TARDIGRADE: An Atomic Broadcast Protocol for Arbitrary Network Conditions

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

[1] University of Maryland College Park
{erblum,jkatz2}@umd.edu
[2] CISPA Helmoltz Center for Information Security
lossjulian@gmail.com

Abstract. We study the problem of *atomic broadcast*—the underlying problem addressed by blockchain protocols—in the presence of a malicious adversary who corrupts some fraction of the n parties running the protocol. Existing protocols are either robust for any number of corruptions in a *synchronous* network (where messages are delivered within some known time Δ) but fail if the synchrony assumption is violated, or tolerate fewer than $n/3$ corrupted parties in an *asynchronous* network (where messages can be delayed arbitrarily) and cannot tolerate more corruptions even if the network happens to be well behaved.

We design an atomic broadcast protocol (TARDIGRADE) that, for any $t_s \geq t_a$ with $2t_s + t_a < n$, provides security against t_s corrupted parties if the network is synchronous, while remaining secure when t_a parties are corrupted even in an asynchronous network. We show that TARDIGRADE achieves optimal tradeoffs between t_s and t_a. Finally, we show a second protocol (UPGRADE) with similar (but slightly weaker) guarantees that achieves per-transaction communication complexity linear in n.

Keywords: Atomic broadcast · Byzantine agreement · Consensus.

1 Introduction

Atomic broadcast [10] is a fundamental problem in distributed computing that can be viewed as a generalization of *Byzantine agreement* (BA) [20,32]. Roughly speaking, a BA protocol allows a set of n parties to agree on a value *once*, even if some parties are *Byzantine*, i.e., corrupted by an adversary who may cause them to behave arbitrarily. In contrast, an atomic broadcast (ABC) protocol allows parties to repeatedly agree on values by including them a totally-ordered, append-only log maintained by all parties. (Formal definitions are given in Section 3. Note that ABC is not obtained by simply repeating a BA protocol multiple times; this point is discussed further below.) Atomic broadcast is used as a

⋆ Work performed under financial assistance award 70NANB19H126 from the U.S. Department of Commerce, National Institute of Standards and Technology, and also supported in part by NSF award #1837517.

⋆⋆ Portions of this work were done while at University of Maryland and Ruhr University Bochum.

building block for *state machine replication*, and has received renewed attention in recent years for its applications to blockchains and cryptocurrencies.

Different network models for atomic broadcast can be considered. In a *synchronous* network [2,8,15,18,28], all messages are delivered within some known time Δ. In an *asynchronous* network [16,25], messages can be delayed arbitrarily. (Some work assumes the *partially synchronous* model [13], where messages are delivered within some time bound Δ that is unknown to the parties. We do not consider this model in our work.) Assuming a public-key infrastructure (PKI), atomic broadcast is feasible for $t_s < n$ adversarial corruptions in a synchronous network, but only for $t_a < n/3$ faults in an asynchronous network. A natural question is whether it is possible to design a protocol that can withstand strictly more than $n/3$ faults if the network happens to be synchronous, without entirely sacrificing security if the network happens to be asynchronous. More precisely, fix two thresholds t_a, t_s with $t_a \leq t_s$. Is it possible to design a *network-agnostic* atomic broadcast protocol that (1) tolerates t_s corruptions if it is run in a synchronous network and (2) tolerates t_a corruptions if it is run in an asynchronous network? Depending on one's assumptions about the probabilities of different events, a network-agnostic protocol could be preferable to either a purely synchronous protocol (which loses security if the network is asynchronous) or a purely asynchronous one (which loses security if there are $n/3$ or more faults).

We settle the above question in a model where there is a trusted dealer who distributes information to the parties in advance of the protocol execution:

- We present an atomic broadcast protocol, TARDIGRADE,[1] that achieves the above for any t_a, t_s satisfying $t_a + 2t_s < n$. We also prove that no atomic broadcast protocol can provide the above guarantees[2] if $t_a + 2t_s \geq n$, and so TARDIGRADE is optimal in terms of the thresholds it tolerates.
- We also describe a second protocol, UPGRADE, that is sub-optimal in terms of t_a, t_s but has asymptotic communication complexity comparable to state-of-the-art asynchronous atomic broadcast protocols (see Table 1).

Our work is inspired by work of Blum et al. [5], who show analogous results (with the same thresholds) for the simpler problem of Byzantine agreement. We emphasize that ABC is not realized by simply repeating a (multi-valued) BA protocol multiple times. In particular, the validity property of BA guarantees only that if a value is used as input by all honest parties then that transaction will be output by all honest parties. In the context of ABC, however, each honest party holds a local buffer containing multiple values called *transactions*. Transactions may arrive at arbitrary times, and there is no way to ensure that all honest parties will input the same transactions to some execution of an underlying BA protocol. (Although generic transformations from BA to ABC are known

[1] Tardigrades, also called water bears, are microscopic animals known for their ability to survive in extreme environments.

[2] This does not contradict the existence of synchronous ABC protocols for $t_s < n$, since such protocols are insecure in an asynchronous setting even if no parties are corrupted.

Protocol	Communication	Network model
HoneyBadger [25]	$O(n \cdot \|\mathsf{tx}\|)$	Asynchronous
BEAT1 / BEAT2 [12]	$O(n^2 \cdot \|\mathsf{tx}\|)$	Asynchronous
Dumbo1 / Dumbo2 [16]	$O(n \cdot \|\mathsf{tx}\|)$	Asynchronous
TARDIGRADE	$O(n^2 \cdot \|\mathsf{tx}\|)$	Network-agnostic
UPGRADE	$O(n \cdot \|\mathsf{tx}\|)$	Network-agnostic

Table 1. Per-transaction communication complexity of ABC protocols, for transactions of length $|\mathsf{tx}|$, assuming infinite block size and suppressing dependence on the security parameter for simplicity.

in other settings [9], no such transformation is known for the network-agnostic setting we consider.) Indeed, translating the approach of Blum et al. from BA to ABC introduces several additional challenges. In particular, as just noted, in the context of atomic broadcast there is no guarantee that honest parties ever use the same transaction, making it more challenging to prove liveness. A central piece of our construction is a novel protocol for the fundamental problem of *asynchronous common subset* (ACS). Our ACS protocol achieves non-standard security properties that turn out to be generally useful for constructing protocols in a network-agnostic setting; it has already served as a crucial ingredient in follow-up work [6] on network-agnostic secure computation.

1.1 Related Work

There is extensive prior work on both Byzantine agreement and atomic broadcast/SMR/blockchain protocols; we do not provide an exhaustive survey, but instead focus only on the most closely related works.

Miller et al. [25] already note that well-known SMR protocols that tolerate malicious faults (e.g., [8,18]) fail to achieve liveness in an asynchronous network. The HoneyBadger protocol [25] is designed for asynchronous networks, but only handles $t < n/3$ faults even if the network is synchronous.

Several of the most prominent blockchain protocols rely on synchrony [15,28]; Nakamoto consensus, in particular, relies on the assumption that messages will be delivered much faster than the time required to solve proof-of-work puzzles, and is insecure if the network latency is too high or nodes become (temporarily) partitioned from the network.

We focus on designing a single protocol that may be run in either a synchronous or asynchronous network while providing security guarantees in either case. Related work includes that of Malkhi et al. [24] and Momose and Ren [26], who consider networks that may be either synchronous or *partially* synchronous; Liu et al. [21], who design a protocol that tolerates a minority of malicious faults in a synchronous network and a minority of *fail-stop* faults in an asynchronous network; and Guo et al. [17] and Abraham et al. [2], who consider temporary disconnections between two synchronous network components.

A different line of work [22,23,29,30] designs protocols with good *responsiveness*. Roughly, such protocols still require synchrony, but terminate in time proportional to the actual message-delivery time δ rather than the upper bound on the network-delivery time Δ. Kursawe [19] gives a protocol for an asynchronous network that terminates more quickly if the network is synchronous (but does not tolerate more faults in that case). Finally, other work [3,11,14,31] considers a model where synchrony is available for some (known) period of time, and the network is asynchronous afterward.

