On the Round Complexity of Asynchronous Crusader Agreement

Ittai Abraham ✉
Intel

Naama Ben-David ✉
The Technion

Gilad Stern ✉
The Hebrew University of Jerusalem

Sravya Yandamuri ✉
Duke University

Abstract

We present new lower and upper bounds on the number of communication rounds required for *asynchronous* Crusader Agreement (CA) and Binding Crusader Agreement (BCA), two primitives that are used for solving binary consensus. We show results for the information theoretic and authenticated settings. In doing so, we present a generic model for proving round complexity lower bounds in the asynchronous setting. In some settings, our attempts to prove lower bounds on round complexity fail. Instead, we show new, tight, rather surprising round complexity upper bounds for Byzantine fault tolerant BCA with and without a PKI setup.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases lower bounds, asynchronous protocols, round complexity

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

Agreement problems are at the core of many distributed systems, finding applications in replicated and reliable systems, transactional systems, cryptocurrencies, and more. It is therefore not surprising that they have gained a lot of attention in the research community, with tens of papers written about agreement problems each year. A key metric of the performance of many distributed tasks, agreement problems included, is their *round complexity*, or, intuitively, the number of sequential network round trips required to solve the task. In practice, round complexity often translates directly to latency, since communication over distributed networks is slow and forms a major bottleneck in many systems [2, 3, 10, 18, 20, 25, 26, 27, 28].

Arguably the most important and well-known agreement problem, called *consensus*, requires all non-faulty parties to unanimously agree on the same valid input value. Unfortunately, a seminal result of Fischer, Lynch and Paterson shows that no consensus algorithm can guarantee termination in an asynchronous failure-prone system [16]. Interestingly, however, weaker agreement problem variants *can* be solved in such systems, and can be sufficient for many applications.

In one such problem, known as Crusader Agreement, all parties receive an input, and non-faulty parties must output either one of the input values or a special value $\bot$. All non-faulty parties outputting a non-$\bot$ value must agree, and are only allowed to output $\bot$ if there were at least two unique input values among the non-faulty parties [11]. This weakening of consensus can be quite powerful; intuitively, if a non-$\bot$ decision represents an action, it ensures that no conflicting actions will be taken by non-faulty parties. Furthermore, CA and its variants have been used as subroutines to solve consensus in randomized protocols [1, 5, 6, 8, 24].

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our contributions

In this paper, we focus on the *Crusader Agreement (CA)* problem, and present an in-depth study of the achievable round-complexity of the problem and its variants. In particular, we consider classic CA, as well as two important variants: *Binding* Crusader Agreement (BCA) and *Graded* (Binding) Crusader Agreement (G(B)CA). In BCA, crusader agreement must be solved, but with the additional requirement that at the time at which the first non-faulty process decides its output, there exists a non-$\perp$ value v such that no non-faulty party can output a different non-$\perp$ value in any continuation of the execution. Intuitively, the adversary is *bound* to one non-$\perp$ output value and cannot adaptively affect outputs based on future knowledge. This property has recently been shown to be crucial for solving randomized consensus an an asynchronous setting [1]. In GBCA, in addition to binding, *confidence levels* or *grades* are introduced, so that parties outputting a non-$\perp$ value do so with a *grade 1* or *grade 2* label, with the guarantee that if any non-faulty party outputs v with grade 2, no non-faulty party outputs $\perp$. This variant of CA is also useful in solving randomized consensus [1]. For all of these problems, we present lower and upper bounds on their round complexity in the asynchronous model, considering both crash and Byzantine failures. We consider networks with n parties and f faulty parties.

The lower bounds for crash-resilient protocols specifically deal with protocols in which the adversary can adaptively choose the inputs of some of the parties when it schedules their first actions. While this notion of adaptive inputs might seem unnatural, when using binding crusader agreement protocols to construct consensus protocols, it is advantageous to use protocols that are also secure when the adversary is able to choose inputs adaptively, both in terms of efficiency and simplicity. For further discussion on this topic, we refer the reader to Appendix D.

We first show that binding crusader agreement (BCA) requires 2 rounds if f parties can crash and $2f+1 \leq n \leq 3f$ in the adaptive input setting.

▶ **Theorem 1.** *It is impossible to solve crash fault tolerant BCA in* 1 *round when* $2f+1 \leq n \leq 3f$*, and the adversary can adaptively choose the inputs of the parties.*

We next turn to more complex lower bounds showing tasks where at least 3 rounds are required. First, we show that at least 3 rounds are required for crash-fault resilient graded binding crusader agreement (GBCA) if $2f+1 \leq n \leq 3f$ in the adaptive input setting.

▶ **Theorem 2.** *It is impossible to solve crash fault tolerant GBCA in* 2 *rounds when* $2f+1 \leq n \leq 3f$*, and the adversary can adaptively choose the inputs of the parties.*

Protocols solving crash-fault tolerant BCA in 2 rounds and crash-fault tolerant GBCA in 3 rounds have been constructed in [1], showing that these lower bounds are tight.

Next, we show that at least 3 rounds are required for solving Byzantine-fault tolerant crusader agreement (CA) if there is no PKI setup and $3f+1 \leq n \leq 4f$.

▶ **Theorem 3.** *It is impossible to solve Byzantine fault tolerant CA in* 2 *rounds when* $3f+1 \leq n \leq 4f$ *without PKI.*

We also show that this lower bound is tight in Theorem 15. Lastly, we show that the same bound holds for Byzantine-fault tolerant binding crusader agreement (BCA) if there is a PKI setup and $f \geq 2$, $3f+1 \leq n \leq 4f$.

▶ **Theorem 4.** *It is impossible to solve Byzantine fault tolerant BCA in 2 rounds with PKI when* $3f+1 \leq n \leq 4f$ *and* $f \geq 2$.

The lower bounds are first proven for one (or two) failures and then generalized to an arbitrary number of failures. Somewhat surprisingly, for our lower bounds that start with $f = 2$, the generalization to arbitrary $f > 2$ requires a non-trivial argument, requiring both a stronger lower bound for the $f = 2$ case and a more intricate method of generalization (see Appendix C).

Our Contributions: Upper Bounds

While thinking through the aforementioned lower bounds, some bounds seemed elusive and quite hard to achieve. This led us to the discovery of some surprising upper bounds. For example, the final lower bound described in the previous section looks suspiciously different from the other bounds: it only holds when $f \geq 2$. It turns out that the reason a more general lower bound couldn't be constructed is that there exists a protocol solving Byzantine-fault tolerant binding crusader agreement in 2 rounds if there is a PKI setup and $n = 4, f = 1$! Following this discovery, we constructed two more protocols that work for a small number of parties but don't seem to obviously generalize to any n and f. More precisely, we construct protocols solving Byzantine-fault tolerant binding crusader agreement in 3 rounds without a PKI setup for $n = 4, f = 1$ and for $n = 7, f = 2$. The resulting protocol is also a 3-round Byzantine-fault tolerant crusader agreement protocol for any n, providing a matching upper bound to one of above lower bounds.

A key insight to constructing these protocols is to design them to be as *patient* and *conservative* as possible. By *conservative*, we mean that parties output a non-$\perp$ value only if they have to. More concretely, they output the value v only if they see that their view could have been generated in a run in which all nonfaulty parties had the input v. In this case, parties must output v; otherwise, they may violate the validity of the protocol in some run. In all other cases, parties output $\perp$. By *patient*, we mean that parties wait and output a value only when they absolutely have to. More precisely, we aim to have parties output a value only when their view could have been generated in a run of the protocol in which they may not receive any more messages. Clearly, if they do not output a value at that point, there is a run in which they never output a value. This allows us to gather as much information as possible before parties output some value.

A somewhat surprising realization is that many protocols aren't as patient as they are allowed to be. For example, many protocols simply wait to hear $n - f$ messages in a given round before proceeding to the next. On the other hand, patient protocols could wait for even more information. For example, in the second round of the protocol, parties could wait to hear both round 1 and round 2 messages from the *same* $n - f$ parties, and for each others' reports to be consistent. From our upper bounds it seems like these conditions can be quite intricate and potentially very expensive to compute for large values of n. As such, we don't suggest these protocols as realistic upper bounds, but rather almost as an impossibility result, showing that a lower bound cannot be constructed for these cases. In further work, we hope to either show that these upper bounds are general, or that a lower bound can be constructed for some $f \geq 3$.

Related Work

It is well known that there are many impossibility results and lower bounds on distributed protocols [22]. Early results in the field show lower bounds on the round complexity in synchronous networks. For example, Fischer and Lynch show that $f + 1$ rounds are needed to reach Byzantine consensus in [15]. This lower bound was later generalized to authenticated protocols in [9] and [13]. In addition, similar lower bounds have been shown for synchronous crash-resilient consensus [4, 14]. Bounds are also known on early-stopping consensus, showing that at if the number of actually faulty parties is smaller than the corruption threshold, the number of needed rounds is at least 2 more than the number of corrupted parties [12].

On the other hand, fewer lower bounds are known on the round complexity of asynchronous protocols. The FLP result [16] shows that no deterministic consensus algorithm exists in an asynchronous system, even in the face of a single crash failure. More precisely, the proof shows that any consensus protocol in this setting has an infinite execution, essentially showing that the round complexity of such protocols is infinite. Similarly, the CAP theorem states that no distributed database can have consistency, availability and resilience to network partitioning [17, 23].

2 Model & Definitions

2.1 Model

Network

This work deals with a network of n parties connected via point-to-point communication channels. The network is asynchronous, meaning that there is no bound on message delay, but every message is eventually delivered in finite time. We assume that the point-to-point channels deliver messages in a FIFO order. The means that if a party sends a message m and then a message m' to the same party, the messages are delivered in that order. This can be enforced by simply adding a counter to each message, signifying when it was sent.

We model message delivery as being controlled by an adversary that can choose any delivery schedule as long as all messages are eventually delivered. We consider two types of faults in this work: crash and Byzantine faults. In networks with crash faults the adversary may cause up to f parties to crash, meaning that those parties do not take any further actions (including receiving or sending messages). On the other hand, in networks with Byzantine faults the adversary can control up to f parties and cause them to deviate arbitrarily from the protocol.

Finally, when we say that a network has a PKI setup, we mean that each party has a well-known public key and a private key that allow it to sign messages. Every party can use the public key to check that a message was indeed sent by a given party. In addition, parties can forward received messages with their signatures, proving that the message was indeed sent by the signing party.

Asynchronous Rounds

In the synchronous setting, rounds are very clearly defined using the bound Δ on message delivery. Defining the notion of round complexity for asynchronous protocols is less straightforward [7, 19, 21], and we follow [21]. We use the idea of "causal chains" in our definition of asynchronous round complexity. Intuitively, we can think of chains of messages, with each message being sent as a result of receiving previous messages. When a message is sent, it lengthens its chain by 1, and it is considered a round k message if its chain is of length k. When mapping this behaviour to synchronous systems, all of the messages that are sent without receiving any message will be sent in round 1. Round 2 messages will be sent after receiving round 1 messages, etc.

