and sends (k, Coin_k) to all parties. (When run in an asynchronous network, messages to and from $\mathsf{CoinFlip}$ can be arbitrarily delayed.) The key property this ensures is that, if at most t parties are corrupted, at least one honest party must send k to $\mathsf{CoinFlip}$ before the adversary can learn Coin_k. Several protocols for realizing such a coin flip[4] in an asynchronous network, based on general assumptions, are known [1,5,29,33]. For our purposes, we need a protocol that is secure for $t < n/3$ faults, and that *terminates* for $t' < n/2$ faults. Such protocols can be constructed using a threshold unique signature scheme [4, 14, 17, 27].

2.1 Definitions

We are ultimately interested in *Byzantine agreement*, but we find it useful to define the related notions of *broadcast* and *graded consensus*. Relevant definitions follow.

Byzantine agreement. Byzantine agreement allows a set of parties who each hold some initial input to agree on their output. We consider several security properties that may hold for such protocols. For simplicity, we consider the case of agreement on a bit; this is without loss of generality as one can run any such protocol ℓ times to agree on a string of length ℓ.

We consider Byzantine agreement protocols where, in some cases, parties may not terminate immediately upon generating output, or may never terminate. For that reason, we treat termination separately in the definition that follows. By convention, any party that terminates generates output before doing so; however, we allow parties to output the special symbol $\perp$.

Definition 1 (Byzantine agreement). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.*

- **Weak validity:** *Π is t-*weakly valid *if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs either v or $\perp$.*
- **Validity:** *Π is t-*valid *if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v.*
- **Validity with termination:** *Π is t-*valid with termination *if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs v and terminates.*
- **Weak consistency:** *Π is t-*weakly consistent *if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1\}$ such that every honest party outputs either v or $\perp$.*

[4] Some of these realize a p-weak coin flip, where honest parties agree on the coin only with probability $p < 1$. We can also rely on such protocols, at an increase in the expected round complexity by a factor of $O(1/p)$.

– **Consistency:** *Π is t-*consistent *if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.*
  (In the terminology of Goldwasser and Lindell [15], weak consistency might be called "consistency with abort" and consistency might be called "consistency with unanimous abort.")
– **Liveness:** *Π is t-*live *if whenever at most t of the parties are corrupted, every honest party outputs a value in $\{0, 1\}$.*
– **Termination:** *Π is t-*terminating *if whenever at most t of the parties are corrupted, every honest party terminates. Π has* guaranteed termination *if it is n-terminating.*

*If Π is t-valid, t-consistent, t-live, and t-terminating, then we say Π is t-*secure.

While several of the above definitions are not entirely standard, our notion of security matches the standard one. In particular, t-liveness and t-consistency imply that whenever at most t parties are corrupted, there is a $v \in \{0, 1\}$ such that every honest party outputs v. Note that t-validity with termination is weaker than t-validity plus t-termination, as the former does not require termination in case the inputs of the honest parties do not agree.

Broadcast. Protocols for *broadcast* allow a set of parties to agree on a value chosen by a designated sender. We only consider broadcast protocols with guaranteed termination, and so do not mention termination explicitly when defining the various properties.

Definition 2 (Broadcast). *Let Π be a protocol executed by parties P_1, $\ldots$, P_n, where a sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input $v^* \in \{0, 1\}$ and all parties are guaranteed to terminate.*

– **Weak validity:** *Π is t-*weakly valid *if the following holds whenever at most t of the parties are corrupted: if P^* is honest, then every honest party outputs either v^* or $\perp$.*
– **Validity:** *Π is t-*valid *if the following holds whenever at most t of the parties are corrupted: if P^* is honest, then every honest party outputs v^*.*
– **Weak consistency:** *Π is t-*weakly consistent *if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1\}$ such that every honest party outputs either v or $\perp$.*
– **Consistency:** *Π is t-*consistent *if the following holds whenever at most t of the parties are corrupted: there is a $v \in \{0, 1, \perp\}$ such that every honest party outputs v.*

– **Liveness:** *Π is t-*live *if whenever at most t of the parties are corrupted, every honest party outputs a value in $\{0, 1\}$.*

*If Π is t-valid, t-consistent, and t-live, then we say Π is t-*secure.

Graded consensus. As a stepping stone to Byzantine agreement, it is also useful to define *graded consensus* [9]. Here, each party outputs both a value $v \in \{0, 1, \perp\}$ as well as a *grade* $g \in \{0, 1, 2\}$. As in the case of Byzantine agreement, we consider protocols that may not terminate.

Definition 3 (Graded consensus). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.*

– **Graded validity:** *Π achieves t-*graded validity *if the following holds whenever at most t of the parties are corrupted: if every honest party's input is equal to the same value v, then all honest parties output $(v, 2)$.*
– **Graded consistency:** *Π achieves t-*graded consistency *if the following hold whenever at most t of the parties are corrupted: (1) If two honest parties output grades g, g', then $|g - g'| \leq 1$. (2) If two honest parties output (v, g) and (v', g') with $g, g' \geq 1$, then $v = v'$.*
– **Liveness:** *Π is t-*live *if whenever at most t of the parties are corrupted, every honest party outputs (v, g) with either $v \in \{0, 1\}$ and $g \geq 1$, or $v = \perp$ and $g = 0$.*

*If Π achieves t-graded validity, t-graded consistency, and t-liveness then we say Π is t-*secure.