1.2 Paper Organization

We describe our model in Section 2, and give formal definitions in Section 3. In Section 4, we describe a protocol for the asynchronous common subset (ACS) problem. Then, in Section 5, we show how to construct a network-agnostic atomic broadcast protocol (TARDIGRADE) achieving optimal security tradeoffs using ACS and other building blocks. In Section 6, we present a second atomic broadcast protocol (UPGRADE) that achieves per-transaction communication complexity linear in n at the cost of tolerating fewer corruptions. Additional constructions, formal proofs, and supplementary results are included in the full version of the paper.[3]

2 Model

We consider protocols run by n parties $P_1, \ldots, P_n$, over point-to-point authenticated channels. Some fraction of these parties are controlled by an adversary, and may deviate arbitrarily from the protocol. For simplicity, we generally assume a static adversary who corrupts parties prior to the start of the protocol; in Section 5.6, however, we do briefly discuss how TARDIGRADE can be modified to tolerate an adaptive adversary who may corrupt parties as the protocol is executed. Parties who are not corrupted are called *honest*.

In our model, the network has two possible states. The state is fixed prior to the beginning of the execution; however, the state is not known to the honest parties. When the network is *synchronous*, all parties begin the protocol at the same time, parties' clocks progress at the same rate, and all messages are delivered within some known time Δ after they are sent. The adversary is able to adaptively delay and reorder messages arbitrarily (subject to the bound Δ). When the network is *asynchronous*, the adversary is able to delay messages for arbitrarily long periods of time (as long as all messages are eventually delivered). The parties still have local clocks in the asynchronous setting; however, in this case their clocks are only assumed to be monotonically increasing. In particular, parties' clocks are not necessarily synchronized, and they may start the protocol at different times.

[3] Available at: eprint.iacr.org/2020/142.pdf

We assume the network is either synchronous or asynchronous for the lifetime of the protocol. A more general model would consider a network that alternates between periods of synchrony and asynchrony. Our adaptively secure protocol (cf. Section 5.6) tolerates an asynchronous network that later becomes synchronous so long as the attacker does not exceed t_a corruptions until all iterations initiated while the network was asynchronous are complete, and does not exceed t_s corruptions overall. Handling a synchronous network that later becomes asynchronous is only interesting if some mechanism is provided to "uncorrupt" parties (as in the proactive setting). This is outside our model, and we leave treatment of this case as an interesting direction for future work.

We assume a trusted *dealer* who initializes parties with some information prior to execution of the protocol. Specifically, we assume the dealer distributes keys for threshold signature and encryption schemes, each secure for up to t_s corruptions. In a threshold signature scheme there is a public key pk, private keys $sk_1, \ldots, sk_n$, and (public) signature verification keys $(pk_1, \ldots, pk_n)$. Each party P_i receives sk_i, pk, and $(pk_1, \ldots, pk_n)$, and can use its secret key sk_i to create a signature share σ_i on a message m. A signature share from party P_i on a message m can be verified using the corresponding public verification key pk_i (and is called *valid* if it verifies successfully); for this reason, we can also view such a signature share as a signature by P_i on m. We often write $\langle m \rangle_i$ as a shorthand for the tuple (i, m, σ_i), where σ_i is a valid signature share on m with respect to P_i's verification key, and implicitly assume that invalid signature shares are discarded. A set of $t_s + 1$ valid signature shares on the same message can be used to compute a signature for that message, which can be verified using the public key pk; a signature σ on a message m is called *valid* if it verifies successfully with respect to pk. We always implicitly assume that parties use some form of domain separation when signing to ensure that signature shares are valid only in the context in which they are generated.

In a threshold encryption scheme, there is a public encryption key ek, (private) decryption keys $dk_1, \ldots, dk_n$, and public verification keys $vk_1, \ldots, vk_n$ that can be used, as above, to verify that a decryption share is correct (relative to a particular ciphertext). A party P_i can use its decryption key dk_i to generate a decryption share of a ciphertext c; any set of $t_s + 1$ correct decryption shares enable recovery of the underlying message m. Security requires that no collection of t_s parties can decrypt on their own.

We idealize the threshold signature and encryption schemes for simplicity, but they can be instantiated using any of several known protocols; in particular, we only require CPA-security for the threshold encryption scheme. We assume that signature shares and signatures have size $O(\kappa)$, where κ is the security parameter; this is easy to ensure using a collision-resistant hash function. We assume that encrypting a message m of length $|m|$ produces a ciphertext of length $|m| + O(\kappa)$, and that decryption shares have length $O(\kappa)$; these are easy to ensure using standard KEM/DEM mechanisms.

3 Definitions

In this section, we formally define atomic broadcast and relevant subprotocols. Throughout, when we say a protocol achieves some property, we include the case where it achieves that property with overwhelming probability in a security parameter κ. Additionally, in some cases we consider protocols where parties may not terminate even upon generating output; for this reason, we mention termination explicitly in our definitions when applicable.

Many of the definitions below are parameterized by a threshold t. This will become relevant in later sections, where we will often analyze a protocol's properties in a synchronous network with t_s corruptions, as well as in an asynchronous network with t_a corruptions.

3.1 Broadcast and Byzantine Agreement

A *reliable broadcast* protocol allows parties to agree on a value chosen by a designated sender. Honest parties are not guaranteed to terminate; hence, reliable broadcast is weaker than standard broadcast. However, if there is some honest party who terminates, then all honest parties terminate.

Definition 1 (Reliable broadcast). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender $P^* \in \{P_1, \ldots, P_n\}$ begins holding input v^* and parties terminate upon generating output.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t parties are corrupted: if P^* is honest, then every honest party outputs v^*.*
- **Consistency:** *Π is t-*consistent *if the following holds whenever at most t parties are corrupted: either no honest party outputs a value, or all honest parties output the same value v.*

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

We reserve the term "broadcast" for reliable broadcast. When a party P_i sends a message m to all parties (over point-to-point channels), we say that P_i *multicasts* m.

Byzantine agreement (BA) is closely related to broadcast. In a BA protocol, there is no designated sender; instead, each party has their own input and the parties would like to agree on an output.

Definition 2 (Byzantine agreement). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i begins holding input $v_i \in \{0, 1\}$.*

- **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.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, every honest party outputs the same value $v \in \{0, 1\}$.*
- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party terminates with some output in $\{0, 1\}$.*

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

3.2 Asynchronous Common Subset

Informally, a protocol for the *asynchronous common subset* (ACS) problem [4] allows n parties, each with some input, to agree on a subset of those inputs. (The term "asynchronous" in the name is historical, and one can also consider protocols for this task in the synchronous setting.)

Definition 3 (ACS). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each P_i begins holding input $v_i \in \{0,1\}^*$, and parties output sets of size at most n.*

- **Validity:** *Π is t-*valid *if the following holds whenever at most t parties are corrupted: if every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$.*
- **Consistency:** *Π is t-*consistent *if whenever at most t parties are corrupted, all honest parties output the same set S.*
- **Liveness:** *Π is t-*live *if whenever at most t parties are corrupted, every honest party generates output.*

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

For our analysis, it will be helpful to define a few additional properties.

Definition 4 (ACS properties). *Let Π be as above.*

- **Set quality:** *Π has t-*set quality *if the following holds whenever at most t parties are corrupted: if an honest party outputs a set S, then S contains the input of at least one honest party.*
- **Validity with termination:** *Π is t-*valid with termination *if, whenever at most t parties are corrupted and every honest party's input is equal to the same value v, then every honest party outputs $\{v\}$ and terminates.*
- **Termination:** *Π is t-*terminating *if whenever at most t parties are corrupted, every honest party generates output and terminates.*

3.3 Atomic Broadcast

Protocols for *atomic broadcast* (ABC) allow parties to maintain agreement on an ever-growing, ordered log of *transactions*. An atomic broadcast protocol does not terminate but instead continues indefinitely. We model the local log held by each party P_i as a write-once array $\mathsf{Blocks}_i = \mathsf{Blocks}_i[1], \mathsf{Blocks}_i[2], \ldots$. Each $\mathsf{Blocks}_i[j]$ is initially set to a special value $\perp$. We say that P_i *outputs a block in iteration j* when P_i writes a set of transactions to $\mathsf{Blocks}_i[j]$; similarly, for each i, j such that $\mathsf{Blocks}_i[j] \neq \perp$, we refer to $\mathsf{Blocks}_i[j]$ as the *block output by P_i in iteration j*. For convenience, we let $\mathsf{Blocks}_i[k : \ell]$ denote the contiguous subarray $\mathsf{Blocks}_i[k], \ldots, \mathsf{Blocks}_i[\ell]$ and let $\mathsf{Blocks}_i[: \ell]$ denote the prefix $\mathsf{Blocks}_i[1 : \ell]$.

For simplicity, we imagine that each party P_i has a local buffer buf_i, and that transactions are added to parties' local buffers by some mechanism external to the protocol (e.g., via a gossip protocol). Whenever P_i outputs a block, they delete from their buffer any transactions that have already been added to their

log. We emphasize that a particular transaction tx may be provided as input to different parties at arbitrary times, and may be provided as input to some honest parties but not others.