More precisely, if a message is sent in the beginning of the protocol without receiving any other message, we consider it to be a round 1 message. If a message is sent by a nonfaulty party as a result of receiving all messages in a set M, we consider it a round $k+1$ message, where k is the maximal round number for nonfaulty messages in M (or $k=0$ if there is no such message). We say that a party is in round k if it sent or received at least one round k message, and did not send or receive any higher-round message.

Using this notion of round complexity, we can define a k-round protocol:

▶ **Definition 5** (k-Round Protocol). *A protocol is a k-round protocol if all honest parties decide a value after at most k rounds.*

Note that it is possible that protocols never terminate or do not have a bound k on the number of rounds. If this happens, these protocols can be defined as having infinite round complexity, but we deal only with finite round complexity protocols in this work.

Adaptive Inputs

We say that an adversary can choose inputs adaptively if parties only have their inputs defined by the adversary at the moment they start participating in the protocol. When dealing with binding protocols, to be defined below, this means that the binding values can only depend on the state of the nonfaulty parties that started participating in the protocol at that time, and cannot depend on the inputs of parties that haven't started participating in the protocol.

2.2 Definitions

We start by defining the different tasks for which we have constructed lower and upper bounds. In this work we only consider protocols in which parties decide on values but continue sending messages even after their decision. This is a very common technique in the design of asynchronous protocols, allowing parties to help each other even after they have all the information needed to complete the protocols.

▶ **Definition 6** (Crusader Agreement (CA)). *In a Crusader Agreement protocol, each party has either 0 or 1 as an input, and parties decide either 0, 1 or $\bot$. A Crusader Agreement protocol has the following properties:*

- ***(Agreement)*** *If two nonfaulty parties decide values x and y, then either $x=y$ or one of the values is $\bot$.*
- ***(Validity)*** *If all nonfaulty parties have the same input, then this is the only possible decision for nonfaulty parties.*
- ***(Termination)*** *All nonfaulty parties eventually decide.*

To be able to implement CA with an optimal tolerance to crash faults, we must weaken its validity property to the following:

- **(Weak Validity)** If all parties have the same input v, then all nonfaulty parties decide v.

▶ **Definition 7** (Graded Crusader Agreement (GCA)). *In a Graded Crusader Agreement protocol, each party has either 0 or 1 as an input, and parties decide on pairs (v,g) such that $v\in\{0,1,\bot\}, g\in\{0,1,2\}$ and $v=\bot$ if and only if $g=0$. A Graded Crusader Agreement protocol has the following properties:*

- ***(Graded Agreement)*** *If two nonfaulty parties decide on the pairs $(v,g),(v',g')$, then $|g-g'|\le 1$ and if $v\ne v'$, either $v=\bot$ or $v'=\bot$.*
- ***(Validity)*** *If all nonfaulty parties have the same input v, then all nonfaulty parties decide $(v,2)$.*

(Termination) *All nonfaulty parties eventually decide.*

We define crash fault tolerant CA by weakening the validity property as with the non-graded version. We are also interested in the binding versions of both of these protocols. These protocols add an additional requirement that once the first nonfaulty party completes the protocol, the decision values are "bound". In a BCA protocol this means that even if the first party decides $\bot$, at that time we know which is the only possible non-$\bot$ decision value.

▶ **Definition 8** (Binding Crusader Agreement (BCA)). *A Binding Crusader Agreement protocol has all of the properties of a Crusader Agreement protocol as well as the following property:*

(Binding) *At the time at which the first nonfaulty party to decide decides on a value, there exists a value $b \in \{0, 1\}$ such that no nonfaulty party decides $1 - b$ in any extension of this execution.*

Note that the binding property is only interesting in the case that the nonfaulty party referred to in the definition decided $\bot$. Otherwise, it trivially follows from agreement. Like in the binding definition of crusader agreement, once the first nonfaulty party decides on a value in a graded binding crusader agreement protocol, there is only one non-$\bot$ value that can be output from the protocol (with some grade).

▶ **Definition 9** (Graded Binding Crusader Agreement (GBCA)). *A Graded Binding Crusader Agreement protocol has all of the properties of a Graded Crusader Agreement protocol as well as the following property:*

(Graded Binding) *At the time at which the first nonfaulty party to decide decides on a value, there exists a value $b \in \{0, 1\}$ such that no nonfaulty party decides either $(1 - b, 2)$ or $(1 - b, 1)$ in any extension of the protocol.*

We define crash fault tolerant BCA and GBCA by weakening the validity property as with the non-graded version.

3 Lower Bounds

General Proof Approach.

Each of the presented lower bounds is proven in two steps. We start by proving a lower bound for a small number of parties, setting f to be 1 or 2. We then generalize these proofs in Appendix C. We show that if a protocol exists for some larger values of n and f, then such a protocol exists for the n and f for which we proved the original lower bound with the same round complexity. This is done by assuming that more general protocols exist, and showing that parties can simulate these protocols in the original settings (with a smaller number of parties).

For the proof of each lower bound, we construct a series of worlds. The worlds are constructed strategically to show that a party must take a certain action because their view is indistinguishable from another world where taking a different action would violate some property. In particular, we show indistinguishability with worlds where (1) all (nonfaulty) parties start with the same value, so deciding a different value would result in a violation of validity, and (2) all nonfaulty parties have sent all possible messages, so waiting for additional messages before deciding would result in a violation of termination. We put the descriptor "nonfaulty" in parenthesis where relevant due to the difference in the validity condition for crash and Byzantine fault tolerant protocols. To give the reader a hint as to the purpose of each world in our proofs, we add certain labels to the worlds.

We now describe the labels. In an x**-validity** world, all (nonfaulty) parties have input value x. In a **false** x**-validity** world, the view of some (nonfaulty) party is the same as in an x-validity world, causing them to decide a non-$\perp$ value (and grade 2, where relevant) even though all (nonfaulty) parties did not have the same input values. In a **maximally patient** world, a party receives all the messages that will be sent to them by nonfaulty parties, and therefore must decide without waiting for additional messages that depend on the actions of faulty parties. For the maximally patient label, we also indicate the party that crashes, meaning another party cannot wait for messages that depend on this party before deciding without violating termination. In a **false maximally patient** world, a nonfaulty party's view is the same as in a maximally patient world, so they decide before receiving all of the messages sent by nonfaulty parties. As previously mentioned, our proofs generally proceed by constructing a chain of worlds, where there are "validity worlds" on opposite ends, and in the middle of the chain some property (binding or agreement) is violated. We indicate when a world is **symmetric** to another previously-described world on the opposite end of the chain. We use the labels **binding violation** and **agreement violation** to indicate worlds in which the properties of binding and agreement are violated, respectively.

In addition to using labels, we separate the description of each world into two bullets. The first bullet indicates the messages sent by the parties and any message delays or specific orderings where needed. The second bullet indicates the view of one or more nonfaulty parties and the actions they take accordingly.

3.1 Results

For our first result, we start with a simple 1 round lower bound for crash fault tolerant BCA with adaptive inputs.

▶ **Theorem 1.** *It is impossible to solve crash fault tolerant BCA in* 1 *round when* $2f+1 \leq n \leq 3f$*, and the adversary can adaptively choose the inputs of the parties.*

We show a proof for a network of three parties: p_1, p_2, and p_3. Our ultimate goal is to build up to **World 4**, in which binding is violated. In **World 4**, a party decides while p_3 lags behind; after this, the adversary adaptively chooses the input of p_3 and forces p_3 to decide 1 or 0 after a party has already decided. In order to show why p_3 decides 1 or 0 in those executions, we show indistinguishability from **World 1** or **World 2**, where all parties start with input 1 or 0, respectively. In those worlds, p_3 must decide 1 or 0 in order to not violate validity. To show why the first-deciding party decides in **World 4** without waiting for any messages from p_3, we show indistinguishability from **World 3**, in which p_3 crashes without sending any messages. In **World 3**, parties cannot wait for messages that are dependent on p_3 before deciding, as this would result in a violation of termination.

3 party proof. **World 1** ($1-$**validity, maximally patient for** p_2 **crash**):

- p_1 and p_3 are nonfaulty. p_2 crashes immediately. All parties have input 1.
- p_1 and p_3 must decide 1 after receiving each other's messages without waiting for any additional messages by validity and termination.

World 2 ($0-$**validity, maximally patient for** p_1 **crash**):

- p_2 and p_3 are nonfaulty. p_1 crashes immediately. All parties have input 0.
- p_2 and p_3 must decide 0 after recieving each other's messages without waiting for any additional messages by validity and termination.

World 3 (maximally patient for p_3 crash):

- p_1 and p_2 are nonfaulty. p_3 crashes immediately. p_1 and p_3 start with inputs 1 and 0 respectively.
- p_1 and p_2 must decide after receiving each other's messages without waiting for any additional messages by termination.

World 4 (false maximally patient, false validity, binding violation):

- p_1, p_2, and p_3 are nonfaulty. p_1 starts with input 1 and p_2 starts with input 0; p_3 lags behind, and its input will be adaptively chosen later. p_1 and p_2's messages are delivered to each other, so they decide due to indistinguishability from **World 3**. The adversary now chooses one of the following extensions:
  1. p_3 has input value 1. p_1's messages are delivered to p_3, and p_2's messages are only delivered after p_3 decides.
  2. p_3 has input value 0. p_2's messages are delivered to p_3, and p_1's messages are only delivered after p_3 decides.
- In extension 1, p_3 outputs 1 due to indistinguishability from **World 1**; or in extension 2, p_3 outputs 0 due to indistigiushability from **World 0**. This constitutes a binding violation, as we show that both 1 or 0 are possible values that p_3 decides after another party has already decided. Note that this does not imply a violation of agreement, as it is possible for the party (or parties) deciding before p_3 to decide $\perp$.

◀

We now present our second result in the crash case: a 2 round lower bound for GBCA.

▶ **Theorem 2.** *It is impossible to solve crash fault tolerant GBCA in 2 rounds when $2f+1 \leq n \leq 3f$, and the adversary can adaptively choose the inputs of the parties.*

We show a proof using a network of three parties: p_1, p_2, and p_3. Our approach is to build up to a world, **World 3**, in which there is a violation of binding. The strategy of the adversary to violate binding is as follows. First, p_1 is forced to output before p_3's input value is chosen. Then, the adversary chooses p_3's input and forces them to decide 1 or 0, thus breaking binding. To show how the adversary has p_3 decide 1 or 0 in **World 3**, we present 2 symmetric sets of 3 worlds. Each set consists of the following three types of worlds:

1. A validity world showing why a party must decide a non-$\perp$ value with grade 2
2. A world where one of the parties crashes
3. A world that is both indistinguishable from the first type of world for some party other than p_3 (meaning that it decides a non-$\perp$ value with grade 2) and indistinguishable from the second type of world for p_3, showing why p_3 decides the non-$\perp$ value that it does (so as not to violate graded agreement) in each extension of **World 3** without waiting for more messages (so as not to violate termination).