3 Synchronous BA with Fallback (Weak) Validity

In this section we show a protocol that is secure for some threshold t_s of corrupted parties when run in a synchronous network, and achieves weak validity (though liveness and weak consistency may not hold) for a lower threshold t_a even when run in an asynchronous network.

Our protocol relies on a variant of the Dolev-Strong broadcast protocol [8] as a subroutine. Since we use a slightly non-standard version of that protocol, we describe it in Figure 1 for completeness. In the protocol, we say that (v, SET) is an *r-correct message* (from the point of view of a party P_i) if SET contains valid signatures on v from P^* and $r - 1$ additional, distinct parties other than P_i.

Lemma 1. *Broadcast protocol Π_{DS} satisfies the following properties:*

1. *When run in a synchronous network, it is n-consistent and n-valid.*

Protocol Π_{DS}

Round 1: P^* signs its input v^* to obtain a signature σ^*. It sets $\mathsf{SET} := \{\sigma^*\}$ and sends (v^*, SET) to all parties.

Rounds 1 to $\mathbf{n-1}$: Each P_i begins with $\mathsf{ACC}_i = \emptyset$, and then acts as follows: upon receiving an r-correct message (v, SET) in round r, add v to ACC_i. If $r < n-1$, then also compute a signature σ_i on v, let $\mathsf{SET} := \mathsf{SET} \cup \{\sigma_i\}$, and send (v, SET) to all parties in the following round. (This is done at most once for each (v, r) pair.)

Output determination: At time $(n-1) \cdot \Delta$, if ACC_i contains one value, then output that value and terminate. In any other case, output $\perp$ and terminate.

Fig. 1. The Dolev-Strong broadcast protocol Π_{DS}.

2. *When run in an asynchronous network, it is n-weakly valid.*

Proof. The standard analysis of the Dolev-Strong protocol shows that, when run in a synchronous network with any number of corrupted parties, $\mathsf{ACC}_i = \mathsf{ACC}_j$ for any honest parties P_i, P_j. This implies n-consistency. Since an honest P^* sends a 1-correct message to all honest parties, and the attacker cannot forge signatures of the honest sender, n-validity holds.

The second claim follows because an attacker cannot forge the signature of an honest P^*.

We now define a BA protocol using Π_{DS} as a sub-routine. This protocol is parameterized by a value t_a which determines the security thresholds the protocol satisfies.

Protocol $\Pi_{\mathsf{SBA}}^{t_a}$

Each P_i initially holds a bit v_i. The protocol proceeds as follows:

- Each party P_i broadcasts v_i by running Π_{DS} as the sender.
- Let v_j^i denote the output of P_i in the jth execution of Π_{DS}.
- Each P_i does: if there are at least $2t_a + 1$ values v_j^i that are in $\{0, 1\}$, output the majority of those values (with a tie broken arbitrarily) and terminate. Otherwise, output $\perp$ and terminate.

Fig. 2. A Byzantine agreement protocol, parameterized by t_a.

Theorem 1. *For any t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$, Byzantine agreement protocol $\Pi_{\mathsf{SBA}}^{t_a}$ satisfies the following properties:*

1. *When the protocol is run in a synchronous network, it is t_s-secure.*

2. *When the protocol is run in an asynchronous network, it is t_a-weakly valid.*

Moreover, the protocol has guaranteed termination in both cases, and when run in a synchronous network every honest party terminates in time at most $n \cdot \Delta$.

Proof. The claim about termination is immediate.

When run in a synchronous network with t_s corrupted parties, at least $n - t_s > 2t_a$ of the executions of Π_{DS} result in boolean output for all honest parties (by n-validity of Π_{DS}) and so all honest parties generate boolean output in $\Pi_{\mathsf{SBA}}^{t_a}$; this proves t_s-liveness. By n-consistency of Π_{DS}, all honest parties agree on the $\{v_j\}$ values they obtain and hence $\Pi_{\mathsf{SBA}}^{t_a}$ is t_s-consistent (in fact, it is n-consistent). Finally, n-validity of Π_{DS} implies that when all honest parties begin holding the same input $v \in \{0, 1\}$, then all honest parties will have v as their majority value. This proves t_s-validity (in fact, the protocol is t-valid for any $t < n/2$).

For the second claim, assume all honest parties begin holding the same input v, and t_a parties are corrupted. Any honest party P_i who generates boolean output must have at least $2t_a + 1$ boolean values $\{v_j^i\}$, of which at most t_a of these can be equal to $\bar{v}$. Hence, any honest party who generates boolean output will in fact output v.

4 Validity-Optimized Asynchronous BA

Here we show a protocol that is secure for some threshold when run in an asynchronous network, and achieves validity for a higher threshold. Throughout this section we only consider protocols running in an asynchronous network, and so drop explicit mention of this fact for the remainder of this section.