Definition 5 (Atomic broadcast). *Let Π be a protocol executed by parties $P_1, \ldots, P_n$ who are provided with transactions as input and locally maintain arrays* Blocks *as described above.*

- **Completeness:** *Π is t-*complete *if the following holds whenever at most t parties are corrupted: for all $j > 0$, every honest party outputs a block in iteration j.*
- **Consistency:** *Π is t-*consistent *if the following holds whenever at most t parties are corrupted: if an honest party outputs a block B in iteration j then all honest parties output B in iteration j.*
- **Liveness:** *Π is t-*live *if the following holds whenever at most t parties are corrupted: if every honest party is provided a transaction tx as input, then every honest party eventually outputs a block that contains tx.*

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

In the above definition, a transaction tx is only guaranteed to be contained in a block output by an honest party if *every* honest party receives tx as input. A stronger definition might require that a transaction is output even if only a *single* honest party receives tx as input; however, it is easy to achieve the latter from the former by requiring honest parties to forward new transactions they receive to the rest of the parties in the network.

4 ACS with Higher Validity Threshold

A key component of our atomic broadcast protocol is an ACS protocol for asynchronous networks that is secure when the number of corrupted parties is below a fixed threshold t_a, and guarantees validity up to a higher threshold t_s. More precisely, fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$; we show a t_a-secure ACS protocol that achieves t_a-termination, t_s-validity with termination, and t_a-set quality. Throughout this section, we assume an asynchronous network. (Of course, the protocol achieves the same guarantees in a synchronous network.)

Our protocol is adapted from the ACS protocol of Ben-Or et al. [4] (later adapted by Miller et al. [25]), which is built using subprotocols for reliable broadcast and Byzantine agreement. We present our construction in two steps: first, we describe an ACS protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ (cf. Figure 1) that is t_a-secure and has t_a-set quality, but is non-terminating. Then, we construct a second protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ (cf. Figure 2) that uses $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ as a subprotocol. $\Pi^{t_a,t_s}_{\mathsf{ACS}}$ inherits security and set quality from $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$, and additionally achieves t_a-termination and t_s-validity with termination.

Protocol $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. At a high level, an execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ involves one instance of reliable broadcast and one instance of Byzantine agreement per party P_i,

denoted Bcast_i and BA_i, respectively. Informally, Bcast_i is used to broadcast P_i's input v_i, and BA_i is used to determine whether P_i's input will be included in the final output. When a party receives output v_i' from Bcast_i, they input 1 to BA_i. Once a party has received output from $n-t_a$ broadcasts, they input 0 to any BA instances they have not yet initiated. Each party keeps track of which BA instances have output 1 using a local variable $S^* := \{i : \mathsf{BA}_i \text{ output } 1\}$. At the end of the protocol, if a party observes a majority value v in the set of values $\{v_i'\}_{i \in S^*}$, it outputs the singleton set $\{v\}$; otherwise, it outputs $\{v_i'\}_{i \in S^*}$, i.e., the set of all values broadcast by parties in S^*.

We assume an ABA subprotocol that is secure for $t_a < n/3$ corruptions and has communication complexity $O(n^2)$, such as the ABA protocol of Mostéfaoui et al. [27]. We also assume an asynchronous reliable broadcast protocol Bcast that is t_s-valid and t_a-consistent with communication complexity $O(n^2\,|v|)$. It is straightforward to adapt Bracha's (asynchronous) reliable broadcast protocol [7] to achieve these properties; an example construction can be found in the full version of the paper.

$\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$

- Set $\mathsf{commit} := \mathsf{false}$ and $S^* := \emptyset$.
- Run Bcast_j as the sender with input v_j, and for each $i \neq j$ run Bcast_i with P_i as the sender.
- Upon Bcast_i terminating with output v_i': if P_j has not yet begun running BA_i then begin running it with input 1.
- Upon BA_i terminating with output 1: add i to S^*.
- Upon setting $|S^*|$ to $n - t_a$: for any BA_i that P_j has not yet begun running, begin running BA_i with input 0.

Predicates:

$C_1(v)$: at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ have output v.
C_1: $\exists v$ for which $C_1(v)$ is true.
$C_2(v)$: $|S^*| \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i \in [n]}$ have terminated, and a strict majority of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ have output v.
C_2: $\exists v$ for which $C_2(v)$ is true.
C_3: $|S^*| \geq n - t_a$, all executions $\{\mathsf{BA}_i\}_{i \in [n]}$ have terminated, and all executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ have terminated.

Output conditions:

(Event 1) If $C_1(v) = \mathsf{true}$ for some v and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.
(Event 2) If $C_1 = \mathsf{false}$, $C_2(v) = \mathsf{true}$ for some v, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v\}$.
(Event 3) If $C_1 = C_2 = \mathsf{false}$, $C_3 = \mathsf{true}$, and $\mathsf{commit} = \mathsf{false}$ then:
set $\mathsf{commit} := \mathsf{true}$ and output $\{v_i'\}_{i \in S^*}$.

Fig. 1. An ACS protocol, from the perspective of party P_j with input v_j.

Lemma 1. *Fix t_s, t_a with $t_a + 2 \cdot t_s < n$, and assume there are at most t_s corrupted parties during some execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. If an honest party P_i outputs a set S_i, then $\exists v_j \in S_i$ such that v_j was input by an honest party P_j.*

Proof. We show that P_i's output S_i always includes a value that was output from an execution of Bcast where the corresponding sender is honest. The lemma follows from t_s-validity of Bcast.

Suppose P_i generates output due to Event 1, so S_i is a singleton set $\{v\}$. P_i must have received v as output from at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$. Because $n - 2t_s > t_a \geq 0$, at least one of those corresponds to an honest sender.

Next, suppose P_i generates output due to Event 2. Again, S_i is a singleton set $\{v\}$. P_i must have seen at least $\lfloor \frac{|S^*|}{2} \rfloor + 1$ broadcast instances terminate with output v, and furthermore $|S^*| \geq n - t_a$. Therefore, P_i has seen at least $\lfloor \frac{n-t_a}{2} \rfloor + 1 \geq \lfloor \frac{2t_s}{2} \rfloor + 1 > t_s$ executions of $\{\mathsf{Bcast}_i\}$ terminate with output v. Since there are at most t_s corrupted parties, at least one of those executions must correspond to an honest sender.

Finally, suppose P_i generates output due to Event 3, so $S_i = \{v'_i\}_{i \in S^*}$. Since there are at most t_s corrupted parties and $|S^*| - t_s \geq n - t_a - t_s > t_s \geq 0$, at least one party in S^* is honest. □

Lemma 2. *If $t_a + 2 \cdot t_s < n$, then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ is t_s-valid.*

Proof. Assume at most t_s parties are corrupted, and all honest parties have the same input v. By t_s-validity of Bcast, at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ (namely, those for which the sender is honest) will eventually output v. It follows that all honest parties eventually set $C_1(v) = \mathsf{true}$, at which point they will output $\{v\}$ if they have not already generated output. It only remains to show that there is no other set an honest party can output.

If an honest party generates output S due to Events 1 or 2, then S is a singleton set. Since all honest parties have input v, Lemma 1 implies $S = \{v\}$.

To conclude, we show that no honest party can generate output due to Event 3. Assume toward a contradiction that some honest party P generates output due to Event 3. Then P must have seen Bcast_i terminate (say, with output v_i) for all $i \in S^*$. Since also $|S^*| \geq n - t_a > 2t_s$, a majority of those executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ correspond to honest senders and so (by t_s-validity of Bcast) resulted in output v. But then $C_2(v)$ would be true for P, and P would not generate output due to Event 3. □

Lemma 3. *Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$, and assume at most t_a parties are corrupted during an execution of $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$. If honest parties P_1, P_2 output sets S_1, S_2, respectively, then $S_1 = S_2$.*

Proof. Say P_1 generates output due to event i and P_2 generates output due to event j, and assume without loss of generality that $i \leq j$. We consider the different possibilities.