For ease of exposition, we include only the worlds described in point 3 above (**World 1** and **World 2**) in the main proof of this theorem. We separate the indistinguishability arguments and the corresponding worlds into two lemmas: Lemma 10 and 12. Apart from the 2 sets of 3 symmetric worlds described above, and **World 3** in which binding is broken, we construct an additional world to show why p_1 decides in **World 3** while p_3 lags behind. This world and the corresponding indistinguishability argument are proven separately in Lemma 13. We provide the proof of the first lemma after the proof of Theorem 2 and refer the reader to Appendix A for similar proofs of the next two lemmas.

3 party proof. In the description of the following worlds, we only describe the runs until a specific point, and have some arbitrary message scheduling following that.

World 1 (false 1-validity, false maximally patient):

- p_1, p_2, and p_3 are nonfaulty. p_1 and p_3 have input 1, while p_2 has input 0. Initially, p_1's round 1 messages are delivered to p_2 and p_3, and then p_3's round 1 messages are delivered to p_1 and p_2. Following that, any round 2 messages that p_1 sends are delivered to p_2, and any of p_3's round 2 messages are delivered to p_1 and p_2. From this point on, p_2 and p_3's messages are delivered to each other without delay.
- By Lemma 10, p_3 decides without waiting for additional messages, and its output is of the form $(1, g)$ such that $g \in \{1, 2\}$.

World 2 (false 0-validity, false maximally patient, symmetric to World 1):

- p_1, p_2 and p_3 are nonfaulty. p_1 has input 1, and p_2 and p_3 have input 0. Initially, p_2's round 1 messages are delivered to p_1 and p_3, and then p_3's round 1 messages are delivered to p_1 and p_2. Following that, any round 2 messages that p_2 sends are delivered to p_1, and any of p_3's round 2 are delivered to p_1 and p_2. From this point on, p_1 and p_3's messages are delivered to each other without delay.
- By Lemma 12, p_3 must decide $(0, g)$ for $g \in \{1, 2\}$.

.

World 3 (binding violation, false maximally patient):

- p_1, p_2 and p_3 are nonfaulty. p_1 has input 1, p_2 has input 0, and p_3's input will be adaptively chosen by the adversary based on the value it wants p_3 to output after the first party to output does so. At the start of the execution, p_1 and p_2's round 1 messages are delivered to each other, and then any resulting round 2 messages are delivered to each other. By Lemma 13, p_1 outputs without waiting for any messages that depend on p_3 at this time. We will now show two extensions of this run, one in which p_3 outputs $(1, g)$ for some $g \in \{1, 2\}$, and one in which it outputs $(0, g)$ for some $g \in \{1, 2\}$, showing that the protocol is not binding.
  1. The adversary adaptively chooses input 1 for p_3. Following that, p_3 receives p_1's round 1 messages, and then continues communicating freely with p_2 without any delays. At this point in time, p_3's view consists of round 1 messages from p_1 and p_2 and any round 2 messages from p_2 sent as a result as receiving p_1's round 1 messages and then p_3's round 1 messages. This view is identical to the one it has in **World 1**, so p_3 decides $(1, g)$ for some $g \in \{1, 2\}$.
  2. The adversary adaptively chooses input 0 for p_3. Following that, p_3 receives p_2's round 1 messages, and then continues communicating freely with p_1 without any delays. At this point in time, p_3's view consists of round 1 messages from p_1 and p_2 and any round 2 messages from p_1 sent as a result as receiving p_2's round 1 messages and then p_3's round 1 messages. This view is identical to the one it has in **World 2**, so p_3 decides $(0, g)$ for some $g \in \{1, 2\}$.

◀

▶ **Lemma 10.** *In **World 1** from the proof of Theorem 2, p_3 must decide (1, g) for $g \in \{1, 2\}$ without waiting for any round 2 messages from p_1.*

Proof. World 1.a) (1-validity, maximally patient for p_2 crash):

- p_1 and p_3 are nonfaulty. p_2 crashes without sending any initial messages. All three parties start with input 1. p_1 and p_3 communicate without delay.

- p_1 and p_3 must decide $(1, 2)$ without waiting for any messages from p_2 by validity and termination.

World 1.b) (maximally patient for p_1 crash):

- p_1 and p_3 have input 1, while p_2 has input 0. p_1 is faulty, sends round 1 messages, which are delivered to both p_2 and p_3, and then p_1 crashes. Following that, p_3's round 1 messages are delivered to p_2. Finally, p_2 and p_3's messages are delivered to each other without delay.
- Because p_1 crashed, p_2 and p_3 must decide without waiting for any round 2 messages sent by p_1, by termination.

We now argue why in **World 1** from the proof of Theorem 2, p_3 must decide (1, g) such that $g \in \{1, 2\}$ without waiting for any round 2 messages from p_1. First, we show that p_1 decides (1, 2), in **World 1**. Observe that p_1's view in **World 1** is indistinguishable from its view in **World 1.a** because p_1 and p_3 have input 1 and they start by exchanging both round 1 and round 2 messages. It follows that p_1 decides (1, 2), and thus when p_3 decides some value, it must decide $(1, g)$ such that $g \in \{1, 2\}$ by graded agreement. Next, we argue that p_3 must decide in **World 1** without waiting for any round 2 messages from p_1. Observe that in **World 1**, since p_1's messages (apart from any round 1 messages) are delayed for p_3, p_3's view is indistinguishable from its view in **World 1.b**. As a result, p_3 must not wait for any round 2 messages from p_1 before deciding so as not to violate termination. Note that p_2 cannot send any messages which rely on p_1's round 2 messages, because this is a 2-round protocol, so p_3's view is indeed indistinguishable in both worlds. ◀

For our third result, we show a lower bound for Byzantine fault tolerant CA without PKI. With a Byzantine adversary and no PKI, the faulty parties are able to simulate receiving certain messages from nonfaulty parties.

▶ **Theorem 3.** *It is impossible to solve Byzantine fault tolerant CA in 2 rounds when $3f + 1 \leq n \leq 4f$ without PKI.*

We present a proof for 4 parties: p_1, p_2, p_3 and p_4. In this proof, we build up to **World 5** in which agreement is violated because nonfaulty parties p_1 and p_4 decide 1 and 0, respectively. We start by showing two maximally patient worlds (**World 1** and **World 2**), where one party has omission failures and sends its input value message only to one other party. By termination, the nonfaulty parties must not wait to hear more messages before deciding. We then show two symmetric validity worlds (**World 3** and **World 4**) in which a Byzantine party simulates receiving a message from a non-faulty party that it didn't send. Due to indistinguishability from the maximally patient worlds, honest parties must decide without waiting for additional messages, but they must decide non-$\perp$ values by validity. Finally, in **World 5**, the adversary uses a Byzantine p_3 to have p_1 and p_4 decide different non-$\perp$ values using indistinguishability from the previously defined worlds.

4 party proof. In the following discussion, when we say that parties p_1, p_2 and p_3 have each other's messages delivered, we mean that the party receives its own messages first, and then p_1's messages are delivered first, then p_2's and then p_3's (similarly for p_2, p_3 and p_4).

World 1 (maximally patient for p_4 crash):

- All parties except p_4 are nonfaulty. p_4 crashes immediately without sending any messages. p_1 and p_2 have input 1; p_3 and p_4 have input 0. p_1, p_2 and p_3 have their round 1 messages delivered to each other, and then any round 2 messages that they send as a result are delivered to each other.

- All nonfaulty parties must decide without waiting for any messages dependent on p_4.

World 2 (maximally patient for p_1 omission, symmetric to World 1):

- All parties other than p_1 are nonfaulty; p_1 has omission failures. p_1 and p_2 have input 1, while p_3 and p_4 have input 0. p_1 sends round 1 messages as an honest party would with input 1 only to party p_2, and the messages are delivered first for p_2. Following that, p_2, p_3 and p_4 have their round 1 messages delivered to each other, and then any round 2 messages that they send as a result are delivered to each other.
- All nonfaulty parties must decide without waiting for any more messages from p_1 by termination.

World 3 (0-validity, false maximally patient, simulation):

- All parties except for p_2 are nonfaulty. p_2 is Byzantine. p_1, p_3 and p_4 start with 0. p_2 acts as if it started with input 1 and simulates p_1 starting with input 1. All messages from p_1 are delayed to p_3 and p_4, until they both decide. p_2 acts as if it is a nonfaulty party with input 1 such that the first message it received was a round 1 message from an honest p_1 with input 1. Following that, p_2, p_3 and p_4 have their round 1 messages delivered to each other, and then any round 2 messages that they send as a result are delivered to each other.
- Due to indistinguishability from **World 2**, p_4 decides without waiting for any additional messages. By validity, p_4 decides 0.

World 4 (1-validity, false maximally patient, simulation, symmetric to World 3):

- p_3 is Byzantine, and the remaining parties are nonfaulty. p_1, p_2, and p_4 start with input 1; p_3 acts as if it nonfaulty and has the input 0. All messages from p_4 are delayed to p_1 and p_2. p_1, p_2 and p_3 have their round 1 messages delivered to each other, and then their round 2 messages delivered to each other.
- Due to indistinguishability from **World 1**, p_1 decides before receiving any messages from p_4. By validity, p_1 decides 1.

World 5 (agreement violation, false maximally patient, false validity):

- p_3 is Byzantine, and the remaining parties are nonfaulty. p_1 and p_2 have input 1, while p_3 and p_4 have input 0. p_3 starts by acting as a nonfaulty party would with input 0. Parties p_1, p_2 and p_3's round 1 messages are delivered to each other, and then any round 2 message that they sent as a result of receiving the round 1 messages. Following that, p_3 acts as if it did not receive any round 1 messages from p_1. Now, p_4's round 1 messages are delivered to p_2 and p_3, and their round 1 messages are delivered to p_4. Finally, all round 2 messages sent by p_2 and p_3 are delivered to p_4.
- This world is indistinguishable from **World 4** for p_1 since it exchanged round 1 and round 2 messages with parties p_2 and p_3 with the same inputs without hearing from p_4. In addition, this world is indistinguishable from **World 3** for p_4 because p_1 acts as if it first received round 1 messages from p_1 with input 1, and then p_2, p_3 and p_4 exchange round 1 and round 2 messages without receiving any further messages from p_1. Therefore, p_1 and p_4 decide 1 and 0 respectively, violating the agreement property.

◀

For our second lower bound in the Byzantine case, we prove the impossibility of Byzantine fault tolerant BCA with PKI in 2 rounds when $f \geq 2$. Since there is PKI, the faulty parties can no longer simulate receiving messages from nonfaulty parties. This necessitates a slightly more complex approach than that required for the previous lower bound.