Theorem 2. *For any t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$, there is an n-party protocol for Byzantine agreement that, when run in an asynchronous network, is t_a-secure and also achieves t_s-validity with termination.*

Our proof of Theorem 2 proceeds in a number of steps. In Section 4.1 we describe a "validity-optimized" protocol $\Pi_{\mathsf{GC}}^{t_s}$ for graded consensus that is t_a-secure and also achieves t_s-graded validity. Then, in Section 4.2, we show a Byzantine agreement protocol $\Pi_{\mathsf{ABA}}^{t_s}$ using $\Pi_{\mathsf{GC}}^{t_s}$ as a subroutine. This protocol illustrates our main ideas, and achieves all the properties claimed in Theorem 2 except termination. We then discuss how termination can be added using existing techniques.

Our protocol is based on the work of Mostéfaoui et al. [29], but allows for variable thresholds. Also, our description simplifies theirs by presenting the protocol in a modular fashion.

4.1 Validity-Optimized Graded Consensus

Our graded consensus protocol relies on a sub-protocol $\Pi^{t_s}_{\mathsf{prop}}$ for proposing values, shown in Figure 3. This protocol is parameterized by a value t_s that determines its security thresholds. We begin by proving some properties of $\Pi^{t_s}_{\mathsf{prop}}$. Throughout, we let n denote the number of parties.

> **Protocol $\Pi^{t_s}_{\mathsf{prop}}$**
>
> We describe the protocol from the point of view of a party holding input $v \in \{0, 1, \lambda\}$.
>
> 1. Set $\mathsf{vals} := \emptyset$.
> 2. Send $(\mathsf{prepare}, v)$ to all parties.
> 3. Upon receiving the message $(\mathsf{prepare}, b)$, for some $b \in \{0, 1, \lambda\}$, from strictly more than t_s parties, do: If $(\mathsf{prepare}, b)$ has not been sent, then send $(\mathsf{prepare}, b)$ to all parties.
> 4. Upon receiving the message $(\mathsf{prepare}, b)$, for some $b \in \{0, 1, \lambda\}$, from at least $n - t_s$ parties, set $\mathsf{vals} := \mathsf{vals} \cup \{b\}$.
> 5. Upon adding the first value $b \in \{0, 1, \lambda\}$ to vals, send $(\mathsf{propose}, b)$ to all parties.
> 6. Once at least $n - t_s$ messages $(\mathsf{propose}, b)$ have been received on values $b \in \mathsf{vals}$, let $\mathsf{prop} \subseteq \mathsf{vals}$ be the set of values carried by those messages. Output prop (but continue running).

Fig. 3. A sub-protocol for proposing values, parameterized by t_s.

Lemma 2. *Assume $t_a < n - 2 \cdot t_s$ parties are corrupted in an execution of $\Pi^{t_s}_{\mathsf{prop}}$. If two honest parties P_i, P_j output $\{b\}$, $\{b'\}$, respectively, then $b = b'$.*

Proof. Since P_i outputs $\{b\}$, it must have received at least $n - t_s$ messages $(\mathsf{propose}, b)$, of which at least $n - t_s - t_a$ of those were sent by honest parties. Similarly, P_j must have received at least $n - t_s - t_a$ messages $(\mathsf{propose}, b')$ that were sent by honest parties. If $b \neq b'$, then because $2 \cdot (n - t_s - t_a)$ is strictly greater than the number of honest parties $n - t_a$, this would mean that some honest party sent $\mathsf{propose}$ messages on two different values, which is impossible.

Lemma 3. *Assume $t_a \leq t_s$ parties are corrupted in an execution of $\Pi^{t_s}_{\mathsf{prop}}$. If no honest party has input v, then no honest party outputs* prop *containing v.*

Proof. If v was not input by any honest party, then at most $t_a \leq t_s$ messages $(\mathsf{prepare}, v)$ are sent in step 2. Thus, no honest party ever sends a message $(\mathsf{prepare}, v)$, and consequently no honest party ever sends a message $(\mathsf{propose}, v)$. It follows that no honest party ever adds v to vals, and so no honest party outputs prop containing v.

Lemma 4. *Assume t_a parties are corrupted in an execution of $\Pi^{t_s}_{\mathsf{prop}}$, where $t_a < n - 2 \cdot t_s$ and $t_a \leq t_s$. If an honest party sends a message $(\mathsf{propose}, b)$, all honest parties add b to* vals.

Proof. Suppose some honest party P_i sends $(\mathsf{propose}, b)$. Then P_i must have received at least $n - t_s$ messages $(\mathsf{prepare}, b)$. At least $n - t_s - t_a > t_s$ of these must have been sent by honest parties, and so eventually all other honest parties also receive strictly more than t_s messages $(\mathsf{prepare}, b)$. We thus see that every honest party will eventually send $(\mathsf{prepare}, b)$. Therefore, every honest party will eventually receive at least $n - t_a \geq n - t_s$ messages $(\mathsf{prepare}, b)$, and consequently every honest party will add b to vals.

The following lemmas will help to prove liveness of $\Pi^{t_a}_{\mathsf{GC}}$.