First, assume $i = 1$ so Event 1 occurs for P_1 and $S_1 = \{v_1\}$ for some value v_1. We have the following sub-cases:

- If Event 1 also occurs for P_2, then $S_2 = \{v_2\}$ for some value v_2. P_1 and P_2 must have each seen some set of at least $n - t_s > n/2$ executions of $\{\mathsf{Bcast}_i\}$ output v_1 and v_2, respectively. The intersection of these sets is non-empty; thus, t_a-consistency of Bcast implies that $v_1 = v_2$ and hence $S_1 = S_2$.
- If Event 2 occurs for P_2, then once again $S_2 = \{v_2\}$ for some v_2. P_2 must have $|S^*| \geq n - t_a$, and must have seen at least $\left\lfloor \frac{|S^*|}{2} \right\rfloor + 1 \geq \left\lfloor \frac{n-t_a}{2} \right\rfloor + 1$ executions of $\{\mathsf{Bcast}_i\}$ output v_2. Moreover, P_1 must have seen at least $n - t_s$ executions of $\{\mathsf{Bcast}_i\}$ output v_1. Since

$$n - t_s + \left\lfloor \frac{n - t_a}{2} \right\rfloor + 1 \geq n - t_s + \left\lfloor \frac{2t_s}{2} \right\rfloor + 1 > n, \tag{1}$$

  these two sets of executions must have a non-empty intersection. But then t_a-consistency of Bcast implies that $v_1 = v_2$ and hence $S_1 = S_2$.
- If Event 3 occurs for P_2 then P_2 must have seen all executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ terminate, where $|S^*| \geq n - t_a$. We know P_1 has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ output v_1, and so (by t_a-consistency of Bcast) there are at most t_s executions $\{\mathsf{Bcast}_i\}_{i \in [n]}$ that P_2 has seen terminate with a value other than v_1. The number of executions of $\{\mathsf{Bcast}_i\}_{i \in S^*}$ that P_2 has seen terminate with output v_1 (which is at least $(n - t_a) - t_s > t_s$) is thus strictly greater than the number of executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ that P_2 has seen terminate with a value other than v_1 (which is at most t_s). But then $C_2(v_1)$ would be true for P_2. We conclude that Event 3 cannot occur for P_2.

Next, assume $i = j = 2$, so Event 2 occurs for P_1 and P_2. Then $S_1 = \{v_1\}$ and $S_2 = \{v_2\}$ for some v_1, v_2. Both P_1 and P_2 must have seen all executions $\{\mathsf{BA}_i\}_{i \in [n]}$ terminate. By t_a-consistency of BA, they must therefore agree on S^*. P_1 must have seen a majority of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v_1; similarly, P_2 must have seen a majority of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v_2. Then t_a-consistency of Bcast implies $v_1 = v_2$.

Finally, consider the case where $j = 3$ (so Event 3 occurs for P_2) but $i > 1$ (so P_1 generates output due either to Event 2 or 3). As above, t_a-consistency of BA ensures that P_1 and P_2 agree on S^*. Moreover, P_2 must have seen all executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ terminate, but without any value being output by a majority of those executions. But then t_a-consistency of Bcast implies that P_1 also does not see any value being output by a majority of those executions, and so Event 2 cannot occur for P_1; thus, Event 3 must have occurred for P_1. Therefore, t_a-consistency of Bcast implies that P_1 outputs the same set as P_2. □

Lemma 4. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi^{t_a,t_s}_{\mathsf{ACS}^*}$ is t_a-live.*

Proof. It follows easily from t_a-security of Bcast and BA that if any honest party generates output then all honest parties generate output, so consider the case where no honest parties have (yet) generated output. Let H denote the indices of the honest parties. By t_s-validity of Bcast, all honest parties eventually see the executions $\{\mathsf{Bcast}_i\}_{i \in H}$ terminate, and so all honest parties input a value to the executions $\{\mathsf{BA}_i\}_{i \in H}$. By t_a-security of BA, all honest parties eventually see those executions terminate and agree on their outputs. There are now two cases:

- If all executions $\{\mathsf{BA}_i\}_{i \in H}$ output 1, then it is immediate that all honest parties have $|S^*| \geq n - t_a$.
- If BA_i outputs 0 for some $i \in H$, then (by t_a-validity of BA) some honest party P must have used input 0 when running BA_i. But then P must have seen at least $n - t_a$ other executions $\{\mathsf{BA}_i\}$ output 1. By t_a-consistency of BA, this implies that all honest parties see at least $n - t_a$ executions $\{\mathsf{BA}_i\}$ output 1, and hence have $|S^*| \geq n - t_a$.

Since all honest parties have $|S^*| \geq n - t_a$, they all execute $\{\mathsf{BA}_i\}_{i \in [n]}$. Once again, t_a-termination of BA implies that all those executions will eventually terminate. Finally, if $i \in S^*$ for some honest party P then P must have seen BA_i terminate with output 1; then t_a-validity of BA implies that some honest party used input 1 when running BA_i and hence has seen Bcast_i terminate. It follows that P will see Bcast_i terminate. As a result, we see that every honest party can (at least) generate output due to Event 3. □

Lemma 5. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi_{\mathsf{ACS}^*}^{t_a, t_s}$ has t_a-set quality.*

Proof. If an honest party P outputs $S = \{v\}$ due to Event 1, then P has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > 0$ must correspond to honest senders. By t_s-validity of Bcast, those honest parties must have all had input v, and so set quality holds. Alternatively, say P outputs a set $\{v\}$ due to Event 2. Then P must have $|S^*| \geq n - t_a$, and at least $\lfloor \frac{|S^*|}{2} \rfloor + 1 \geq \lfloor \frac{n - t_a}{2} \rfloor + 1 > t_a$ of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v. At least one of those executions must correspond to an honest party, and that honest party must have had input v (by t_s-validity of Bcast); thus, set quality holds. Finally, if P output a set S due to Event 3, then S contains every value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$ with $|S^* \geq n - t_a$. Since S^* must contain at least one honest party, set quality follows as before. □

Theorem 1. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ACS}^*}^{t_a, t_s}$ is t_a-secure and t_s-valid, and has t_a-set quality.*

Proof. Lemma 2 proves t_s-validity. Lemmas 3 and 4 together prove t_a-liveness and t_a-consistency, and Lemma 6 proves t_a-set quality. □

Lemma 6. *If $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, then $\Pi_{\mathsf{ACS}^*}^{t_a, t_s}$ has t_a-set quality.*

Proof. If an honest party P outputs $S = \{v\}$ due to Event 1, then P has seen at least $n - t_s$ executions $\{\mathsf{Bcast}_i\}$ terminate with output v. Of these, at least $n - t_s - t_a > 0$ must correspond to honest senders. By t_s-validity of Bcast, those honest parties must have all had input v, and so set quality holds. Alternatively, say P outputs a set $\{v\}$ due to Event 2. Then P must have $|S^*| \geq n - t_a$, and at least $\lfloor \frac{|S^*|}{2} \rfloor + 1 \geq \lfloor \frac{n - t_a}{2} \rfloor + 1 > t_a$ of the executions $\{\mathsf{Bcast}_i\}_{i \in S^*}$ output v. At least one of those executions must correspond to an honest party, and that honest party must have had input v (by t_s-validity of Bcast); thus, set quality holds. Finally, if P output a set S due to Event 3, then S contains every value output by $\{\mathsf{Bcast}_i\}_{i \in S^*}$. Since $|S^*| \geq n - t_a$, S^* must contain at least one honest party, and so set quality follows as before. □

Protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Protocol $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ does not guarantee termination. We transform $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ to a terminating ACS protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using digital signatures. The parties first run $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$. When a party P_i generates output S_i from that protocol, it then notifies the other parties by multicasting a signature share$\langle \mathsf{commit}, S_i \rangle_i$ on S_i. Any party who receives enough signature shares to form a signature—or receives a signature directly—multicasts the signature to all other parties, outputs the corresponding set, and terminates.

$\Pi_{\mathsf{ACS}}^{t_a,t_s}$

- Run $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ using input v_j.
- Upon receiving output S_j from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$, multicast $\langle \mathsf{commit}, S_j \rangle_j$.
- Upon receiving $t_s + 1$ signature shares of (commit, S), form a signature σ on (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.
- Upon receiving a valid signature σ of (commit, S), multicast $(\mathsf{commit}, S, \sigma)$, output S, and terminate.

Fig. 2. A terminating ACS protocol, from the perspective of party P_j with input v_j.