▶ **Theorem 4.** *It is impossible to solve Byzantine fault tolerant BCA in 2 rounds with PKI when $3f + 1 \leq n \leq 4f$ and $f \geq 2$.*

In this proof, we build up to a **World 6** where we show a binding violation by having an extension where a nonfaulty p_1 decides 1 and an extension where a nonfaulty p_7 decides 0 after another nonfaulty party p_5 decides. Unlike in the proof of the previous lower bound, we can no longer rely on simulation due to the presence of PKI. If we want a nonfaulty party to decide a non-$\perp$ value $v \in \{0, 1\}$, it can hear that at most $f = 2$ parties started with $1 - v$. This is because, in order to argue that a party must decide a non-$\perp$ value in a given world, we show that this party's view is indistinguishable from its view in another world in which all nonfaulty parties started with that value, enabling us to invoke validity. With PKI, if a party hears that more than f parties started with the value opposite its input value, then it knows that it is not in a validity world. As such, when attempting to understand this proof it is helpful to work backwards, starting from **World 6** to see the views of p_1 and p_7 when they decide 1 and 0, respectively. The maximally patient worlds **World 1**, **World 2**, and **World 5** show why p_1, p_5, and p_7 decide without waiting for additional messages in **World 6**. To show why the views of p_1 and p_7 are indistinguishable from validity worlds, forcing them to decide 1 and 0 respectively, we show **World 3** and **World 4** in which the honest parties all start with the same value.

Proof. As in previous proofs, when we say a party receives messages from a list of parties, they receive the messages in the listed order. For example, if a party receives messages from $p_1, \ldots, p_4$, it receives the messages from p_1 first, then p_2, and so on.

World 1 (maximally patient for p_2 and p_1 crash):
- All parties except p_1 and p_2 are nonfaulty. p_1 and p_2 crash immediately without sending any messages. p_3 and p_4 start with input 1, while p_5, p_6 and p_7 start with input 0.
- All nonfaulty parties must decide without waiting for any messages dependent on p_1 or p_2; otherwise, termination is violated.

World 2 (maximally patient for p_5 crash and p_6 omission):
- All parties except p_5 and p_6 are nonfaulty. p_1, p_2, p_3, and p_4 start with input 1. p_6 and p_7 start with input 0. p_5 crashes immediately without sending any messages. p_6 is omission failure; all messages except for any round 1 messages it sends to p_2 are omitted, and these messages are delivered for p_2 before any messages from any other parties.
- Nonfaulty parties must decide without waiting for any messages dependent on p_5 or any messages dependent on p_6 (other than any round 1 messages it sends to p_2); otherwise, termination is violated.

World 3 (0-validity, false maximally patient):
- p_3 and p_4 are Byzantine and have input 1. The rest of the parties are honest and start with input 0. All messages from p_1 and p_2 are delayed for the other parties. p_3, p_4, p_5, p_6 and p_7 exchange the same messages as in **World 1** and in the same order.
- This world is indistinguishable from **World 1** for p_7. Therefore, it decides without waiting for any additional messages. By validity, p_7 decides 0.

World 4 (1-validity, false maximally patient):
- p_6 and p_7 are Byzantine and start with input 0; the rest of the parties are honest and start with input 1. All messages from p_5 are delayed for the other parties. p_6 doesn't send any messages except for any round 1 messages that it would have sent to p_2 if it was honest, and as in **World 2**, this message is delivered for p_2 before any messages from

any other parties. p_1, p_2, p_3, p_4 and p_7 send the same messages in the same order as in **World 2**.

- The world is indistinguishable from **World 2** for p_1, so it decides without waiting for any additional messages. By validity, p_1 decides 1.

World 5 (maximally patient for p_7 and p_1 omission):

- All parties except for p_1 and p_7 are nonfaulty. $p_1, \ldots, p_4$ start with input 1 and $p_5, \ldots, p_7$ start with input 0. All honest parties start by sending their round 1 messages. p_7 crashes immediately after sending its round 1 messages to all of the other parties. p_1 is omission failure, and the only message it sends is its round 1 message to p_2. p_2 receives round 1 messages from p_6 first, then from $p_1, \ldots, p_4$ and p_7, and finally from p_5. p_2 sends round 2 messages as a result of receiving the aforementioned round 1 messages. Parties $p_3, \ldots, p_6$ receive round 1 messages from $p_3, \ldots, p_7$ and send any resulting round 2 messages. They receive any round 1 messages from p_2 following that, and possibly send additional round 2 messages. Finally, p_5 receives all round 2 messages from parties $p_2, \ldots, p_6$.
- Note that parties $p_2, \ldots, p_6$ received all round 1 messages sent by each other, and p_5 received any round 2 message sent as a result from these parties as well. This means that p_5 receives all messages from nonfaulty parties in this world, and thus by termination, p_5 decides without waiting for any additional messages.

World 6 (binding violation, false maximally patient):

- p_3 and p_4 are Byzantine, and the remaining parties are nonfaulty. $p_1, \ldots, p_4$ have the input 1 and $p_5, \ldots, p_7$ have the input 1, like **World 5**. Initially, all messages from other parties are delayed for p_7 and p_1. In addition, messages from p_1 are delayed for $p_3, \ldots, p_6$. The beginning of the run is exactly the same the run in **World 5** for $p_2, \ldots, p_6$, with p_3, p_4 sending the required messages only to parties $p_2, \ldots, p_6$ and not to p_1, p_7. Since p_5's view is identical to one which causes it to decide, it decides some value in this world as well. Next, we show the two executions in which the adversary can get p_1 to decide 1 or p_7 to decide 0, which would mean the protocol isn't binding.
  - **(Extension where p_1 decides 1)** p_1 and p_7 start by receiving round 1 messages from $p_1, \ldots, p_4, p_7$. p_1 then receives any round 2 messages from $p_1, \ldots, p_4, p_7$ except for p_2 final round 2 message sent by p_2 as a result of receiving p_5's round 1 message (which it received last). In the above, p_3 and p_4 are Byzantine, and they only send p_1 the round 2 messages they would have as a result of receiving round 1 messages from $p_1, \ldots, p_4, p_7$. Note that p_1 receives round 1 messages from $p_1, \ldots, p_4, p_7$ and then round 2 messages from $p_1, \ldots, p_4, p_7$ corresponding to p_2 receiving p_6's round 1 messages first, and then all of the parties receiving round 1 messages from each other. p_1's view is identical to the view it would have in **World 4**, so it decides 1.
  - **(Extension where p_7 decides 0)** p_7 sees round 1 messages from $p_3, \ldots, p_6$, and then all round 2 messages that they sent as a result of receiving round 1 messages from $p_3, \ldots, p_7$. Note that they received round 1 message from p_1, p_2 only after receiving those messages. At this point, p_7's view is identical to its view in **World 3**, so it decides 0.

◀

▶ Remark 11. It is possible to define $S = \{p_2, p_3, p_5, p_7\}$ and $T = \{p_1, p_4, p_6\}$. For these sets, $S \cup T = \{p_1, \ldots, p_7\}$, $S \cap T = \emptyset$ and $|S| = 4$, $|T| = 3$. In the proof of Theorem 4, the adversary always corrupts at most one party in S and one party in T. From Theorem 20 we can conclude that no 2-round Byzantine fault tolerant protocol exists even for any $3f + 1 \leq n \leq 4f$ and $f \geq 2$.

4 Upper Bounds

Notation.

The notation for a message from a party p_i is i. The initial message from a party is a special case, as it also contains a subscript $v \in \{0, 1\}$ indicating the party's input value. The first message in a valid chain of messages is always an initial message of this form. Chains of messages are separated by the operator $\cdot$. As an example, $\langle i_1 \cdot j \rangle$ is a length two chain where p_j is forwarding the initial message of p_i, where p_i has input value 1. We define the notion of a prefix of a chain recursively. Message chain C' is a prefix of chain C if $C' = C$ or there exists a party p_j such that $\langle C' \cdot j \rangle {=} \langle C \rangle$. We say that a message chain C *depends on* party p_i if the first message in the chain is of the form i_x such that $x \in \{0, 1\}$ or there exists a prefix of chain C, P, such that $\langle P \cdot i \rangle$ is also a prefix of chain C.

4.1 Results

The following upper bounds are designed such that parties forward any message they receive each other and wait for as long as they can (or nearly as much as they can). By this we mean that parties only decide on values if the messages they received could have been all messages nonfaulty parties ever send throughout an execution of the protocol. The protocols are also conservative in the sense that parties default to outputting $\bot$ unless doing so might lead to a validity violation. A party is forced to output a value $x \neq \bot$ if its view could have been obtained in an execution in which all nonfaulty parties have the input x.

The protocol described in Algorithm 1 is designed to work as described above. Parties start by sending their signed input to all parties, and then forwarding that input to all parties. Whenever a party receives a signed input message it forwards that message to all parties. Every party p_i then waits until there are three parties (including itself) such that p_i received all of these parties' inputs, and the messages forwarding each other's inputs. Once that happens, p_i chooses whether to output the value x that it received as input, or the value $\bot$. If p_i saw that more than one party reported its input as $1 - x$ (either by receiving its input directly, or by receiving a forwarded input), p_i outputs $\bot$. Otherwise, p_i outputs x. We prove this protocol is a binding crusader agreement protocol in Theorem 17, provided in Appendix B.

Similarly to the previous protocol, in the protocol described in Algorithm 2, parties start by sending each other their inputs. They then forward any received input and any message forwarding an input, also indicating the messages' senders. Every party p_i then waits until there are three parties (including himself) that report consistent information about each other's messages. More specifically, they forward the same messages about each other as the messages the p_i received and forwarded. Then, p_i outputs its input x if it forwarded at most one input message with the value $1 - x$ and at most one of the three aforementioned parties forwarded more than one input message with the value $1 - x$. Otherwise, p_i outputs $\bot$.

In Appendix B, we show that the protocol is a CA protocol for any number of parties n such that $n \geq 3f + 1$ in Theorem 15. We then proceed to show that the protocol is also binding for $n = 4, f = 1$ and $n = 7, f \geq 2$ in Theorems 17 and 18 respectively, meaning that in these cases it is also a BCA protocol.