Lemma 5. *Assume t_a parties are corrupted in an execution of $\Pi^{t_s}_{\mathsf{prop}}$, where $t_a < n - 2 \cdot t_s$ and $t_a \leq t_s$. If all honest parties hold one of two different inputs, then all honest parties output.*

Proof. We first argue that every honest party sends a $\mathsf{propose}$ message. Indeed, there are $n - t_a$ honest parties, so at least $\frac{1}{2}(n - t_a) > t_s$ honest parties must have the same input v. Therefore, all honest parties receive strictly more than t_s messages $(\mathsf{prepare}, v)$. Consequently, all honest parties will eventually send $(\mathsf{prepare}, v)$. Thus, every honest party receives $n - t_a \geq n - t_s$ messages $(\mathsf{prepare}, v)$ and adds v to vals. In particular, vals is nonempty and so every honest party sends a $\mathsf{propose}$ message.

Each honest party thus receives at least $n - t_a \geq n - t_s$ $\mathsf{propose}$ messages sent by honest parties. By Lemma 4, for any b proposed by an honest party, all honest parties eventually have $b \in \mathsf{vals}$. Thus, every honest party eventually receives at least $n - t_s$ propose messages for values in their set vals, and therefore all honest parties terminate.

$\Pi^{t_s}_{\mathsf{prop}}$ satisfies a notion of validity even for t_s corrupted parties.

Lemma 6. *Assume $t_s < n/2$ parties are corrupted in an execution of $\Pi_{\mathsf{prop}}^{t_s}$. If all honest parties hold the same input v, then all honest parties output* $\mathsf{prop} = \{v\}$.

Proof. Suppose t_s parties are corrupted, and all honest parties hold the same input v. In step 2, all $n - t_s$ honest parties send $(\mathsf{prepare}, v)$, and so all honest parties add v to vals. Any $\mathsf{prepare}$ messages on other values in step 2 are sent by the $t_s < n - t_s$ corrupted parties, and so no honest party ever adds a value other than v to vals. Thus, all $n - t_s$ honest parties send their (single) $\mathsf{propose}$ message $(\mathsf{propose}, v)$ in step 5. It follows that every honest party outputs $\mathsf{prop} = \{v\}$ in step 6.

Protocol $\Pi_{\mathsf{GC}}^{t_s}$

We describe the protocol from the point of view of a party with input $v \in \{0, 1\}$.

- Set $b_1 := v$.
- Run protocol $\Pi_{\mathsf{prop}}^{t_s}$ using input b_1, and let prop_1 denote the output.
- If $\mathsf{prop}_1 = \{b\}$, then set $b_2 := b$. Otherwise, set $b_2 := \lambda$.
- Run protocol $\Pi_{\mathsf{prop}}^{t_s}$ using input b_2, and let prop_2 denote the output.
- If $\mathsf{prop}_2 = \{b'\}$ for $b' \neq \lambda$, then output $(b', 2)$. If $\mathsf{prop}_2 = \{b', \lambda\}$ for $b' \neq \lambda$, then output $(b', 1)$. If $\mathsf{prop}_2 = \{\lambda\}$, then output $(\perp, 0)$.

Fig. 4. A protocol for graded consensus, parameterized by t_s.

In Figure 4 we show a graded consensus protocol $\Pi_{\mathsf{GC}}^{t_s}$ that relies on $\Pi_{\mathsf{prop}}^{t_s}$ as a subroutine. Note that parties do not terminate upon generating output, and instead continue to participate. (We will revisit this point later in the section when we construct Π_{ABA}, which does terminate. It is important that parties continue participating until they are sure all other honest parties will be able to terminate.) We now analyze the protocol.

Lemma 7. *If $t_s < n/2$, then $\Pi_{\mathsf{GC}}^{t_s}$ achieves t_s-graded validity.*

Proof. Suppose t_s parties are corrupted, and every honest party's input is equal to the same value v. By Lemma 6, all honest parties have $\mathsf{prop}_1 = \{v\}$ following the first execution of $\Pi_{\mathsf{prop}}^{t_s}$, and so use v as the input for the second execution of $\Pi_{\mathsf{prop}}^{t_s}$. By the same reasoning, all honest parties have $\mathsf{prop}_2 = \{v\}$ after the second execution of $\Pi_{\mathsf{prop}}^{t_s}$. Thus, all honest parties output $(v, 2)$.

Lemma 8. *Assume $t_a \le t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_s}$ achieves t_a-graded consistency.*

Proof. Suppose t_a parties are corrupted. First, we show that the grades output by two honest parties P_i, P_j differ by at most 1. The only way this can possibly fail is if one of the parties (say, P_i) outputs a grade of 2. P_i must then have received $\mathsf{prop}_2 = \{b\}$, for some $b \in \{0, 1\}$, as its output from the second execution of $\Pi_{\mathsf{prop}}^{t_s}$. It follows from Lemma 2 that another party P_j could not have received $\mathsf{prop}_2 = \{\lambda\}$. Therefore, it is not possible for P_j to output grade 0.

