Byzantine Agreement with Signature Tokens

Yuval Efron [1], Mahimna Kelkar [2], Saachi Mutreja [3], and Tim Roughgarden [1]

[1]IAS
[2]Purdue University
[3]Columbia University

Abstract

We investigate the power and limitations of quantum cryptography in distributed consensus protocols. Specifically, we study the fundamental problem of Byzantine Agreement in partial synchrony, in a setting in which each party has access to a finite number k of signature tokens. Classically, this problem is solvable only when less than one-third of the parties can be faulty (Dwork, Lynch, Stockmeyer 1988). We prove that protocols with signature tokens can bypass this lower bound by designing a protocol that remains secure against up to a $\frac{2^{k+1}-1}{2^{k+2}-1}$-fraction of faults. In particular, with $k = \Theta(\log n)$, where n denotes the number of parties, our protocol solves Byzantine Agreement in partial synchrony assuming only an honest majority. Conversely, we prove that $\Omega(\log n)$ signature tokens are necessary for such a result. Finally, for $k = 1$, we prove that the fault-tolerance threshold of $3/7$ achieved by our protocol is the best possible.

1 Introduction

Byzantine Agreement (BA), introduced in the seminal work of Lamport, Shostak, and Pease [LSP82], is a cornerstone problem in the theory of distributed computing, with broad practical applications. Intuitively, in a BA protocol, a set of n parties (of which f could be arbitrarily corrupt) each hold an input and must agree on a common output, subject to some validity condition. Over the past several decades, the BA problem has inspired an enormous body of research, and has been studied along a myriad of different axes, models, and settings [DS83; FLP85; FLM86; BG93; BJBO98; CKS05; KK06; Gar+07; Nak08; KS11].

One important axis of study is on the timing of communication between parties. In the synchronous setting, messages between honest parties are delivered within a known time bound Δ, while in the asynchronous setting, message delays can be arbitrarily long. The partially synchronous setting offers a middle ground: the network starts as asynchronous but transitions to synchronous after some global stabilization time (GST) that is a priori unknown [DLS88]. The partially synchronous setting has been broadly adopted in the academic literature and has also guided the design of many consensus protocols that have been deployed in practice [CL99; BKM18; Yin+19; Civ+22; CP23; LP+24; Civ+25; KSW25; Abr+26; CLPO26; Jal+26].

In the partially synchronous setting, a classical result of Dwork, Lynch, and Stockmeyer (DLS) [DLS88] establishes that no BA protocol can tolerate even a $\frac{1}{3}$-fraction of corrupt parties. Notably, the bound holds irrespective of the parties' classical cryptographic resources, including authenticated communication via digital signatures—one of the most widely adopted and arguably most useful cryptographic tools in distributed computing. This stands in contrast to many other settings, where authenticated communication, established via a Public-Key Infrastructure (PKI), can circumvent information-theoretic barriers, improve resilience, and enable more efficient protocols [DS83; FHW04; CKS05; KK06; FR09; Gor+14; Abr+19; Wan+20; MR21b; CEK25; Col+26; Len+26].

Preventing equivocation. At the heart of the DLS bound lies *equivocation*, i.e., the ability of a corrupt party to convey conflicting information to different parties. Importantly, a digital signature allows its recipient to verify who signed a message, but does not establish that the signer has refrained from signing a conflicting message. A corrupt party can sign two conflicting messages using its own signing key without breaking unforgeability of the digital signature scheme. Could there be some way to prevent or restrict such equivocation? An affirmative answer could potentially open the door to protocols that overcome the DLS lower bound.

Quantum information suggests a way forward. A classical signing key is a string that a corrupt party can copy and retain, even if the prescribed signing procedure erases it after use. In contrast, the *no-cloning theorem* rules out a general procedure for copying arbitrary unknown quantum states.

Quantum cryptography meets distributed computing. Turning such limitations of quantum information into cryptographic resources is a central aim of quantum cryptography. The past decade has witnessed tremendous activity in this area, yielding a rapidly expanding toolkit that exploits properties of quantum information unavailable classically [Wie83; BB84; Aar09; JLS18; Mah18; BFV20; Kre21; Zha21; MH25; SZ25]. Our goal in this work is to show that a modest amount of quantum power does, in fact, enable protocols that circumvent classical lower bounds in distributed computing. To this end, we ask: Can we encode the ability to authenticate messages in a quantum state, so that authenticating one message consumes the ability to authenticate another?

Signature tokens for granular non-equivocation. A simple and natural cryptographic prim-

itive that realizes this idea is a *signature token* [Shm22; BDS23]: a quantum state issued by a trusted authority that allows its holder to sign a message of its choice and is consumed in the process. Crucially, its security guarantee prevents even a malicious holder from producing valid signatures for two distinct messages using a single token. The quantum state is used locally, while the resulting signature and its verification key are classical. Thus, after setup, parties can employ this quantum resource while communicating entirely over classical channels.