Algorithm 1 4-party authenticated Asynchronous BCA for Byzantine faults for party p_i

---

Input: x

$fwdVals_1 = fwdVals_2 = fwdVals_3 = fwdVals_4 = \{\}$, $initVals = \{\}$
send $\langle i_x \rangle$ and $\langle i_x \cdot i \rangle$ to all, $fwdVals_i = fwdVals_i \cup \{i_x\}$
upon receiving $\langle k_v \rangle$ from p_k and not having forwarded a message from p_k:
    send $\langle k_v \cdot i \rangle$ to all
    $fwdVals_i = fwdVals_i \cup \{k_v\}$
    $initVals = initVals \cup \{k_v\}$
upon receiving $\langle j_v \cdot k \rangle$ from p_k
    $initVals = initVals \cup \{j_v\}$
    if j_{1-v} hasn't been added to $fwdVals_k$: $fwdVals_k = fwdVals_k \cup \{j_v\}$
upon $\exists p_j, p_k \neq p_i$ s.t. i_x, k_v, and $j_{v'}$ are in $fwdVals_i \cap fwdVals_k \cap fwdVals_j$ s.t. $v, v' \in \{0, 1\}$:
    let S be the set $\{s | s_{1-x} \in initVals\}$
    if $|S| \leq 1$ **then** decide x
    else, decide $\perp$

Algorithm 2 7-party unauthenticated Asynchronous BCA for Byzantine faults for party p_i

---

Input: x

$coreSet_i = \{\}$
for $j \in 1 \ldots n$:
    $initVals_j = \{\}$
    for $k \in 1 \ldots n$:
        $fwdedMsgs_{j,k} = []$
send $\langle i_x \rangle$ to all
upon receiving $\langle j_v \rangle$ from p_j and $fwdedMsgs_{i,j} = []$:
    send $\langle j_v \cdot i \rangle$ to all
    $initVals_i = initVals_i \cup \{j_v\}$
    $fwdedMsgs_{i,j} = fwdedMsgs_{i,j}.append(j_v)$
upon receiving $\langle k_v \cdot j \rangle$ from p_j and $k_* \cdot j \notin fwdedMsgs_{i,j}$:
    send $\langle k_v \cdot j \cdot i \rangle$ to all
    $initVals_j = initVals_j \cup \{k_v\}$
    $fwdedMsgs_{i,j} = fwdedMsgs_{i,j}.append(k_v \cdot j)$
    $fwdedMsgs_{j,k} = fwdedMsgs_{j,k}.append(k_v)$
upon receiving $\langle k_v \cdot l \cdot j \rangle$ from p_j and having received $k_v \cdot l$ from p_l:
    $fwdedMsgs_{j,l} = fwdedMsgs_{j,l}.append(k_v \cdot l)$
upon $\exists$ a set of $n - f$ distinct parties $coreSet_i$ s.t. the following 3 conditions hold:
    1. $p_i \in coreSet_i$
    2. $\forall (j, k, l) \in coreSet_i$, $fwdedMsgs_{j,k} = fwdedMsgs_{l,k}$
    3. $\forall j \in coreSet_i$, $\exists v \in \{0, 1\}$ s.t. $fwdedMsgs_{i,j}[1] = v_j$ and $\forall k \in coreSet_i$, $v_j \in initVals_k$
    $\forall j \in \{1 \ldots n\}$ let $S_j = \{s | s_{1-x} \in initVals_j\}$
    if $|S_i| \leq f$ and $|\{j \in \{1 \ldots n\}$ s.t. $|S_j| > f\}| \leq f$:
        decide x
    else decide $\perp$

References

1. Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 381–391, 2022.
2. Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Virendra J Marathe, Athanasios Xygkis, and Igor Zablotchi. Microsecond consensus for microsecond applications. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, pages 599–616, 2020.
3. Marcos K Aguilera, Naama Ben-David, Rachid Guerraoui, Antoine Murat, Athanasios Xygkis, and Igor Zablotchi. Ubft: Microsecond-scale bft using disaggregated memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 862–877, 2023.
4. Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires $t + 1$ rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999.
5. Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.
6. Christian Cachin and Luca Zanolini. From symmetric to asymmetric asynchronous byzantine consensus. *arXiv preprint arXiv:2005.08795*, 2020.
7. Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 42–51. ACM, 1993.
8. Tyler Crain. Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with $t < n/3$ and $o(n^2)$ messages and $o(1)$ round expected termination. *arXiv preprint arXiv:2002.08765*, 2020.
9. Richard A. DeMillo, Nancy A. Lynch, and Michael Merritt. Cryptographic protocols. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 383–400. ACM, 1982.
10. Dan Dobre and Neeraj Suri. One-step consensus with zero-degradation. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 137–146. IEEE, 2006.
11. Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3(1):14–30, 1982.
12. Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.
13. Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.
14. Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: Crash failures. *Inf. Comput.*, 88(2):156–186, 1990.
15. Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.
16. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.
17. Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.
18. Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.
19. Leslie Lamport. Lower bounds on consensus. *Unpublished manuscript*, 2000.
20. Leslie Lamport. Fast paxos. *Distributed Computing*, 19:79–103, 2006.

21 Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Comput.*, 19(2):104–125, 2006.

22 Nancy Lynch. A hundred impossibility proofs for distributed computing. In *Proceedings of the eighth annual ACM Symposium on Principles of distributed computing*, pages 1–28, 1989.

23 Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

24 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 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*, pages 2–9, 2014.

25 Adriana Szekeres, Michael Whittaker, Jialin Li, Naveen Kr Sharma, Arvind Krishnamurthy, Dan RK Ports, and Irene Zhang. Meerkat: Multicore-scalable replicated transactions following the zero-coordination principle. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–14, 2020.

26 Cheng Wang, Jianyu Jiang, Xusheng Chen, Ning Yi, and Heming Cui. Apus: Fast and scalable paxos on rdma. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 94–107, 2017.

27 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

28 Irene Zhang, Naveen Kr Sharma, Adriana Szekeres, Arvind Krishnamurthy, and Dan RK Ports. Building consistent transactions with inconsistent replication. *ACM Transactions on Computer Systems (TOCS)*, 35(4):1–37, 2018.

A Proofs of Lower Bounds

▶ **Lemma 12.** *In **World 2** from the proof of Theorem 2, p_3 must decide (0, g) for $g \in \{1, 2\}$ without waiting for any round 2 messages from p_2.*

Proof. **World 2.a) (0-validity, maximally patient for p_1 crash, symmetric to World 1.a)**:

- All three parties have the input 0. p_2 and p_3 are nonfaulty, and p_1 crashes prior to sending any messages.
- p_2 and p_3 must decide (0, 2) without waiting for any messages dependent on p_1 by validity and termination.

World 2.b) (maximally patient for p_2 crash, symmetric to World 1.b):

- p_1 has input 1, while p_2 and p_3 start with input 0. p_2 sends round 1 messages, which are delivered to both p_1 and p_3, and then p_2 crashes. Following that, p_3's round 1 messages are delivered to p_1. Finally, p_1 and p_3's messages are delivered to each other without delay.
- Because p_2 crashed, p_1 and p_3 must decide without waiting for any additional messages from p_2, by termination.

We now argue why p_3 must decide $(0, g)$ for $g \in \{1, 2\}$ in **World 2** without waiting for any of p_2's round 2 messages. First, we show that p_2 decides (0, 2). Since p_1's messages are initially delayed, p_2 decides $(0, 2)$ due to indistinguishability from **World 2.a**, in which p_1 crashes. As a result, if p_3 decides, it must decide $(0, g)$ such that $g \in \{1, 2\}$ so as not to violate graded agreement. Next, we show why p_3 decides without waiting for any round 2 messages from p_2. This follows an indistinguishability argument with **World 2.b** for p_3, since any messages from p_2 apart from its round 1 messages are delayed for p_3 in **World 2**. ◀

▶ **Lemma 13.** *In **World 3** from the proof of Theorem 2, p_1 must output without waiting for any messages that depend on p_3.*

Proof. **World 3.a) (maximally patient for p_3 crash)**:

- p_1 and p_2 are nonfaulty, while p_3 crashes immediately before sending any messages. p_1 has input 1 and p_2 has input 0.
- p_1 and p_2 must decide without waiting for any messages dependent on p_3 by termination.

The lemma follows from a straightforward indistinguishability argument from **World 3.a)**, as any messages from p_3 and dependent on p_3 are delayed for p_1 in **World 3**. ◀

B Proofs of Upper Bounds

▶ **Theorem 14.** *Algorithm 1 solves Byzantine fault tolerant BCA in 2 rounds with a PKI setup when there are 4 parties, $n = 3f + 1$.*

Proof. **Termination.** Termination follows from the fact that there are at least 3 honest parties, and they all will eventually receive and forward each others' initial messages.

Validity. Assume all nonfaulty parties have the same input $x \in \{0, 1\}$. Parties only add values j_y to $initVals$ after receiving a message j_y, which contains j's signature on the value y. Nonfaulty parties only sign such messages with their input x, so nonfaulty parties can receive one signature on $1 - x$ by the single faulty party. Therefore, if some nonfaulty party decide on some value, it will see that $|S| \leq 1$ in line 12 and decide x.

Agreement. Assume by way of contradiction that two parties p_i and p_j output 1 and 0 respectively. Before deciding, each of those parties waited to hear at least 3 forwarded messages from at least 3 parties. Since there are 4 parties, and at most 1 Byzantine party, they have at least one such nonfaulty party in common. Let that party be p_k. At the time p_i terminated, it heard at most one forwarded input of 0, meaning that in p_k's first 3 forwarded messages, it sent at least two messages with the value 1. Therefore, before terminating p_j heard at least two forwarded 1 inputs, and thus it could not have output 0.

Binding. Assume without loss of generality that party p_1 is the first nonfaulty party to output some value. If it outputs a value $b \neq \perp$, then we can define b to be the binding value, and the binding property trivially holds because of the agreement property. Otherwise, let I be the indices of the parties that caused p_1 to terminate, and let G be the nonfaulty parties among them. Without loss of generality, assume that $I = \{1, 2, 3\}$ and that p_2 is nonfaulty (and possibly also p_3). For each $i \in G$, define $m_{1,i}, m_{2,i}, m_{3,i}$ to be the first three values echoed by p_i, and define m_i to be the most common value among $m_{1,i}, m_{2,i}, m_{3,i}$. Now, define b to be the most common value in the multiset $\{m_i | i \in G\}$, if such a value is uniquely defined. If there is no single most common value, define b to be p_4's input, which we will show is defined by this point in time.

If b is the most common value in the multiset $\{m_i | i \in G\}$, then at least two nonfaulty parties in G sent at least two echoes with the value b in their first three echoes. Any nonfaulty party that terminates must hear at least three echoes from at least one of those parties, and thus will not output $1 - b$. Otherwise, the most common value in $\{m_i | i \in G\}$ is not uniquely defined. This must mean p_3 is faulty and thus $G = \{p_1, p_2\}$. In addition, since the most common value is not defined, $m_1 \neq m_2$. Note that p_1 and p_2 agree on the value sent by p_3, so it cannot be the case that their first three echoed values are echoes of messages sent by the same three parties. In other words, at least one of them heard from p_4, and thus p_4's input is already defined to be some value x_4. We defined $b = x_4$ as above, and all that is left to show is that no party outputs $1 - b$. We already know that p_1 output $\perp$, and by construction, p_4 cannot output $1 - b = 1 - x_4$. Therefore, only p_2 might output $1 - b$ if that was its input. If p_2's first three echoes contain the value b twice, it would not output $1 - b$. Otherwise, $m_2 = 1 - b$ and thus $m_1 = b$. This means that if p_2 hears three echoes from p_1 before terminating, it will hear at least two echoes with the value $m_1 = b$ and will output $\perp$. Otherwise, before terminating it hears an input message from p_3 and p_4, as well as echoes of each others' values. In addition, it hears p_1's input before terminating, because p_1 is the first party to terminate and it heard p_2 echo its value at that time. In other words, p_2 hears all parties' input messages before terminating. As shown above, p_1 echoes three messages, with the input value m_1 appearing twice. Similarly, p_2 echoes three messages with the input value m_2 appearing twice. Finally, both forward the same message from p_3, and thus in total p_2 receives two messages with the value m_1 and two messages with the value m_2 before terminating. Since $m_1 \neq m_2$, in that case it outputs $\perp$. In other words, in all cases p_2 either outputs $\perp$ or b.