Next, we show that any two honest parties that output nonzero grades must output the same value. Observe first that there is a bit b such that the inputs of all the honest parties to the second execution of $\Pi_{\mathsf{prop}}^{t_s}$ lie in $\{b, \lambda\}$. (Indeed, if all honest parties set $b_2 := \lambda$ this claim is immediate. On the other hand, if some honest party sets $b_2 := b \in \{0, 1\}$ then they must have $\mathsf{prop}_1 = \{b\}$; but then Lemma 2 implies that any other honest party who sets b_2 to anything other than λ will set it equal to b as well.) Lemma 3 thus implies that no honest party outputs a set prop_2 after the second execution of $\Pi_{\mathsf{prop}}^{t_s}$ that contains a value other than b or λ. Thus, any two honest parties that output a nonzero grade must output the same value b.

Lemma 9. *Assume $t_a \le t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{GC}}^{t_s}$ achieves t_a-liveness.*

Proof. All honest parties hold input in $\{0, 1\}$ in the first execution of $\Pi_{\mathsf{prop}}^{t_s}$, so Lemma 5 shows that all honest parties generate output in that execution. As in the proof of the previous lemma, there is a bit b such that the inputs of all the honest parties to the second execution of $\Pi_{\mathsf{prop}}^{t_s}$ lie in $\{b, \lambda\}$; so, applying Lemma 5 again, that execution also produces output for all honest parties. Moreover, by Lemma 3, the set prop_2 output by any honest party is a nonempty subset of $\{b, \lambda\}$, i.e., is either $\{b\}, \{b, \lambda\}$, or $\{\lambda\}$. Thus, $\Pi_{\mathsf{GC}}^{t_s}$ is t_s-live.

4.2 Validity-Optimized Byzantine Agreement

We present a Byzantine agreement protocol $\Pi_{\mathsf{ABA}}^{t_s}$ in Figure 5. Recall from Section 2 that we assume an atomic primitive $\mathsf{CoinFlip}$ that allows all parties to generate and learn an unbiased value $\mathsf{Coin}_k \in \{0, 1\}$ for $k = 1, 2, \ldots$; we refer there for a discussion as to how it can be realized.

When a party terminates in $\Pi_{\mathsf{ABA}}^{t_s}$, they will also stop participating in any $\Pi_{\mathsf{prop}}^{t_s}$ or $\Pi_{\mathsf{GC}}^{t_s}$ executions. Because the security properties of $\Pi_{\mathsf{prop}}^{t_s}$ and

$\Pi_{\mathsf{GC}}^{t_s}$ hold for an asynchronous network, honest parties dropping out may cause the executions to lose liveness, but cannot break security otherwise. We argue below that if any honest party is able to terminate in $\Pi_{\mathsf{ABA}}^{t_s}$ with output b, then all honest parties are indeed able to terminate, and by the security properties of the subprotocols, they can only output the same value b.

In the following, we say a message $(\mathsf{commit}, b, \sigma)$ from party P_i is *valid* if $b \in \{0,1\}$ and σ is a valid signature from P_i on (commit, b). Furthermore, we say that a set of signatures is a *certificate for* b if it contains valid signatures on (commit, b) from at least $t_s + 1$ distinct parties.

Protocol $\Pi_{\mathsf{ABA}}^{t_s}$

We describe the protocol from the point of view of a party P_i with input $v \in \{0,1\}$ and secret key sk_i.

Set $b := v$, $\mathsf{committed} = \mathsf{false}$, and $k := 1$. Then do:

1. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input b, and let (b, g) denote the output.
2. $\mathsf{Coin}_k \leftarrow \mathsf{CoinFlip}(k)$.
3. If $g < 2$ then set $b := \mathsf{Coin}_k$.
4. Run $\Pi_{\mathsf{GC}}^{t_s}$ on input b, and let (b', g) denote the output. If $g > 0$, set $b := b'$.
5. If $g = 2$ and $\mathsf{committed} = \mathsf{false}$, set $\mathsf{committed} := \mathsf{true}$, compute $\sigma \leftarrow \mathsf{Sign}_{sk_i}(\mathsf{commit}, b)$, and send $(\mathsf{commit}, b, \sigma)$ to all parties.
6. Set $k := k + 1$ and repeat from (1).

To terminate:

- Upon receiving valid commit messages on a single value b from at least $t_s + 1$ distinct parties, combine the signatures into a certificate $\Sigma := (\sigma_1, \ldots, \sigma_{t_s+1})$, send $(\mathsf{notify}, b, \Sigma)$ to all parties, output b, and terminate.
- Upon receiving $(\mathsf{notify}, b, \Sigma)$ such that Σ contains valid signatures on (commit, b) from at least $t_s + 1$ distinct parties, forward $(\mathsf{notify}, b, \Sigma)$ to all parties, output b and terminate.

Fig. 5. A Byzantine agreement protocol, parameterized by t_s.

Lemma 10. *If $t_s < n/2$, then protocol $\Pi_{\mathsf{ABA}}^{t_s}$ satisfies t_s-validity with termination.*

Proof. Suppose there are at most t_s corrupted parties and all honest parties initially hold $v \in \{0,1\}$. All honest parties use input v in the first execution of $\Pi_{\mathsf{GC}}^{t_s}$ in the first iteration; t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_s}$ (cf.