Signature tokens provide a granular notion of non-equivocation: Each token guarantees non-equivocation for one signing operation, allowing us to study the possibility frontier as a function of the number of tokens. Specifically, we consider BA in the partially synchronous model with a PKI, which provides each party with an unlimited number of classical signatures. Additionally, each party is provided with k signature tokens, and can therefore choose at most k messages throughout the protocol to sign using them. Our work answers the following questions:

- Do signature tokens enable protocols that overcome the DLS lower bound of $\frac{1}{3}$? If so, how many are required?
- Can a sufficiently large number of signature tokens enable protocols with a fault-tolerance threshold of $\frac{1}{2}$, the barrier that applies already with a non-equivocating adversary (i.e., with omission faults only)? If so, how many are sufficient?

1.1 Our results

For our first result, we present a deterministic protocol for Byzantine Agreement (BA) in partial synchrony that, given a PKI and k signature tokens per party, is secure against up to a $\frac{2^{k+1}-1}{2^{k+2}-1}$-fraction of corrupt parties. This gives resilience $\frac{3}{7}$ with a single token and $\frac{7}{15}$ with two, approaching $\frac{1}{2}$ as k grows. In particular, $O(\log n)$ signature tokens per party suffice to tolerate any $f < \frac{n}{2}$.

Theorem 1.1 (Main theorem, See also Theorem 4.1)**.** *Let $k \geq 1$, and let n, f such that $f < \frac{2^{k+1}-1}{2^{k+2}-1}n$. Assuming PKI and a k-SigT setup, there exists a deterministic n-party* BA *protocol secure against f Byzantine faults in partial synchrony.*

Surpassing $\frac{1}{3}$. It might be tempting to think that the SigT setup trivially transforms BA with Byzantine faults to BA with omission faults, thereby changing the optimal fault-tolerance from $\frac{1}{3}$ to $\frac{1}{2}$. Our impossibility results (described below) prove that this is not the case. To develop some initial intuition about this, consider one obvious approach to employing a SigT for BA: Have each party sign its input with the SigT and then broadcast it. After all, the DLS attack hinges on a corrupt party being able to fool correct parties into having different opinions about its input to the protocol. This approach turns out to be fundamentally flawed. In fact, a careful look at the DLS attack itself can convince the reader that any protocol that uses its SigT in this manner cannot be secure against even a $\frac{1}{3}$ fraction of faults! We must therefore consider more intricate approaches to how the SigT power ought to be leveraged during a protocol.

Intuition for $k = 1$. The bound $\frac{3}{7}$, although peculiar at first glance, emerges naturally from the central idea behind our protocol. Intuitively, the SigT serves as the catalyst for a mechanism of *accountability*: We structure the protocol so that any attempt at an equivocation by the adversary leaves a cryptographic trace of guilt, allowing correct parties to identify a set of *provably corrupt*

parties, and expel them from the execution. Once these parties are ostracized, the corruption among the remaining parties falls below the classical $\frac{1}{3}$ threshold, which paves the way for a classical protocol to converge to consensus. A brief calculation shows this reduction goes through precisely when $f < \frac{3n}{7}$. Realizing this idea, and generalizing it to arbitrary k, demands considerable care and technical details, and we defer a fuller account to the technical overview in Section 3, and the full proof details to Section 4.

The lower bound. Our second result is a lower bound on the feasibility of BA with a PKI and a k-SigT setup. We first establish that our protocol's resilience is optimal for $k = 1$. The accountability idea underlying our protocol suggests an attack: the adversary uses part of its corruption budget to force parties to use their signature tokens while preserving executions with different correct decisions. We show that this attack is not specific to our construction, but applies to any protocol in this model.

Theorem 1.2 (One-token lower bound, informal. See also Theorem 5.1)**.** *With a PKI and a* 1-SigT *setup, no deterministic n-party* BA *protocol in partial synchrony tolerates $f \geq \frac{3n}{7}$ Byzantine faults.*

We then generalize the attack to arbitrary k. Together with our protocol, the resulting lower bound establishes an asymptotically tight exponential tradeoff: The gap between the optimal resilience and $1/2$ is $2^{-\Theta(k)}$. In particular, $\Theta(\log n)$ signature tokens per party are necessary and sufficient to tolerate $f < n/2$.

Theorem 1.3 (k-token lower bound, informal. See also Theorem 5.2)**.** *Let $k \geq 2$. With a PKI and a k-*SigT *setup, no deterministic n-party* BA *protocol in partial synchrony tolerates $f \geq \frac{2(4^k-1)}{4^{k+1}-1}n$ Byzantine faults.*

See Section 3.2 for the intuition and Section 5 for the detailed execution descriptions and proofs.