Lemma 7. *$\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is t_a-terminating.*

Proof. If one honest party terminates $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ then all honest parties will eventually receive a valid signature and thus terminate $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. But as long as no honest parties has yet terminated, t_a-liveness of $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ implies that all honest parties will generate output from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$; moreover, t_a-consistency of $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ implies that all those outputs will be equal to the same set S. So the $n - t_a \geq t_s + 1$ honest parties will send signature shares on S to all parties, which means that all honest parties will terminate. □

Lemma 8. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is t_a-secure, t_a-terminating, and t_s-valid with termination, and has t_a-set quality.*

Proof. Lemma 7 implies that $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is t_a-live as well as t_a-terminating. If an honest party outputs a set S from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, then (as long as at most t_s parties are corrupted) at least one honest party must have output S from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$. Thus, $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ inherits t_a-set quality, t_a-consistency, and t_s-validity (without termination) from $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ (cf. Theorem 1). It is straightforward to extend t_s-validity to t_s-validity with termination using an identical argument as in Lemma 7. □

Communication complexity of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Let $|v|$ be the size of each party's input. Recall that each instance of Bcast has communication complexity $O(n^2 |v|)$, and each instance of BA has cost $O(n^2)$. Since the inner protocol $\Pi_{\mathsf{ACS}^*}^{t_a,t_s}$ consists of n parallel instances of Bcast and BA, the cost of the inner protocol is $O(n^3 |v|)$. In the remaining steps, each party sends a set of size at most n plus a signature share (or signature) to everyone else, contributing an additional $O(n^2 \cdot (n |v| + \kappa))$ communication. The total communication for $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is thus $O(n^3 |v| + n^2 \kappa)$.

5 Network-Agnostic Atomic Broadcast

In this section, we show our main result: for any $t_s \geq t_a$ with $t_a + 2t_s < n$, an atomic broadcast protocol that is t_s-secure in a synchronous network and t_a-secure in an asynchronous network.

5.1 Technical Overview

At a high level, each iteration of the protocol consists of four main steps. First, there is an information-gathering phase in which each party sends its input to all other parties, and waits for a fixed amount of time to receive inputs from others. Any party who receives enough inputs during the first phase will use them as input to a synchronous *block agreement* (BLA) protocol $\Pi_{\mathsf{BLA}}^{t_s}$. If the network is synchronous and at most t_s parties are corrupted, the BLA subprotocol will output a set of inputs that contains sufficiently many honest parties' inputs. The BLA subprotocol is run for a fixed amount of time, with the timeout chosen to ensure that (with high probability) it will terminate before the timeout if the network is synchronous. This brings us to the third phase, in which parties run the ACS protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. If a party received output from the BLA protocol before the timeout, they will use that as their input to the ACS subprotocol; otherwise, they wait until they have received sufficiently many inputs from other parties and use those. The final phase occurs once parties have received output from the ACS protocol. The parties use that output to form the next block.

The BLA and ACS protocols are designed to have complementary security properties. In particular, if the network is synchronous then the BLA protocol will ensure that all honest parties use the same input value B in the ACS protocol. This is exactly why $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ has t_s-validity with termination: so that that, in this case, all parties will be in agreement on the singleton set $\{B\}$ after running $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. On the other hand, if the network is not synchronous and at most t_a parties are corrupted, it is possible that $\Pi_{\mathsf{BLA}}^{t_s}$ will not succeed, and parties may input different values to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. However, in this case t_a-security of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ ensures that the parties will agree on a set of values $B = \{\beta_1, \beta_2, \dots\}$. Moreover, the output-quality property ensures that at least a constant fraction of the values in B were contributed by honest parties.

5.2 Block Agreement

We use a block-agreement protocol to agree on objects that we call *pre-blocks*. (The name alludes to their role in our eventual atomic broadcast protocol, where they will serve as an intermediate between parties' raw inputs and the final blocks.) A pre-block is a vector of length n whose ith entry is either $\perp$ or a message along with a valid signature by P_i on that message. The *quality* of a pre-block is defined as the number of entries that are not $\perp$; we say that a pre-block is a *k-quality pre-block* if it has quality at least k.

Definition 6 (Block agreement). *Let Π be a protocol executed by parties $P_1, \dots, P_n$, where parties terminate upon generating output.*

- **Validity:** *Π is t-*valid *if whenever at most t of the parties are corrupted and every honest party's input is an $(n-t)$-quality pre-block, then every honest party outputs an $(n-t)$-quality pre-block.*
- **Consistency:** *Π is t-*consistent *if whenever at most t of the parties are corrupted, every honest party outputs the same pre-block B.*

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

A synchronous block-agreement protocol can be constructed using a straightforward adaptation of the *synod protocol* by Abraham et al. [1]. (For completeness, a construction and security analysis can be found in the full version.)

Theorem 2. *Fix a maximum input length $|m|$. There is a block-agreement protocol Π_{BLA} with communication complexity $O(n^3\kappa^2 + n^2\kappa|m|)$ that is t-secure for any $t < n/2$ when run in a synchronous network and terminates in time $5\kappa\Delta$.*

5.3 A Network-Agnostic Atomic Broadcast Protocol

We now describe our atomic broadcast protocol TARDIGRADE (cf. Figure 3), parameterized by thresholds t_s and t_a. Let L denote a desired maximum *block size*, i.e., the maximum number of transactions that can appear in a block. At a high level, parties agree on each new block via the following steps. First, each party P_i chooses a set V_i of L/n transactions from among the first L transactions in its local buffer. (We assume without loss of generality that parties always have at least L transactions in their buffer, since they can always pad their buffers with null transactions.) Next, P_i encrypts V_i using a (t_s, n)-threshold encryption scheme to give a ciphertext μ_i. (As in HoneyBadger [25], transactions are encrypted to limit the adversary's ability to selectively censor certain transactions.) Each party signs its ciphertext and multicasts it, then waits for a fixed period of time to receive signed ciphertexts from the other parties. Whenever a party receives a signed ciphertext during this time, they add it to a pre-block. Any party who forms an $(n-t_s)$-quality pre-block in this way within the time limit will input that pre-block to Π_{BLA}. The parties then wait for another fixed period of time to see whether Π_{BLA} outputs an $(n-t_s)$-quality pre-block. If a party receives an $(n-t_s)$-quality pre-block as output from Π_{BLA} within this time limit, it inputs that pre-block to the ACS protocol $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Otherwise, if some party does not receive suitable output within the time limit, it inputs a pre-block containing the signed ciphertexts it received directly from other parties. (In this case, if a party has not received enough signed ciphertexts to form an $(n-t_s)$-quality pre-block, it waits for additional ciphertexts to arrive before inputting its pre-block to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$.) At this point, each party waits for $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ to output a set of pre-blocks. The output of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ is passed into a subroutine ConstructBlock that performs threshold decryption for each ciphertext in each pre-block in the set, and combines the resulting transactions into a final block.

Each party begins iteration k when its local clock reaches time $T_k := \lambda \cdot (k-1)$, where $\lambda > 0$ is a *spacing parameter*. (The value of λ is irrelevant for the security proofs, but can be tuned to achieve better performance in practice; see further

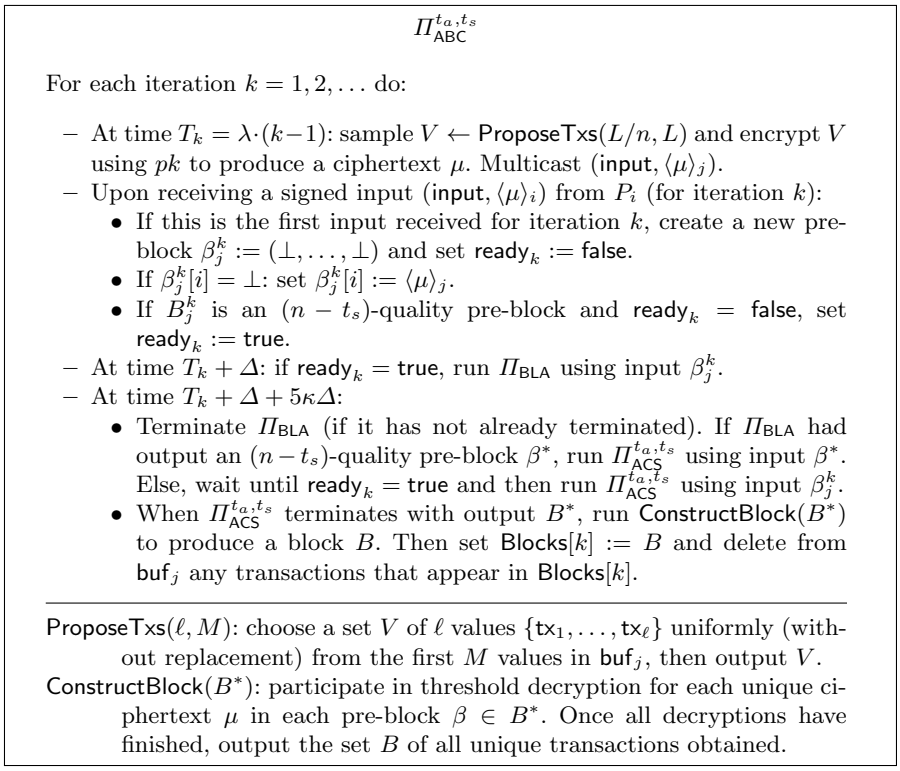

Fig. 3. Our atomic broadcast protocol TARDIGRADE, from the perspective of party P_j.

discussion in Section 5.4.) If the network is synchronous, parties' clocks are synchronized and so all parties begin each iteration at the same time. If the network is asynchronous, we do not have this guarantee. In either case, parties do not necessarily finish agreeing on block k prior to starting iteration $k' > k$, and so it is possible for parties to be participating in several iterations in parallel.