Lemma 7) implies they all output $(v, 2)$ from that execution. Thus, all honest parties ignore the result of the coin flip and run a second instance of $\Pi_{\mathsf{GC}}^{t_s}$ using input v, again unanimously obtaining $(v, 2)$ as output. Therefore, all honest parties either send a commit message on v in step 5 of the first iteration of $\Pi_{\mathsf{ABA}}^{t_s}$, or have already received (and forwarded) a certificate Σ containing at least t_s signatures on v. Furthermore, honest parties will receive at most $t_s < t_s + 1$ commit messages on $v' \neq v$. Therefore, all honest parties eventually receive valid commit messages on v from $n - t_s \geq t_s + 1$ distinct parties (via either a single notify message or individual commit messages), output v, and terminate.

Lemma 11. *Assume $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ABA}}^{t_s}$ satisfies t_a-liveness and t_a-consistency. Moreover: (1) some honest party sends a* commit *message within an expected constant number of iterations, and (2) if some honest party sends a* commit *message on a bit b, then all honest parties terminate with output b.*

Proof. Assume t_a parties are corrupted. Consider an iteration k of the protocol by which no honest party has yet sent a commit message. Let Agree be the event that all honest parties use the same input to the second execution of $\Pi_{\mathsf{GC}}^{t_s}$ in that iteration. If Agree occurs, then t_s-graded validity of $\Pi_{\mathsf{GC}}^{t_s}$ implies that all honest parties will obtain a grade of 2 in that execution and hence at least one honest party will send a commit message in iteration k. We show that Agree occurs with probability at least $1/2$. We distinguish two cases:

- Say some honest party outputs $(b, 2)$ in the first execution of $\Pi_{\mathsf{GC}}^{t_s}$ in iteration k. By t_a-graded consistency of $\Pi_{\mathsf{GC}}^{t_s}$, all honest parties output either $(b, 2)$ or $(b, 1)$ in that execution of $\Pi_{\mathsf{GC}}^{t_s}$. Since Coin_k is not revealed until after the first honest party generates output for that execution of $\Pi_{\mathsf{GC}}^{t_s}$, this means b is chosen independently of Coin_k. If $\mathsf{Coin}_k = b$, which occurs with probability $1/2$, then all honest parties will use the same input in the second execution of $\Pi_{\mathsf{GC}}^{t_s}$ in iteration k.
- If no honest party outputs $(b, 2)$ after the first execution of $\Pi_{\mathsf{GC}}^{t_s}$, then all honest parties will use Coin_k as their input in the second execution of $\Pi_{\mathsf{GC}}^{t_s}$ in iteration k.

Thus, in expected constant rounds *some* honest party sends a commit message. We next show that if some honest party P_i sends a commit message on b, then all other honest parties terminate with output b.

Let k be the first iteration in which some honest party sends a commit message, and let P_i be the first honest party to have sent a commit message on b in iteration k. P_i must have seen $(b, 2)$ as the output of the second execution of $\Pi^{t_s}_{\mathsf{GC}}$ in iteration k. By t_a-graded consistency of $\Pi^{t_s}_{\mathsf{GC}}$, every honest party who obtains output from that execution of $\Pi^{t_s}_{\mathsf{GC}}$ will receive either $(b, 1)$ or $(b, 2)$. This means that all honest parties who continue running will input b to the next iteration of $\Pi^{t_s}_{\mathsf{GC}}$. By t_s-graded validity of $\Pi^{t_s}_{\mathsf{GC}}$, honest parties will continue to input b to each iteration of $\Pi^{t_s}_{\mathsf{GC}}$, ignoring the coin, for as long as they continue running. Thus, we see that no honest party will ever send a commit message on $1 - b$. Hence, there will never exist a set of commit messages on $1 - b$ from at least $t_s + 1$ distinct parties, and so no honest party will ever output $1 - b$.[5]

We now consider two cases: either some honest party terminates, or (by t_a-liveness and t_s-validity of $\Pi^{t_s}_{\mathsf{GC}}$) all honest parties reach step 5 of iteration $k + 1$ and receive output $(b, 2)$. In the latter case, all honest parties will send a commit message on b. Therefore, eventually all honest parties will receive at least $n - t_a \geq t_s + 1$ commit messages on b and can terminate with output b. In the case that some honest party terminates, liveness of $\Pi^{t_s}_{\mathsf{GC}}$ may be lost. However, in order for that party to terminate, they must have already received sufficient signatures on b, and forwarded those signatures as a certificate Σ to all other parties. Therefore, all honest parties eventually receive Σ and terminate with output b.

Corollary 1. *For any t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$, there is an n-party protocol for Byzantine agreement that, when run in an asynchronous network, achieves t_s-validity, t_a-consistency, and t_a-termination.*

Proof. If $t_s < t_a$, then we parameterize $\Pi^{t_s}_{\mathsf{ABA}}$ with $t_s = t_a$. Lemma 11 and Lemma 10 immediately show that $\Pi^{t_s}_{\mathsf{ABA}}$ is t_a-secure (and therefore also t_s-valid if $t_s < t_a$).

Otherwise, if $t_s \geq t_a$, we can apply Lemma 11 directly. The corollary thus follows from Lemmas 10 and 11.