Round Complexity. We now prove that the protocol requires only 2 rounds. This follows from the fact that the only messages sent by honest parties are their initial messages with their input values (which don't depend on any other messages), and messages forwarding the initial messages of other parties. ◀

▶ **Theorem 15.** *Algorithm 2 solves Byzantine fault tolerant CA for $n \geq 3f + 1$ parties in 3 communication rounds without a PKI setup.*

Proof. **Validity.** Assume all nonfaulty parties have the input b, and that some nonfaulty party p_i outputs some value. At that time, it received the message $\langle j_{1-b}\rangle$ from at most f parties, and thus $|S_i| \le f$. In addition, every nonfaulty p_j only sends $\langle k_x \cdot j\rangle$ messages after receiving a $\langle k_x\rangle$ message from p_k. This means that each p_j sends at most f such messages with $x = 1 - b$, and thus for every nonfaulty p_j, $|S_j| \le f$. Therefore, both conditions of line 20 hold, and thus p_i outputs b, as required.

Agreement. Assume by way of contradiction that two nonfaulty parties p and q output 0 and 1 respectively. Define $coreSet_0$ and $coreSet_1$ to be the sets $coreSet$ they have at the time they output their respective values. Define $coreSet_{0,1} = coreSet_0 \cap coreSet_1$, and note that $|coreSet_{0,1}| \ge f + 1$ because $|coreSet_0| = |coreSet_1| = n - f$. There are at most f Byzantine parties, so let p_i be a nonfaulty party in $coreSet_{0,1}$. Both p and q completed the protocol with p_i in their respective core sets, so it saw that it forwarded the initial value messages sent by all parties in their respective cores. Assume without loss of generality that p_i sent messages of the form $\langle k_v \cdot j \cdot i\rangle$ for each pair of parties $p_j, p_k \in coreSet_0$ before it did so for all such pairs of parties in $coreSet_1$. From condition 2 of line 18, p received the messages $\langle k_v \cdot j\rangle$ from each such p_j as well as k_v from p_k. From the first condition of line 20, p_i saw that at most f of those k_v messages had $v = 1$, because otherwise $|S_i| > f$ would have been true, and p_i would have output $\perp$ instead. On the other hand, p_i forwarded the same messages in the same order to q. This means that for every $p_j \in coreSet_{0,1}$, p_i forwarded at least $f + 1$ messages of the form $\langle k_0 \cdot j \cdot i\rangle$ before forwarding the final message required for q to terminate. From the second condition of line 18, q waits to hear the messages $\langle k_0 \cdot j\rangle$ from the parties $p_j \in coreSet_{0,1}$, and thus when it terminates, it sees that at least $f + 1$ parties in $coreSet_1$ have forwarded at least $f + 1$ initial 0 values, causing it not to output 1 and reaching a contradiction.

Termination. All nonfaulty parties eventually send their input messages. After receiving those messages, every p_i sends a $\langle j_x \cdot i\rangle$ for every $\langle j_x\rangle$ message it received. Similarly, every p_i sends a $\langle k_x \cdot j \cdot i\rangle$ for every $\langle k_x \cdot j\rangle$ message it received. After receiving the all of these messages from each nonfaulty party, every nonfaulty party has the conditions of line 18 hold with respect to the $n - f$ nonfaulty parties, and thus every nonfaulty party decides some value if it hadn't done so previously.

Round Complexity. Parties send at most chains of length 3, and thus the protocol is a 3-round protocol. ◀

We now turn to show that the protocol is also binding in the case of $n = 4, f = 1$ and $n = 7, f \le 2$.

▶ **Lemma 16.** *Let p_i and p_j be two nonfaulty parties. If $p_k \in coreSet_i \cap coreSet_j$, then $\forall p_l \in coreSet_i$ and $\forall p_m \in coreSet_j$, either p_i's $fwdedMsgs_{l,k}$ is a prefix of p_j's $fwdedMsgs_{m,k}$ or p_j's $fwdedMsgs_{m,k}$ is a prefix of p_i's $fwdedMsgs_{l,k}$.*

Proof. By quorum intersection, $|coreSet_i \cap coreSet_j| \ge f + 1$ and at least one of the parties in the intersection must be nonfaulty. The lemma follows from condition 2 on line 18. ◀

▶ **Theorem 17.** *Algorithm 2 solves Byzantine fault tolerant BCA for $n = 4$ parties and $f = 1$ in 3 communication rounds without a PKI setup.*

Proof. As shown in Theorem 15, the protocol is a 3-round CA protocol for any $n \ge 3f + 1$, and thus it has the Validity, Agreement and Termination properties. All that is left to show is that the protocol is also binding.

Binding. Assume that the first nonfaulty party to output outputs $\perp$ (otherwise binding follows from agreement). W.l.o.g. assume p_2 is the first nonfaulty party to output, that it started with input value 1, and consider the set $coreSet_2$ at the time that p_2 decides. For binding not to hold, there must be an extension of this execution where some nonfaulty party decides 1 and one in which a nonfaulty party decides 0. W.l.o.g. assume that p_1 is the nonfaulty party who can decide 1 and p_4 is the nonfaulty party who can decide 0. Note that parties only decide a non-$\perp$ value if that was their input value. As such, p_1 must have started with 1 and p_4 must have started with 0. If p_1, p_2 and p_4 are in $coreSet_2$, then p_4 cannot later output 0 by the condition on line 20. Since $|coreSet_2| \geq 3$, there are two possible cases:

1. p_1, p_2 and p_3 are in $coreSet_2$. Then p_1 and p_2 both forward the messages $\langle 1_1 \cdot 3 \rangle$ and $\langle 2_1 \cdot 3 \rangle$. Since only one of p_1 and p_2 can be in $coreSet_4$ for p_4 to output 0, p_3 must be in $coreSet_4$. Assume first that p_2 is the other party in $coreSet_4$. For p_4 to be able to decide 0, it must not hear the messages $\langle 1_1 \cdot 3 \cdot 2 \rangle$ and $\langle 2_1 \cdot 3 \cdot 2 \rangle$ from p_2, or $coreSet_4$ will not satisfy the condition on line 20 for p_4 to output 0 (since p_4 will wait to hear that p_3 forwarded the initial values of p_1 and p_2 or it will hear messages from p_1). So p_2 must send all the messages necessary for p_4 to output 0 before it sends the messages $\langle 1_1 \cdot 3 \cdot 2 \rangle$ and $\langle 2_1 \cdot 3 \cdot 2 \rangle$. This necessarily includes the messages $\langle 4_0 \cdot 3 \cdot 2 \rangle$ and $\langle 3_0 \cdot 3 \cdot 2 \rangle$, as well as $\langle 4_0 \cdot 2 \cdot 2 \rangle$ and $\langle 3_0 \cdot 2 \cdot 2 \rangle$, and $\langle 4_0 \cdot 4 \cdot 2 \rangle$ and $\langle 3_0 \cdot 4 \cdot 2 \rangle$. If this happens, it cannot be the case that $coreSet_1$ satisfies the conditions for p_1 to output 1, and we have arrived at a contradiction. If p_1 is in P_4, a similar argument follows.
2. p_2, p_3, and p_4 are in $coreSet_2$. Assume that an extension in which p_4 later outputs 0 exists. Then it must be the case that p_3 starts with input value 0. Since p_1 can't hear from both p_3 and p_4 before later outputting 1, it must hear from p_2. If it hears from p_2 the messages of the form $\langle 4_0 \cdot 4 \cdot 2 \rangle$ and $\langle 3_0 \cdot 4 \cdot 2 \rangle$, and $\langle 4_0 \cdot 3 \cdot 2 \rangle$ and $\langle 3_0 \cdot 3 \cdot 2 \rangle$, it cannot later output 1 (since necessarily p_2 and another party in $coreSet_4$ must have forwarded more than one initial value message with value 0). So then p_2 must send to p_1 the messages necessary for p_1 to output 1 before it sends those messages. But if it does that, p_4 would hear all of those messages and eventually have more than one party in its set $coreSet_4$ that forward more than one initial value message containing 1, a contradiction.

◀

▶ **Theorem 18.** *Algorithm 2 solves Byzantine fault tolerant BCA for $n = 7$ parties and $f \leq 2$ in 3 communication rounds without a PKI setup.*

Proof. As in the previous theorem, all that is left to show is that the protocol is binding.

Binding. We use a proof by contradiction. Consider the first nonfaulty party to output, p_*. Once p_* outputs, there must be an extension in which a nonfaulty party p_1 outputs 1 and an extension in which a nonfaulty party p_0 outputs 0. We refer to the extensions as ext-1 and ext-0, respectively. Assume w.l.o.g. that a majority of the parties in $coreSet_*$ (≥ 3) sent input value messages containing 1. Then p_0 cannot be in $coreSet_*$. This follows from two points: the fact that p_* outputs before p_0 does and the condition on line 20 by which a party decides a non-$\perp$ value. Let $support_l$ for $l \in \{0, 1\}$ be a set of 3 distinct parties from $coreSet_l$ at the time at which p_l decides such that $\forall p_j \in coreSet_l$, $|S_j| \leq 2$, where $S_j = \{s | s_{1-l} \in initVals_j\}$ (note that $coreSet_l$ must contain at least 3 parties satisfying this condition for p_l to decide l). $coreSet_0 \cap coreSet_*$ contains at least 3 parties, at least one of which must be in $support_0$. We consider 3 possible cases:

1. There is a single party in $support_0 \cap coreset*$, and it is honest. Refer to this party as p_H. It must send all of its messages to p_0 that are necessary for p_0 to output 0 before it forwards the initial messages of all parties in $coreSet_1$ (otherwise it cannot be in $support_0$). Therefore, it must receive the messages where all parties in $coreSet_0$ forward the initial messages of all of the parties in $coreSet_0$. At least 3 parties in $coreSet_0$ must

also be in $coreSet_*$. Note that then p_1 cannot be in $coreSet_0$, and p_1 must be in $coreSet_*$. p_* expects all parties in $coreSet_*$ to forward all of the 2-chain messages sent by parties in $coreSet_*$, contradicting that p_1 could output 1 after p_* outputs by the condition on line 20.