Structure of the paper. In Section 2, we provide the formal treatment of our model and problem definition. Section 3 contains a technical overview of our main results: Our protocol, and the corresponding lower bounds. Section 4 contains the full details, pseudo-code, theorem statement, and proof of our protocol. Section 5 contains the full details and proofs of our lower bounds.

1.2 Discussion and open problems

On black-boxing the SigT. Our model treats the SigT as a black box to isolate the distributed-computing core of the BA problem. The modeling choice of treating the underlying cryptography as ideal follows the convention in the vast majority of the consensus literature, with only a handful of exceptions even in the classical setting [Bad+18; Bad+24; Coh+25].[1] An interesting research direction, outside the scope of this paper, is to develop composable security notions for distributed computing (in both the quantum setting and the classical setting), in the spirit of the UC framework [Can20] or the fully composable quantum security proofs in e.g. [BOM04; Unr10].

[1] [Bad+18; Bad+24] mostly explore composable security notions for the task of consensus itself, rather than the use of ideal cryptographic primitives for consensus protocols.

What does circumventing DLS require? Our work shows that signature tokens are sufficient to circumvent the DLS lower bound. One can consider alternative ways of harnessing no-cloning for this purpose. For example, at the more powerful end lie one-shot signatures [Amo+20; SZ25]: in concurrent[2] work, Shmueli and Tennenhouse [ST26] showed that, together with a PKI, one-shot signatures used in every round of a BA protocol essentially restrict the adversary to omission failures, leading immediately to the fault-tolerance bound of $\frac{1}{2}$ known for omission adversaries. What is the weakest form of no-cloning that circumvents the DLS lower bound? We leave formalizing and answering this question as an open research direction.

Distinguishable versus indistinguishable SigT uses. In this work, each party's k SigT uses are *distinguishable*, indexed by $[k]$. One can also consider k uses that are *indistinguishable* from one another. Making the uses indistinguishable can only help the adversary, and thus cannot increase the optimal resilience. Our lower bounds therefore still apply, whereas our protocol for $k > 1$ relies on the indices. Determining the optimal resilience in this setting for $k > 1$ remains open.

Removing the PKI. Given the k-SigT setup already assumed, we view a PKI as a natural additional assumption. One can also consider the setting in which parties have the k-SigT setup but no PKI, while all else is information-theoretic. Determining the optimal resilience in this setting remains open.

2 Preliminaries

Model. We assume a peer-to-peer communication network, with a communication link between each pair of parties i, j among n parties ($V = [n]$ denotes the set of parties). At most f of the parties are corrupt, controlled by an *adversary* $\mathcal{A}$. We assume a permissioned setting, i.e., the identities of all parties are known to all parties, and all correct parties are active in the protocol at all times. We assume a Public Key Infrastructure (PKI), in which each party is handed a public/secret key pair (sk_i, pk_i) to be used throughout the protocol. We use the established PKI for digital signatures. We denote by λ the security parameter dictating the length of signatures. Without loss of generality, we assume that n is such that all fractions we consider are integers.

Partial synchrony & The adversary. We consider the standard partially synchronous model. Specifically, there exists an unknown Global Stabilization Time (GST), chosen by the adversary $\mathcal{A}$, that separates an initial asynchronous period from a subsequent synchronous one. The synchronous period is governed by a known upper bound Δ on message delays: any message broadcast at time τ is received by time $\max(\tau, \text{GST}) + \Delta$.

We further assume that all correct processes begin executing the protocol on or before GST, and that hardware clocks, while subject to arbitrary skew & drift before GST, do not drift after GST. We further allow the adversary, at the beginning of the protocol execution, and given the protocol code, to select a set $F \subseteq [n]$ of parties to become *corrupt*. Corrupt parties cede the entirety of their private state to the adversary, and are completely controlled by it, and in particular may exhibit arbitrary behavior throughout the execution of the protocol, to the discretion of the adversary. We say that an adversary $\mathcal{A}$ is f-bounded if $|F| \leq f$.

[2] See [Efr+26].

SigT setup. In order to distill the study of the power of signature tokens in partial synchrony, we incorporate them into our setting in a black-box manner. Formally, we assume the existence of an environment E that can sign messages with its own secret key, can interact with the parties, and provides to each party p a *one-time use* of the following interface.

- Use $\mathsf{SigT}(m)$: Send m to E, and E returns the message (m, p) signed by the environment.

As to not encumber notation, we simply denote the message returned by E to party p as $\mathsf{SigT}_p(m)$. More generally, for $k \geq 1$, a *k-SigT setup* provides each party p with k independently indexed one-time uses of this interface: for each $i \in \{1, \ldots, k\}$, using $\mathsf{SigT}^{(i)}(m)$ sends m to E, and E returns the message (m, p, i) signed by the environment, which we denote by $\mathsf{SigT}_p^{(i)}(m)$. Each index can be used once, regardless of the other indices, and the SigT setup above corresponds to $k = 1$.