We implicitly assume that messages in each iteration, including messages corresponding to the various subprotocols, carry an identifier for the corresponding iteration so that parties know the iteration to which it belongs. Importantly, the executions of Π_{BLA} and $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ associated with a particular iteration are entirely separate from those of other iterations.

Theorem 3 (Completeness and consistency). *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ABC}}^{t_a,t_s}$ is t_a-complete/consistent when run in an asynchronous network, and t_s-complete/consistent when run in a synchronous network.*

Proof. First, consider the case where at most t_s parties are corrupted and the network is synchronous. In the beginning of each iteration k, each honest party multicasts a set of transactions, and so every honest party can form an $(n - t_s)$-quality pre-block by time $T_k + \Delta$. Thus, every honest party starts running Π_{BLA}

at time $T_k + \Delta$ using an $(n - t_s)$-quality pre-block as input. By t_s-security of Π_{BLA} in a synchronous network (note $t_s < n/2$), with overwhelming probability every honest party outputs the same $(n - t_s)$-quality pre-block β^* from Π_{BLA} by time $T_k + \Delta + 5\kappa\Delta$. Therefore, each honest party inputs β^* to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. By t_s-validity with termination of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, every honest party obtains the same output B^* from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. So all honest parties eventually receive $n - t_s > t_s$ valid decryption shares for each ciphertext in each pre-block of B^*, and they all output the same block.

The case where the network is asynchronous and at most t_a parties are corrupted is similar. In each iteration, each honest party multicasts a set of transactions and so every honest party eventually receives input from at least $n - t_a \geq n - t_s$ distinct parties and can form an $(n - t_s)$-quality pre-block. This means that every honest party eventually runs $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using an $(n - t_s)$-quality pre-block as input. By t_a-security of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, all honest parties eventually receive the same output B^* from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. So all honest parties will eventually receive $n - t_a > t_s$ valid decryption shares for each ciphertext in each pre-block of B^*, and they all output the same block. □

In what follows, we let $\mathsf{Blocks}[k]$ denote the block output by honest parties in iteration k. We now turn our attention to liveness. We begin by proving a bound on the number of honest parties who contribute transactions to a block. Formally, we say that an honest party P_i *contributes transactions* to a block $B := \mathsf{ConstructBlock}(B^*)$ if there is a pre-block $\beta \in B^*$ such that $B[i] \neq \perp$. Using this lower bound, we show that any transaction that is at the front of most honest parties' buffers will eventually be output with overwhelming probability. Liveness follows by arguing that any transaction that is in the buffer of all honest parties will eventually move to the front of most honest parties' buffers.

Lemma 9. *Fix t_a, t_s with $t_a \leq t_s$ and $t_a + 2 \cdot t_s < n$, and assume at most t_a parties are corrupted and the network is asynchronous, or at most t_s parties are corrupted and the network is synchronous. Then in an execution of $\Pi_{\mathsf{ABC}}^{t_a,t_s}$, for any block B output by an honest party, at least $n - (t_s + t_a)$ honest parties contributed transactions to B.*

Proof. First, consider the case where at most t_a parties are corrupted and the network is asynchronous. As shown in the proof of Theorem 3, every honest party executes $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using an $(n - t_s)$-quality pre-block as input. Thus, the input of every honest party to $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ contains at least $n - (t_s + t_a)$ ciphertexts created by honest parties. By t_a-set quality of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, the output of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ contains some honest party's input and the lemma follows.

Next, consider the case where at most t_s parties are corrupted and the network is synchronous. As shown in the proof of Theorem 3, every honest party executes $\Pi_{\mathsf{ACS}}^{t_a,t_s}$ using the same $(n-t_s)$-quality pre-block β as input. By t_s-validity with termination of $\Pi_{\mathsf{ACS}}^{t_a,t_s}$, all honest parties output $B^* = \{\beta\}$ from $\Pi_{\mathsf{ACS}}^{t_a,t_s}$. Because β is $(n-t_s)$-quality, it contains at least $(n-2t_s)$ honest parties' ciphertexts; the lemma follows. □

Lemma 10. *Assume the conditions of Lemma 9. Consider an iteration k and a transaction* tx *such that, at the beginning of iteration k, all but at most t_s honest parties have* tx *among the first L transactions in their buffers. Then for any $r > 0$,* tx *is in* $\mathsf{Blocks}[k : k+r]$ *except with probability at most* $(1-1/n)^{r+1}$.

Proof. By Lemma 9, at least $n-(t_s+t_a)$ honest parties contribute transactions to $\mathsf{Blocks}[k]$. So even if t_s parties are corrupted, at least one of the $n-2t_s$ honest parties who have tx among the first L transactions in their buffers contributes transactions to $\mathsf{Blocks}[k]$. That party fails to include tx in the set V of transactions it chooses with probability $\binom{L-1}{L/n}/\binom{L}{L/n} = 1-\frac{1}{n}$, and so tx is in $\mathsf{Blocks}[k]$ except with probability at most $1-\frac{1}{n}$. (Note that this does not take into account the fact that the adversary may be able to choose which honest parties contribute transactions to B. However, because the parties encrypt their transactions, the adversary's choice has no effect on the calculation.) If tx does not appear in $\mathsf{Blocks}[k]$, then we can repeat the argument in all successive iterations $k+1, \ldots, k+r$ until it does. □

Theorem 4 (Liveness). *Fix $t_a \leq t_s$ with $t_a + 2 \cdot t_s < n$. Then $\Pi_{\mathsf{ABC}}^{t_a,t_s}$ is t_a-live in an asynchronous network, and t_s-live in a synchronous network.*

Proof. Suppose all honest parties have received a transaction tx. If, at any point afterward, tx is not in some honest party's buffer then tx must have already been included in a block output by that party (and that block will eventually be output by all honest parties). If all honest parties have tx in their buffers, then they each have a finite number of transactions ahead of tx. By completeness, all honest parties eventually output a block in each iteration. Additionally, by Lemma 9, at least $n-(t_s+t_a)$ honest parties' inputs are incorporated into each block, and so in each iteration all but at most t_s honest parties each remove at least L/n transactions from their buffers. It follows that eventually all but at most t_s honest parties will have tx among the first L transactions in their buffers. Once that occurs, Lemma 10 implies that tx is included in the next κ blocks except with probability negligible in κ. □

The above shows that a transaction received by all honest parties is eventually output. This is the standard notion of liveness in asynchronous networks. When working in a synchronous model, on the other hand, it is common to analyze liveness in more concrete terms. We provide such an analysis in the full version of the paper.

5.4 Efficiency and Choice of Parameters

The communication cost per iteration is dominated by the cost of the ACS and BLA subprotocols. Both ACS and BLA are run on pre-blocks, which have size $L \cdot |\mathsf{tx}| + O(n \cdot \kappa)$. Thus, each execution of BLA incurs cost $O(n^3\kappa^2 + n^2 L|\mathsf{tx}|\kappa)$, and an execution of ACS incurs cost $O(n^4\kappa + n^3 L|\mathsf{tx}|)$. The overall communication per block is therefore $O(n^4\kappa + n^3\kappa^2 + n^3 L|\mathsf{tx}| + n^2 L|\mathsf{tx}|\kappa)$.