2. There is a single party in $support_0 \cap coreset*$, and it is Byzantine. Refer to this party as p_B. In ext-0, p_0 must not hear all of the messages where p_B forwarded the input value of all parties in $coreSet_*$, since $f + 1 = 3$ of those parties have input value 1 and in that case, p_B can't be in $support_0$. In order not to contradict Lemma 16 in ext-0, p_B must forward all of the input value messages of parties in $coreSet_0$ prior to forwarding all of the input value messages of parties in $coreSet_*$. In addition, all honest parties in $coreSet_0 \cap coreset*$ (there must be at least 1) must forward all messages that they need to send p_0 in ext-0 prior to forwarding the messages where p_B forwards the input value messages of all parties in $coreSet_*$ (otherwise p_0 hears that p_B forwarded 3 initial value messages with value 1, and it waits to receive the corresponding 2-chain messages from p_B prior to outputting in ext-0, a contradiction). Refer to such an honest party in $coreSet_0 \cap coreset*$ as p_{h0*}. To forward all messages that they need to send p_0 in ext-0 prior to forwarding the messages where p_B forwards the input value messages of all parties in $coreSet_*$, p_{h0*} must receive messages from all parties in $coreSet_0$ forwarding the initial value message of all parties in $coreSet_0$. This implies that p_1 cannot be in $coreSet_0$, and by quorum intersection it must be in $coreSet_*$. The rest of the proof follows the same as that of case 1.

3. Both parties in $support_0 \setminus p_0$ are honest and in $coreSet_*$. Note that the parties must send all messages that they need to send to p_0 in ext-0 prior to forwarding the initial messages of all parties in $coreSet_*$ (as they cannot be in $support_0$ if p_0 hears them forward the initial value messages of three parties with value 1). To do so, they need to forward the initial messages of all parties in $coreSet_0$, as well as the messages in which every party in $coreSet_0$ forwards the initial message of every party in $coreSet_0$. This means that they must receive those 1-chain and 2-chain messages from each party in $coreSet_0$ (implying that p_1 cannot be in $coreSet_0$). By quorum intersection, it must be the case that there is at least one party in $support_1 \cap coreSet_0$. Let this party be p_{s1c0}. Note that p_1 cannot be in $coreSet_*$ since there are at least 3 parties in $coreSet_0 \cap coreSet_*$, $p*$ hears from the parties in $support_0 \setminus p_0$ that they forwarded all of the initial messages of parties in $coreSet_0$, and p_* expects all parties in $coreSet_*$ to forward all 1-chain and 2-chain messages sent by these parties. Since neither p_1 nor p_0 are in $coreSet_*$, it must be the case that $p_{s1c0} \in coreSet_*$. In order to output, p_*, by the conditions on line 18, requires all parties in $coreSet_*$ to also forward the messages where p_{s1c0} forwards the initial messages of all parties in $coreSet_0$, and it has to hear these corresponding 2-chain messages from p_{s1c0}. Unless all nonfaulty parties in $coreSet_* \cap coreSet_1$ send all of the messages they need to send to p_1 for it to output 1 before forwarding these messages, ext-1 cannot exist. There must be at least one honest party in $coreSet_* \cap coreSet_1$ and it must receive from all parties in $coreSet_1$ the forwarded initial messages of all parties in $coreSet_1$ to do so. Clearly then, p_0 can't be in $coreSet_1$, so one of the parties in $support_0 \setminus p_0$ must be in $coreSet_1$. This party cannot forward all of the input value messages of all parties in $coreSet_1$ prior to sending to p_0 all of the messages it needs to send for it to output 0 (as then p_0 would hear that a party in $support_0$ forwarded $> f$ initial value messages with value 1). Due to FIFO channels p_1 inevitably hears from this party that p_{s1c0} forwarded the initial messages of all parties in $coreSet_0$, a contradiction.

4. There is one honest party and one Byzantine party in $support_0 \setminus p_0$, and both of them are in $coreSet_*$. Refer to the Byzantine party in this set as p_B and the honest party in this set as p_H. Note that by quorum intersection, one of the parties in the set $\{p_0, p_H, p_B\}$ must be in $coreSet_1$. Using similar reasoning to that of case 3, we first show that it cannot be the case that p_0 or p_H is in $coreSet_1$. p_0 cannot hear the 2-chain messages in which p_H forwards the initial value messages of all parties in $coreSet_*$, but p_H must send these messages prior to p_* outputting. So p_H must send to p_0 all of the messages it needs to send to p_0 in ext-0 for p_0 to output 0 prior to forwarding the initial value messages of all parties in $coreSet_*$. For this to happen, p_H must receive from all parties in $coreSet_0$ the 2-chain messages in which they forward the input value messages of all parties in $coreSet_0$. By quorum intersection, there must be at least one party, p_{s1c0} in $support_1 \cap coreSet_0$. If p_H is in $coreSet_1$, it notifies p_1 that a party in $support_1$ forwarded 3 input value messages with value 0, a contradiction. By the fact that at least 3 parties in $coreSet_0$ must also be in $coreSet_*$, using similar logic to that used in case 3, p_1 can't be in $coreSet_*$ or $coreSet_0$, so p_{s1c0} must be in $coreSet_*$. Due to FIFO channels, p_* expects all parties in $coreSet_*$ to forward the 2-chain messages where p_{s1c0} forwards all of the input value messages of parties in $coreSet_0$. This implies that all honest parties in $coreSet_1 \cap coreSet_*$ must send to p_1 all of the messages they need to send to p_1 in ext-1 prior to forwarding all 2-chain messages of p_{s1c0}, and prior to p_* outputting. This implies that p_0 cannot be in $coreSet_1$.

   We now show that binding cannot be broken if p_B is in $coreSet_1$. As argued above, the honest party in $coreSet_* \cap coreSet_1$ requires all parties in $coreSet_1$ to send it the forwarded initial messages of all parties in $coreSet_1$ before it forwards all messages necessary for p_* to output. So it requires a message from p_B forwarding all initial messages of all parties in $coreSet_1$. Since p_B is in $support_0$, p_0 should not hear this message. Since we have shown that p_1 and p_0 cannot be in each others' $coreSet$ or in $coreSet_*$, there are at least 3 parties in $coreSet_0 \cap coreSet_* \cap coreSet_1$ and at least one of them must be honest. This honest party must send all the messages it needs to send to p_0 and p_1 prior to sending p_* all the messages it needs to send p_* to output (otherwise it will notify p_0 that p_B forwarded the initial messages of 3 parties with input 1 or it will inform p_1 that a party in $support_1$ forwarded the initial messages of 3 parties with input 0). If it sends all messages for ext-0 first, it will notify p_1 that a party in $support_1$ forwarded 3 initial value messages containing 0 due to FIFO channels. If it sends all of the messages for ext-1 first, it will notify p_0 that a party in $support_0$ forwarded 3 initial value messages containing 1. Either way, we have arrived at a contradiction.

5. There are two Byzantine parties (p_{B1} and p_{B2}) in $support_0 \setminus p_0$, and both of them are in $coreSet_*$. By quorum intersection, there must be some honest party, p_h in $coreSet_0 \cap coreSet_*$ that has to send everything for ext-0 to p_0 before it sends all of its messages for p_* to output, because otherwise it will reveal to p_0 that a party in $support_0$ forwarded 3 input value messages with value 1, a contradiction. Note that this implies that p_* hears that 3 parties in $coreSet_*$ forwarded 3 input value messages with value 0 prior to outputting, and it must hear all parties in $coreSet_*$ forward these messages. Hence, p_1 cannot be in $coreSet_*$. p_1 also can't be in $coreSet_0$ since p_h expects to hear from all parties in $coreSet_0$ that they forward the input value messages of all parties in $coreSet_0$. Since p_{B1} and p_{B2} are in $coreSet_*$, and p_0 forwards the initial value messages of all parties in $coreSet_0$ and cannot hear 3 input value messages with value 1 before p_* outputs for ext-0 to exist, Lemma 16 implies that $support_1 \cap support_0 = \emptyset$. Thus, the two parties in $support_1 \notin p_1$ must be honest and in $coreSet_*$. By quorum intersection,

some party in $support_1$ must be in $coreSet_0$, some party in $support_0$ must be in $coreSet_1$, and there must be an honest party $p_{h,int} \in coreSet_1 \cap coreSet_* \cap coreSet_0$. As already noted, all honest parties in $coreSet_0 \cap coreSet_*$, including $p_{h,int}$, must send all messages they need to send in ext-0 prior to sending all messages for p_* (and thus revealing that a party in $support_0$ forwarded 3 1s). To do so, $p_{h,int}$ will send messages that a party in $support_1 \cap coreSet_0$ (there must be at least 1) forwarded 3 0s; but this should not be revealed to p_1 before it outputs. This means that $p_{h,int}$ should send all messages for ext-1 before sending all messages for ext-0, but then it would reveal to p_0 that a party in $support_0$ forwarded 3 1s. Either way, ext-1 and ext-0 cannot both be possible when p_* outputs, and binding cannot be broken.

◀

C Generalizing the Lower Bounds

In this section, we generalize the lower bounds from lower bounds specifically for $n = 3, n = 4$ or $n = 7$ to lower bounds for $n \geq 3, n \geq 4$ or $n \geq 7$. The techniques for generalizing the lower bound in the case that $n \geq 3, n \geq 4$ are standard and provided for completeness. On the other hand, generalizing the lower bound for $n \geq 7$ is slightly more intricate. In the following we simply show how to generalize two of the lower bounds presented above, but generalizing the other ones (with different corruption models or numbers of rounds) is done in the same manner.

We start by showing how to generalize the lower bound for $n = 4$ and $f = 1$ to any n, f such that $4f \geq n \geq 3f + 1$. Identical arguments can be made to generalize the lower bounds for $n = 3$ and $f = 1$ to any n, f such that $3f \geq n \geq 2f + 1$.

▶ **Theorem 19.** *Assume that it is impossible to solve Byzantine fault tolerant crusader agreement in two rounds with $n = 4$ parties and $f = 1$ faults. Then it is impossible to construct such a protocol for any $f \in \mathbb{N}$ and $4f \geq n \geq 3f + 1$.*

Proof. Assume by way of contradiction, that for some f, n such that $4f \geq n > 3f$ there exists a Byzantine fault tolerant crusader agreement protocol for n parties resilient to f corruptions in which all parties decide on a value after at most two rounds without a PKI setup. We will use this protocol to construct a Byzantine fault tolerant crusader agreement protocol for 4 parties with 1 corruption that requires the same number of rounds, contradicting the theorem statement.

The protocol is designed for 4 parties $p'_1, \ldots, p'_4$ which simulate a full run of the n-party protocol running with parties $p_1, \ldots, p_4$. Start by partitioning the parties $p_1, \ldots, p_n$ into 4 roughly-equal groups: $P_1, \ldots, P_4$. Since n is not necessarily a multiple of 4, it is possible that some of the groups will contain one more party than the other groups. More precisely, set $\ell = (n \mod 4)$, and let $P_1, \ldots, P_\ell$ be of size $\lceil \frac{n}{4} \rceil$ and $P_{\ell+1}, \ldots P_4$ be of size $\lfloor \frac{n}{4} \rfloor$. In case that $\ell = 0$, this means that all set are exactly of size $\frac{n}{4}$. Note that in all other cases, this means that the sets do indeed contain a total of n parties, since their combined sizes are $\ell \cdot \lceil \frac{n}{4} \rceil + (4 - \ell) \lfloor \frac{n}{4} \rfloor = \ell \cdot (\lfloor \frac{n}{4} \rfloor + 1) + (4 - \ell) \lfloor \frac{n}{4} \rfloor = 4 \cdot \lfloor \frac{n}{4} \rfloor + (n \mod 4) = n$.