5 Main Protocol

Fix n, t_a, t_s with $t_a < n/3$ and $t_a + 2 \cdot t_s < n$. As in the proof of Corollary 1 we may assume $t_a \leq t_s$. Our main protocol $\Pi^{t_a,t_s}_{\mathsf{HBA}}$ is given in Figure 6. It relies on the following sub-protocols:

– $\Pi^{t_a}_{\mathsf{SBA}}$ is an n-party BA protocol that is t_s-secure when run in a synchronous network, and t_a-weakly valid when run in an asynchronous

[5] It is important to note that $\Pi^{t_s}_{\mathsf{GC}}$ cannot lose consistency or validity if honest parties stop sending messages, only liveness. Because it is an asynchronous protocol, it necessarily tolerates that parties stop sending messages.

network. Moreover, the protocol has guaranteed termination regardless of the network, and when run in a synchronous network all honest parties terminate by time $n \cdot \Delta$. The existence of such a protocol is guaranteed by Theorem 1.
– $\Pi^{t_s}_{\mathsf{ABA}}$ is an n-party BA protocol that is t_a-secure and t_s-valid with termination when run in an asynchronous network. (Of course, these properties also hold if the protocol is run in a synchronous network.) The existence of such a protocol is guaranteed by Theorem 2.

Protocol $\Pi^{t_a,t_s}_{\mathsf{HBA}}$

Each P_i initially holds a bit v_i. The protocol proceeds as follows:

– Each party P_i runs $\Pi^{t_a}_{\mathsf{SBA}}$ using input v_i for time $n \cdot \Delta$. Let b_i denote the output of P_i from this protocol, with $b_i = \perp$ denoting no output.
– Each party P_i does the following: if $b_i \neq \perp$, set $v_i^* := b_i$; otherwise set $v_i^* := v_i$. Then run $\Pi^{t_s}_{\mathsf{ABA}}$ using input v_i^*, output the result, and terminate.

Fig. 6. A Byzantine agreement protocol, parameterized by t_a, t_s.

Theorem 3. *Let n, t_a, t_s be as above. Then protocol $\Pi^{t_a,t_s}_{\mathsf{HBA}}$ satisfies the following properties:*

1. *When the protocol is run in a synchronous network, it is t_s-secure.*
2. *When the protocol is run in an asynchronous network, it is t_a-secure.*

Proof. First consider the case when $\Pi^{t_a,t_s}_{\mathsf{HBA}}$ is run in a synchronous network, and at most t_s parties are corrupted. By t_s-security of $\Pi^{t_a}_{\mathsf{SBA}}$, after running $\Pi^{t_a}_{\mathsf{SBA}}$ there is a value $b \neq \perp$ such that $b_i = b$ for every honest P_i. Moreover, if every honest party's input was equal to the same value v, then $b = v$. Thus, all honest parties set v_i^* to the same value b and, if every party's input was the same value v, then $v_i^* = v$. By t_s-validity with termination of $\Pi^{t_s}_{\mathsf{ABA}}$, all honest parties terminate and agree on their output from $\Pi^{t_a,t_s}_{\mathsf{HBA}}$, proving t_s-consistency, t_s-liveness, and t_s-termination. Moreover, if every honest party's original input was equal to the same value v, then the output of $\Pi^{t_s}_{\mathsf{ABA}}$ (and thus of $\Pi^{t_a,t_s}_{\mathsf{HBA}}$) is equal to v. This proves t_s-validity.

Next consider the case when $\Pi^{t_a,t_s}_{\mathsf{HBA}}$ is run in an asynchronous network, and at most t_a parties are corrupted. The protocol inherits t_a-consistency, t_a-liveness, and t_a-termination from t_a-security of $\Pi^{t_s}_{\mathsf{ABA}}$, and so it only remains to argue t_a-validity. Assume every honest party's initial input is equal to the same value v. Then t_a-weak validity of $\Pi_{\mathsf{SBA}}^{t_a}$, plus the fact that it always terminates, imply that $b_i \in \{v, \bot\}$, and hence $v_i^* = v$, for every honest P_i. It follows from t_s-validity with termination (note $t_a \leq t_s$) of $\Pi_{\mathsf{ABA}}^{t_s}$ that all honest parties output v and terminate.

6 Impossibility Result

We show here that our positive result from the previous section is tight. That is:

Theorem 4. *For any n, if $t_a \geq n/3$ or $t_a + 2 \cdot t_s \geq n$ there is no n-party protocol for Byzantine agreement that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network.*

The case of $t_a \geq n/3$ follows from existing impossibility results for asynchronous consensus, so the interesting case is when $t_a < n/3$ but $t_a + 2 \cdot t_s \geq n$. Theorem 4 follows from the lemma below.