At the beginning of every iteration, each honest party uniformly selects L/n transactions from among the first L transactions in its buffer. The following lemma shows that the expected number of *distinct* transactions they collectively choose is $O(L)$:

Lemma 11. *Assume the conditions of Lemma 9. In any iteration of* $\Pi_{\mathsf{ABC}}^{t_a,t_s}$*, the expected number of distinct transactions contributed by honest parties to the block* B *output by the honest parties in that iteration is at least* $L/4$*.*

Proof. The expectation is minimized when all honest parties have the same L transactions as the first L transactions in their buffers, so we assume this to be the case. As in Lemma 10, for some particular such transaction tx, the probability that some particular honest party fails to include tx in the set V of transactions it chooses is $1 - \frac{1}{n}$. Since, by Lemma 9, at least $n - (t_s + t_a) > n/3$ honest parties contribute transactions to B, the probability that none of those parties choose tx is at most $\left(1 - \frac{1}{n}\right)^{n/3} \leq e^{-1/3} < 3/4$, and so tx is chosen by at least one of those parties with probability at least $1/4$. (Once again, we do not take into account the fact that the adversary may be able to choose which honest parties contribute transactions because honest parties encrypt the transactions they choose.) The lemma follows by linearity of expectation. □

Because each block contains $O(L)$ transactions, the communication cost per transaction is $O((n^4\kappa + n^3\kappa^2)/L + n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$. So for $L = \Theta(n\kappa)$, the amortized communication cost per transaction is $O(n^3|\mathsf{tx}| + n^2|\mathsf{tx}|\kappa)$.

We remark that although each block contains at least $L/4$ distinct transactions in expectation, it is possible that some of those transactions are not new, i.e., they may have already been included in a previous block. This is possible because honest parties may sample their input in some iteration before having finished outputting blocks in all previous iterations. Thus, the actual communication cost per transaction may be higher than what we computed above. In general, the amount of overlap between blocks will depend on the spacing parameter λ as well as the actual network conditions and the parties' local clocks. If λ is too small, some space in each block may be wasted on redundant transactions; however, setting λ to be too large could introduce unnecessary delays in a synchronous network. Understanding how different choices of λ affect our protocol's performance in various network conditions is an interesting challenge for future work.

5.5 Optimality of Our Thresholds

We show that our protocol achieves the optimal tradeoff between the security thresholds. This result does not follow immediately from the impossibility result of Blum et al. [5] for network-agnostic Byzantine agreement because reductions from BA to atomic broadcast do not trivially translate to the network-agnostic setting; however, the main ideas of their proof readily extend to the case of atomic broadcast.

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

Proof. Assume $t_a + 2t_s = n$ and fix an ABC protocol Π. Partition the n parties into sets S_0, S_1, S_a where $|S_0| = |S_1| = t_s$ and $|S_a| = t_a$. Consider the following experiment:

- Choose uniform $m_0, m_1 \leftarrow \{0,1\}^\kappa$. At global time 0, parties in S_0 begin running Π holding only m_0 in their buffer, and parties in S_1 begin running Π holding only m_1 in their buffer.
- All communication between parties in S_0 and parties in S_1 is blocked. 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^b begin running Π (at global time 0) with their buffers containing only m_b, and communicate only with each other and parties in S_b.

Compare this experiment to a hypothetical execution E_{sync} of Π in a synchronous network, in which parties in S_1 are corrupted and simply abort, and the remaining parties are honest and initially hold only (uniformly chosen) m_0 in their buffer. The views of parties $S_0 \cup S_a^0$ in the experiment are distributed identically to the views of the honest parties in E_{sync}. Thus, t_s-liveness of Π implies that in the experiment, all parties in S_0 include m_0 in some block. Moreover, since parties in S_0 never receive information about m_1, they include m_1 in any block with negligible probability. By a symmetric argument, in the experiment, all parties in S_1 include m_1 in some block, and include m_0 in any block with negligible probability.

Now, consider a hypothetical execution E_{async} of Π, this time in an asynchronous network. In this execution, parties in S_0 and S_1 are honest while parties in S_a are corrupted. The parties in S_0 and S_1 initially hold $m_0, m_1 \leftarrow \{0,1\}^\kappa$, respectively. The corrupted parties interact with parties in S_0 as if they are honest and have m_0 in their buffer, and interact with parties in S_1 as if they are honest and have m_1 in their buffer. Meanwhile, all communication between 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 t_a-consistency is violated with overwhelming probability. □

5.6 Adaptive Security

Our analysis of TARDIGRADE assumes a static adversary who must choose the set of corrupted parties prior to the start of the protocol. In fact, TARDIGRADE is not secure against an adaptive adversary, since an adaptive adversary can prevent Π_{BLA} from terminating within time $5\kappa\Delta$ by corrupting the parties who are chosen as leaders. It is possible to modify TARDIGRADE to achieve adaptive security by suitably modifying Π_{BLA} in a relatively standard way: rather than choosing a leader who acts as the only proposer, each party will act as the proposer for

one instance of the propose protocol, and a leader is then chosen retroactively after all instances terminate. Designing an adaptively secure network-agnostic atomic broadcast protocol with improved communication complexity is an interesting direction for future work. (Note that the committee-based approach in the following section is not adaptively secure.)

6 Improving Complexity using Committees

In this section, we describe an extension to TARDIGRADE that achieves lower amortized communication complexity in the presence of a static adversary. The improved protocol, UPGRADE, achieves expected communication complexity per transaction that is *linear* in n; specifically, it has expected per-transaction communication complexity $O(n\kappa|\mathsf{tx}| + \kappa^2|\mathsf{tx}|)$. This is made possible by delegating the most expensive steps of TARDIGRADE to a small committee.

To prove security for TARDIGRADE, we often used the fact that any sufficiently large subset of parties contained at least some minimum number of honest parties. We cannot assume this about the committees in UPGRADE, as the committee may be constant size, and in particular may be less than the number of corrupted parties. Instead, we prove that UPGRADE is secure in a setting with $O(\epsilon)$ fewer corrupted parties, where ϵ is a positive constant parameter of the protocol. More formally, fix t_s, t_a as before, and fix $\hat{t}_s$ such that $\hat{t}_s \leq (1 - 2\epsilon) \cdot t_s$ (for some $\epsilon > 0$); with probability $1 - e^{-O(\epsilon^2\kappa)}$, the improved ABC protocol is $\hat{t}_s$-secure in a synchronous network and t_a-secure in an asynchronous network. (Unless otherwise mentioned, all of the claims in this section hold with this probability.)

As in TARDIGRADE, we assume a trusted dealer who sets up threshold encryption and signature schemes. During the setup phase, the dealer also selects a *committee* $C \subset \{P_1, \ldots, P_n\}$ of size $O(\kappa)$ and provides each committee member $P_i \in C$ with a special credential π_i that proves P_i is on the committee. (For example, π_i might be a signature $\langle i \rangle_D$ on the index i that can be checked against the dealer's public key.) We also assume that there is a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$ known to all the parties.

6.1 Committee-Based Reliable Broadcast

We briefly describe a committee-based reliable broadcast protocol that will prove useful in our improved ACS construction. The basis for the committee-based protocol is a plain reliable broadcast protocol Bcast that is t_s-valid and t_a-consistent with communication complexity $O(n^2\,|v|)$ a hash of the sender's input. (An example construction can be found in the full version.) The sender sends their input v individually to each of the committee members. If the hash output by the reliable broadcast matches this value, the committee members propagate v to all parties.

The security analysis uses standard techniques for broadcast; for completeness, proofs can be found in the full version.

$\Pi^{t_a,t_s}_{\mathsf{BB}^+}(v)$

Throughout, let $t_\kappa := \lfloor \frac{(1-\epsilon)\cdot\kappa\cdot t_s}{n} \rfloor$.

- If $P_i = P^*$: send input v to each $P_j \in C$, and input $h = H(v)$ to $\Pi^{t_s}_{\mathsf{BB}}$.
- Run $\Pi^{t_s}_{\mathsf{BB}}$.
- If $P_i \in C$ and $\Pi^{t_s}_{\mathsf{BB}}$ has output h': upon receiving a message v' from P^* or (v', h', π_j) from some $P_j \in C$, if $H(v') = h'$, multicast (v', h', π_i).
- Upon receiving $(v'', H(v''), \pi_j)$ from at least $t_\kappa + 1$ distinct $P_j \in C$ (even if $\Pi^{t_s}_{\mathsf{BB}}$ has not yet output a value), output v'' and terminate.

Fig. 4. A reliable broadcast protocol for sender P^* and committee C, from the perspective of party P_i.

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{BB}^+}$. Running the inner broadcast on hashes of size $O(\kappa)$ has communication complexity $O(n^2\kappa)$, while sending the value, hash, and credential to all parties costs $O(n\kappa(|m|+\kappa))$. Thus, sending a message of size $|m|$ using the 'wrapped' reliable broadcast costs $O(n^2\kappa + n|m|\kappa + n\kappa^2)$, while sending it using the inner reliable broadcast alone costs $O(n^2|m|)$.