Now, in the 4-party protocol each party p'_i simulates the full n-party protocol for the parties in P_i. Every party p'_i receives an input x_i and simulates the actions of all parties in P_i after starting with the input x_i. This is done by running the code of each of those parties after receiving that input, and sending messages if required as described below. Whenever p'_i sees that party $p \in P_i$ sends a message m to some party $q \in P_j$ it does the following: if $j = i$, it simulates q receiving m by running the code that q would have run upon receiving the message from p. Otherwise, p'_i sends the message m to p'_j, along with the information that p sent the message to q. Similarly, when a party p'_j receives a message m from p'_i with the information that $p \in P_i$ sent that message to $q \in P_j$, p'_j simulates q receiving that message by running the code that q would have run upon receiving that message from p. Once p'_i sees that all of the simulated parties in P_i output values, it does the following: if at least one party in P_i output $\perp$, it outputs $\perp$. Otherwise, it outputs some non-$\perp$ value that a party in P_i output[1]. In this setting, the adversary can only corrupt a single party p'_i, which simulates the parties in P_i. The number of parties in P_i is at most $\lceil \frac{n}{4} \rceil$. By assumption, $n \leq 4f$, so $\lceil \frac{n}{4} \rceil \leq \lceil \frac{4f}{4} \rceil = f$. All other simulated parties act exactly the same as they would when receiving messages in the original protocol, since they are instructed to send and receive messages exactly as they would in the original protocol. In other words, the simulated run perfectly corresponds to a run in which the adversary corrupts at most f parties, in which messages between parties in the same set P_i are delivered immediately and the rest of the messages are delivered according to the scheduling dictated by the adversary. The protocol is secure under these conditions, and thus Validity, Agreement and Termination hold in the simulated run.

In order to complete the proof, all that is left to show is that the resulting 4-party protocol is a two-round Byzantine fault tolerant crusader agreement protocol with $n = 4$ and $f = 1$, reaching a contradiction to the theorem statement.

Validity. If all parties have the same input b, then each nonfaulty p'_i simulates all of the parties in P_i with the input b. This means that the run corresponds to a run in which all parties simulated by nonfaulty parties have the input b. From the Validity property of the original protocol, all simulated nonfaulty parties output b as well, and thus every nonfaulty p'_i output b after seeing that all of the parties in P_i output that value.

Agreement. Assume that two nonfaulty parties p'_i and p'_j output the non-$\perp$ value b_i and b_j respectively. Before doing so, each one saw that all of the parties simulated by it completed the protocol and that at least one of the parties simulated by p'_i and p'_j output b_i and b_j respectively. Those parties are simulated as nonfaulty parties, so $b_i = b_j$ from the Agreement property of the original protocol.

Termination. If each nonfaulty p'_i starts the protocol, it simulates all of the parties in P_i correctly throughout the whole protocol. This means that all of the parties in the P_i sets simulated by nonfaulty parties act as nonfaulty parties would in the original protocol, and thus eventually decide. After seeing that all of the parties in P_i output some value, every nonfaulty p'_i outputs a value as well.

Round Complexity. In the original n-party protocol, all parties output a value after two rounds. More precisely, all nonfaulty parties send only round 1 or round 2 messages. Observe a given run of the 4-party protocol. In the simulated n-party protocol, all simulated parties output a value after at most 2 rounds without sending any message from round 3 or higher. Therefore, in the 4-party protocol, no party sends a message from round 3 message or higher, and after every nonfaulty simulated party decides a value, every nonfautly p'_i outputs a value as well. ◀

▶ **Theorem 20.** *Assume there is a network of 7 parties $p_1, \dots, p_7$, and let S, T be a partitioning of the parties such that $|S| = 4, |T| = 3$, $S \cup T = \{p_1, \dots, p_7\}$ and $S \cap T = \emptyset$. Assume that it is impossible to solve Byzantine fault tolerant binding crusader agreement in two rounds*

[1] An alternative choice is to output $\perp$ only if all simulated parties did, and otherwise output some non-$\perp$ value.

with $n = 7$ parties and $f = 2$ faults, even if the adversary can corrupt at most one party in S and one party in T. Then it is impossible to construct such a protocol for any $f \geq 2$ and $4f > n > 3f$.

Proof. Assume by way of contradiction that such a protocol exists for some n, f such that $f \geq 2$ and $4f > n > 3f$. The proof follows a similar outline to the previous proof, simulating the n party protocol in the 7 party setting. Without loss of generality, assume that $S = \{p_1, \dots, p_4\}$ and that $T = \{p_5, \dots, p_7\}$. Since $4f > n > 3f$, there exists some $k \in [f-1]$ such that $n = 3f + k$.

We will now construct a protocol for 7 parties $p'_1, \dots, p'_7$. Start by partitioning the parties $\{p_1, \dots, p_n\}$ into 7 sets $P_1, \dots, P_7$. Each set in $P_1, \dots, P_4$ contains k parties for the k defined above, and each party in $P_5, \dots, P_7$ contains $f - k$ parties such that for every $i \neq j$, $P_i \cap P_j = \emptyset$. First, note that by definition $f > k > 0$ and thus also $f > f - k > 0$. This means that each of these sets has a positive number of parties, smaller than f. In addition, the total number of parties is $4 \cdot k + 3 \cdot (f - k) = 3f - 3k + 4k = 3f + k = n$. In other words, it is possible to partition the n parties into non-intersecting sets of these exact sizes.

From this point on, the simulation is exactly the same as in Theorem 19. Each party p'_i is in charge of simulating the parties in P_i. It starts the protocol by receiving its input x_i and simulating all of the parties in P_i starting the protocol with the same input x_i. Following that, if some simulated party $p \in P_i$ sends a message m to $q \in P_j$ it either delivers it immediately if $i = j$ or sends m to p'_j and signifies that p sent the message to q. Upon p'_j receiving a message m from p'_i saying that p sent that message to q, p'_j checks that $p \in P_i$ and $q \in P_j$. If that is the case, p'_j simulates q receiving that message from p. In all of the above discussion, by "simulating receiving the message" we mean that the simulating party runs the code that the simulated party would have run, and sends any messages according to the above description.

Once p'_i sees that all of the parties in P_i output some value, it outputs if at least one of the parties in P_i output $\bot$, p'_i outputs $\bot$ as well. Otherwise, it outputs some non-$\bot$ value that a party in P_i output. All that is left to do, is to show that the protocol is a 2-round protocol, resilient against a Byzantine adversary that controls at most one party in S and one party in T, reaching a contradiction. An adversary controlling at most one party in S and one party in T is in charge of simulating at most $f - k + k = f$ parties. This means that any run of the 7-party protocol corresponds to a run of the n-party protocol in which the adversary controls at most f parties, and the scheduling is the same as the one described in Theorem 19. Therefore, the simulated run terminates in two rounds and has the Validity, Agreement, Termination and Binding properties.

The proof that the 7-party protocol requires two rounds and that it has the Validity, Agreement and Termination properties is identical to the proof in Theorem 19 and is thus omitted. For the final property, Binding, assume some nonfaulty party p'_i outputs some value. At that point in time, it saw that all of the parties in P_i output values. All of those parties are nonfaulty, and thus from the Binding property of the n-party protocol, at that time there exists some value $b \in \{0, 1\}$ such that all nonfaulty parties output either b or $\bot$ in the n-party protocol. We will show that all nonfaulty parties output either b or $\bot$ in the 7-party protocol. Observe some nonfaulty party p'_j in the 7-party protocol. If it outputs the value $\bot$ from the protocol, the property holds. Otherwise, it output some value b' after seeing that at least one party $p \in P_j$ output b', and no party in P_j output $\bot$. From the Binding property of the n-party protocol, $b' = b$, and thus p'_j outputs b as well. ◀

D Crash Fault Tolerant Binding Crusader Agreement for Adaptive Inputs

In this section, we discuss our interest in crash fault tolerant protocols for binding crusader agreement that are secure even with *adaptive inputs*. By this, we mean that the binding property holds even if the adversary may adaptively choose the inputs of parties at any point in the execution of the protocol prior to scheduling their first actions. There are two advantages of using binding crusader agreement for adaptive inputs as a building block for crash fault tolerant asynchronous agreement protocols: efficiency and simplicity.

In [1], Abraham, Ben-David and Yandamuri show a simple framework for asynchronous agreement that uses a strong common coin (such that all parties see value $v \in \{0,1\}$ with probability $\frac{1}{2}$) and binding crusader agreement. Although the authors don't explicitly state it, the crash fault tolerant BCA protocol from [1] withstands an adversary that can adaptively choose the inputs of parties when they start the protocol. In fact, when this requirement is removed, we obtain the simpler 1 round protocol of Algorithm 3 for crash fault tolerant BCA. When the inputs of all parties are fixed prior to the start of the protocol, binding trivially follows from the fact that there is at most one value $v \in \{0,1\}$ such that $n-t$ parties start the protocol with value v. In fact, binding is only guaranteed in this protocol *if the inputs of all parties are fixed prior to the start of the protocol.*

To see why this matters, first we review how the asynchronous agreement protocol terminates with the original BCA protocol for adaptive inputs. With the original BCA protocol for adaptive inputs, the asynchronous agreement protocol takes at most 7 rounds of broadcast in expectation for all parties to terminate the protocol. This follows from a simple invariant: in any given round of the AA protocol, with probability $\frac{1}{2}$, the value of the common coin is equal to the value to which the adversary is bound in that round's BCA. In that case, all parties adopt the same value *est*, and they all decide that value in the next round in which the coin is again equal to that value. In other words, the original protocol requires a single good event to occur, which happens with constant probability in each round.

Now, consider what happens when we plug the BCA protocol from Algorithm 3 into the asynchronous agreement protocol of [1]. Since the BCA protocol is not binding when the adversary can adaptively choose the inputs of parties, we can no longer apply the same invariant to ensure termination. This is because the adversary can lag a party behind in the previous round of the AA protocol and choose its input to the next round's BCA. In this case, to argue termination it is necessary that two independent good events occur in two consecutive rounds, resulting in an AA protocol that requires more rounds of broadcast till termination and a more complex proof than the one presented in [1].

Algorithm 3 Asynchronous Binding Crusader Agreement for Crash Faults with Static Inputs

Input: x
1: send $\langle \text{val}, x \rangle$ to all
2: **upon** receiving $\langle \text{val}, * \rangle$ messages from $n-f$ parties:
3: **if** all the messages contain the same value x, decide x
4: **else**, decide $\perp$