Lemma 12. *Fix n, t_a, t_s with $t_a + 2t_s \geq n$. If an n-party Byzantine agreement protocol is t_s-valid in a synchronous network, then it cannot also be t_a-weakly consistent in an asynchronous network.*

Proof. The proof is similar to that of [36]. Assume $t_a + 2t_s = n$ and fix a BA protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$, and consider the following experiment:

- Parties in S_0 run Π using input 0, and parties in S_1 run Π using input 1. All communication between parties in S_0 and parties in S_1 is blocked (but all other messages are delivered within time Δ).
- Create virtual copies of each party in S_a, call them S_a^0 and S_a^1. Parties in S_a^0 run Π using input 0, and communicate only with each other and parties in S_0. Parties in S_a^1 run Π using input 1, and communicate only with each other and parties in S_1.

Consider an execution of Π in a synchronous network where parties in S_1 are corrupted and simply abort, and all remaining (honest) parties use input 0. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_a^0$ in the above experiment. In particular, t_s-validity of Π implies that all parties in S_0 output 0. Analogously, all parties in S_1 output 1.

Next consider an execution of Π in an asynchronous network where parties in S_a are corrupted, and run Π using input 0 when interacting with S_0 while running Π using input 1 when interacting with S_1. Moreover, all communication between the (honest) parties in S_0 and S_1 is delayed indefinitely. The views of the honest parties in this execution are distributed identically to the views of $S_0 \cup S_1$ in the above experiment, yet the conclusion of the preceding paragraph shows that weak consistency is violated.

Acknowledgments. Julian Loss was supported by ERC Project ERCC (FP7/615074).

References

1. Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous Byzantine agreement with optimal resilience. In *27th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 405–414. ACM Press, 2008.
2. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication, 2019. Available at `http://eprint.iacr.org/2019/270`.
3. Zuzana Beerliová-Trubíniová, Martin Hirt, and Jesper Buus Nielsen. On the theoretical gap between synchronous and asynchronous MPC protocols. In *29th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 211–218. ACM Press, 2010.
4. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *J. Cryptology*, 18(3):219–246, 2005.
5. Ran Canetti and Tal Rabin. Fast asynchronous Byzantine agreement with optimal resilience. In *25th Annual ACM Symp. on Theory of Computing (STOC)*, pages 42–51. ACM Press, 1993.
6. Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Computer Systems*, 20(4):398–461, 2002.
7. Ivan Damgård, Martin Geisler, Mikkel Krøigaard, and Jesper Buus Nielsen. Asynchronous multiparty computation: Theory and implementation. In *12th Intl. Conference on Theory and Practice of Public Key Cryptography—PKC 2009*, volume 5443 of *LNCS*, pages 160–179. Springer, 2009.
8. Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Computing*, 12(4):656–666, 1983.
9. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM J. Computing*, 26(4):873–933, 1997.
10. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
11. Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-threshold broadcast and detectable multi-party computation. In *Advances in Cryptology—Eurocrypt 2003*, volume 2656 of *LNCS*, pages 51–67. Springer, 2003.
12. Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds sufficient for authenticated Byzantine agreement. In *23rd Intl. Symp. on Distributed Computing (DISC)*, volume 5805 of *LNCS*, pages 449–463. Springer, 2009.

13. Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *30th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 179–186. ACM Press, 2011.
14. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies, 2017. Available at `http://eprint.iacr.org/2017/454`.
15. Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005.
16. Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Advances in Cryptology—Crypto 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, 2019.
17. Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series consensus system, rev. 1, 2018. Available at `https://dfinity.org/faq`.
18. Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *24th USENIX Security Symposium*, pages 129–144. USENIX Association, 2015.
19. Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology—Eurocrypt 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, 2010.
20. Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. *J. Computer and System Sciences*, 75(2):91–112, 2009.
21. Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: Speculative Byzantine fault tolerance. *ACM Trans. Computer Systems*, 27(4):7:1–7:39, 2009.
22. Klaus Kursawe. Optimistic Byzantine agreement. In *21st Symposium on Reliable Distributed Systems (SRDS)*, pages 262–267. IEEE Computer Society, 2002.
23. Leslie Lamport. The part-time parliament. Technical Report 49, DEC Systems Research Center, 1989.
24. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Programming Language Systems*, 4(3):382–401, 1982.
25. Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolic. XFT: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 485–500. USENIX Association, 2016.
26. Chen-Da Liu-Zhang, Julian Loss, Tal Moran, Ueli Maurer, and Daniel Tschudi. Robust MPC: Asynchronous responsiveness yet synchronous security, 2019. Available at `http://eprint.iacr.org/2019/159`.
27. Julian Loss and Tal Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds, 2018. Available at `http://eprint.iacr.org/2018/235`.
28. Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible Byzantine fault tolerance, 2019. Available at `https://arxiv.org/abs/1904.10067`.
29. Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *J. ACM*, 62(4):31:1–31:21, 2015.
30. Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319. USENIX Association, 2014.
31. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st Intl. Symp. on Distributed Computing (DISC)*, volume 91

of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2017.

32. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Advances in Cryptology—Eurocrypt 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, 2018.
33. Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Simple and efficient asynchronous byzantine agreement with optimal resilience. In *28th Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 92–101. ACM Press, 2009.
34. Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Trans. Information Theory*, 64(6):4207–4227, 2018.
35. M. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.
36. Sam Toueg. Randomized Byzantine agreements. In *3rd Annual ACM Symp. on Principles of Distributed Computing (PODC)*, pages 163–178. ACM Press, 1984.