6.2 Committee-Based ACS

We can construct a committee-based ACS protocol (Figure 5) by making two minor changes to the basic ACS protocol introduced in Section 4. First, the inner (non-terminating) ACS protocol is modified to use the committee-based broadcast described in Section 6.1. Because broadcast is used opaquely by the inner ACS protocol, this change does not require any special modifications, and the claims previously proven about the inner ACS protocol still hold. Second, the termination wrapper is modified so that only the members of the committee send the output in its entirety. Upon outputting a set S from the inner (non-terminating) ACS subprotocol, each committee member P_i multicasts S and $\langle \mathsf{commit}, H(S) \rangle_i$, along with the credential they received from the dealer. The other parties will echo signature shares and hashes, but not the set S itself.

The proof that $\Pi^{t_a,t_s}_{\mathsf{ACS}^+}(v)$ is t_a-secure and has $\hat{t}_s$-validity with termination is very similar to the security proof for the basic ACS protocol, so we omit it.

Communication complexity of $\Pi^{t_a,t_s}_{\mathsf{ACS}^+}$. As before, let $|m|$ represent the size of parties' inputs. When instantiated using the committee-based broadcast protocol from Section 6.1, the communication complexity of the inner ACS protocol is $O(n^3\kappa + n^2|m|\kappa + n^2\kappa^2)$. Moving on to the rest of the protocol, we see that the committee members multicast their output, a signature share, and the credential they received from the dealer. (Note that the signature share is for a hash of the output rather than the entire output.) Since the signature share, credential, and hash are each of size $O(\kappa)$, this step contributes $O(n \cdot \kappa(n \cdot |m| + \kappa)) = O(n^2\kappa \cdot |m| + n\kappa^2)$. Next, all parties multicast a combined signature of size $O(\kappa)$, for a total cost of $O(n^2\kappa)$. All together, the total cost of the improved ACS protocol is $O(n^3\kappa + n^2|m|\kappa + n^2\kappa^2)$.

$\Pi^{t_a,t_s}_{\mathsf{ACS}^+}(v_j)$

Throughout, let $t_\kappa := \lfloor \frac{(1-\epsilon)\cdot\kappa\cdot t_s}{n} \rfloor$.

- Input v_j to $\Pi^{t_a,t_s}_{\mathsf{ACS}}$.
- If $P_j \in C$: upon receiving output S_j from $\Pi^{t_a,t_s}_{\mathsf{ACS}}$, compute $h := H(S_j)$ and multicast $(S_j, \langle \mathsf{commit}, h\rangle_j, \pi_j)$.
- Upon receiving at least $t_\kappa + 1$ valid signature shares $\sigma_i = \langle \mathsf{commit}, h\rangle_i$ from distinct parties in C on the same value h, form a combined signature σ for h and multicast (σ, h).
- Upon receiving a valid combined signature σ for some h, multicast σ.
- Upon holding S, σ such that σ is a combined signature for h from parties in C and S is a set such that $H(S) = h$, output S and terminate.

Fig. 5. A terminating ACS protocol with predetermined committee C, shown from the perspective of party P_j with input v_j.

6.3 An ABC Protocol with Improved Communication Complexity

Here, we give an overview of UPGRADE. Because the high-level techniques are similar to TARDIGRADE, we will highlight the key differences between the two protocols and defer further details to the full version.

The first (and simplest) difference is that wherever TARDIGRADE would run an instance of the plain ACS protocol, UPGRADE runs the improved version described in Section 6.2. The second difference concerns how parties choose and share their inputs, and how those inputs are combined to form a final block. At the beginning of the protocol, when parties choose a set of transactions to input, they will now also choose a second, larger input set, which is encrypted and sent only to the committee members. The committee members form the large ciphertexts into a separate pre-block, which is used to construct the final block in case ACS outputs only one small pre-block is output. Sending a large pre-block all-to-all is costly, so the committee members also form a placeholder called a *block pointer*. A block pointer contains a hash of a large pre-block and a combined signature on that hash by members of the committee. In most steps, the block pointer can be sent in place of the large pre-block. Although forming and sharing the block pointer adds some extra communication, we are able to significantly increase the expected number of distinct transactions.

References

1. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Efficient synchronous Byzantine consensus (2017), available at `https://eprint.iacr.org/2017/307`
2. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync hotstuff: Simple and practical synchronous state machine replication. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 106–118. IEEE (2020)
3. Beerliová-Trubíniová, Z., Hirt, M., Nielsen, J.B.: On the theoretical gap between synchronous and asynchronous mpc protocols. In: Proceedings of the 29th ACM

SIGACT-SIGOPS symposium on Principles of distributed computing (PODC). pp. 211–218 (2010)
4. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: Proceedings of the 13th annual ACM symposium on Principles of distributed computing (PODC). pp. 183–192 (1994)
5. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: Theory of Cryptography Conference (TCC). pp. 131–150. Springer (2019)
6. Blum, E., Liu-Zhang, C.D., Loss, J.: Always have a backup plan: fully secure synchronous mpc with asynchronous fallback. In: Annual International Cryptology Conference (CRYPTO). pp. 707–731. Springer (2020)
7. Bracha, G.: An asynchronous [(n-1)/3]-resilient consensus protocol. In: Proceedings of the third annual ACM symposium on Principles of distributed computing (PODC). pp. 154–162 (1984)
8. Castro, M., Liskov, B.: Practical Byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. pp. 173–186. OSDI '99, USENIX Association (1999)
9. Correia, M., Neves, N., Veríssimo, P.: From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. The Computer Journal **49**(1), 82–96 (2006)
10. Cristian, F., Aghili, H., Strong, R., Dolev, D.: Atomic broadcast: From simple message diffusion to Byzantine agreement. Information and Computation **118**(1), 158–179 (1995). https://doi.org/https://doi.org/10.1006/inco.1995.1060, https://www.sciencedirect.com/science/article/pii/S0890540185710607
11. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: International workshop on public key cryptography (PKC). pp. 160–179. Springer (2009)
12. Duan, S., Reiter, M.K., Zhang, H.: Beat: Asynchronous bft made practical. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2028–2041 (2018)
13. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (Apr 1988). https://doi.org/10.1145/42282.42283, https://doi.org/10.1145/42282.42283
14. Fitzi, M., Nielsen, J.B.: On the number of synchronous rounds sufficient for authenticated Byzantine agreement. In: Keidar, I. (ed.) Distributed Computing. pp. 449–463. Springer Berlin Heidelberg (2009)
15. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques (Eurocrypt 2015). pp. 281–310. Springer (2015)
16. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 803–818 (2020)
17. Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. In: Annual International Cryptology Conference (CRYPTO). pp. 499–529. Springer (2019)
18. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.: Zyzzyva: Speculative Byzantine fault tolerance. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles. pp. 45–58. SOSP '07, ACM (2007). https://doi.org/10.1145/1294261.1294267, https://doi.org/10.1145/1294261.1294267

19. Kursawe, K.: Optimistic Byzantine agreement. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems. p. 262. SRDS '02, IEEE Computer Society (2002)
20. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Programming Language Systems **4**(3), 382–401 (1982)
21. Liu, S., Viotti, P., Cachin, C., Quema, V., Vukolic, M.: XFT: Practical fault tolerance beyond crashes. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 485–500. USENIX Association, Savannah, GA (Nov 2016), https://www.usenix.org/conference/osdi16/technical-sessions/presentation/liu
22. Liu-Zhang, C.D., Loss, J., Maurer, U., Moran, T., Tschudi, D.: Mpc with synchronous security and asynchronous responsiveness. In: International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt 2020). pp. 92–119. Springer (2020)
23. Loss, J., Moran, T.: Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235 (2018), https://eprint.iacr.org/2018/235
24. Malkhi, D., Nayak, K., Ren, L.: Flexible byzantine fault tolerance. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 1041–1053 (2019)
25. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 31–42 (2016)
26. Momose, A., Ren, L.: Multi-threshold Byzantine fault tolerance. In: 28th Conference on Computer and Communications Security (CCS) (2021), available at `https://eprint.iacr.org/2017/307`
27. Mostefaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous byzantine consensus with t¡ n/3 and o (n2) messages. In: Proceedings of the 2014 ACM symposium on Principles of distributed computing (PODC). pp. 2–9 (2014)
28. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt 2017). pp. 643–673. Springer (2017)
29. Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
30. Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt 2018). pp. 3–33. Springer (2018)
31. Patra, A., Ravi, D.: On the power of hybrid networks in multi-party computation. IEEE Transactions on Information Theory **64**(6), 4207–4227 (2018). https://doi.org/10.1109/TIT.2018.2827360
32. Pease, M., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)