On realizing the setup. While our results pertain to the black-box model introduced above, we briefly discuss a candidate trusted setup procedure that can securely realize it using signature tokens [BDS23]. In a tokenized signature scheme, the holder of a classical secret key can generate quantum signing tokens for a classical public key vk. Each token can be used to sign a single message, and signatures are verified with vk. Signature tokens are known relative to an oracle [BDS23], and in the plain model assuming post-quantum indistinguishability obfuscation and one-way functions [Col+21]. Our setup can be established via a trusted dealer D as follows.

1. D generates a pair (sk, pk) for a *classical* signature scheme, and distributes pk to all parties.
2. For each $i \in [n]$, D independently runs $(vk_i, sk_i^{\mathrm{T}}) \leftarrow \mathsf{key\text{-}gen}(1^\lambda)$ of the tokenized signature scheme to generate a classical public/secret key pair. It then runs $\mathsf{token\text{-}gen}_{sk_i^{\mathrm{T}}}$ exactly once to generate a signing token, sends the token to party p_i, and erases sk_i^{T}. It signs each of $vk_1, \ldots, vk_n$ using sk, and distributes the signed verification keys to all parties.
3. Throughout the protocol, the parties $p_1, \ldots, p_n$ only accept token signatures under verification keys signed by D, and ignore any other.

Since D generates a single signing token for each verification key and erases the corresponding secret keys, each party can sign only one message under its verification key. At least intuitively, the unforgeability of the classical scheme used by D prevents corrupt parties from fooling correct parties into accepting verification keys of their own. For the k-SigT setup, repeat this procedure independently k times, with D signing each verification key together with its party identity and token index. See Section 1.2 for a more detailed discussion on our choice to black-box and idealize this primitive.

Byzantine Agreement. In this work, we design a Byzantine Agreement (BA) protocol that operates in the model described above. We use its standard formulation in the partially synchronous setting. Formally, we solve the problem in which every correct party p holds an input $u_p \in \{0, 1\}$, and all correct parties aim to decide a value while satisfying the following properties:

- *Agreement:* No two correct processes decide different values.
- *Termination:* Every correct process eventually decides.

- *Validity:* If all parties are correct and input the same value u, then no correct party decides a value different from u.

We note that while various other validity constraints and problem formulations (such as Byzantine Broadcast) exist, we deliberately choose this BA formulation both because it isolates the core algorithmic challenges emerging in this novel setting, and because it is the formulation considered in the seminal work of [DLS88].

Remark. Another justification for our choice of validity notion is that the other popular validity notion for the BA task, that of strong unanimity, in which if all correct parties input the same value, they must output that value, is unattainable, even in our setting, in partial synchrony when $f \geq \frac{n}{3}$. This is witnessed by a straightforward DLS-style indistinguishability argument.

3 Technical Overview

In this section, we explain the chain of thought, and main techniques, that we employ in the design of our protocol, which establishes Theorem 4.1, and our lower bound results, Theorems 5.1 and 5.2.

3.1 Protocol Overview

We first focus on the case $k = 1$, in which each party has a single signature token.

Our starting point: Tendermint. Well, we are tasked with designing a partially synchronous consensus protocol. It would be prudent of us to commence this ascent on the shoulders of giants, and see whether we can build on top of the rich literature on such protocols. Two natural choices are Simplex [CP23] and Tendermint [BKM18]. A relevant difference is that Simplex also uses certificates for missed views. As will soon become evident from the overview, it is advantageous for us that every certificate we build on supports a value, and thus we proceed with Tendermint. For context, we quickly recap its high-level structure and key properties. Recall that Tendermint follows the *view-based* and *leader-based* approach: Time is divided into intervals, referred to as *views*, each view has a designated, predetermined party serving as the *leader* of the view, with leaders rotating between views in a round-robin fashion. In each view, the leader proposes a value, and the parties vote on it in two rounds. A party casts a $\mathsf{vote1}$ for the proposal unless doing so would contradict its *lock*, the most recent value it saw certified. A certificate of $n-f$ $\mathsf{vote1}$ messages for a value a, denoted V_a, makes a party lock on a and cast a $\mathsf{vote2}$ for it, and a certificate of $n-f$ $\mathsf{vote2}$ messages for a, denoted $\mathsf{V2}_a$, is a *decision* certificate for a.

When $f < \frac{n}{3}$, a quorum intersection argument allows Tendermint to boast the following exclusion properties between certificates in any execution of the protocol.

1. For any view v, two certificates $\mathsf{V}_a, \mathsf{V}_b$ of view v for values $a \neq b$ can not *both* form, and neither can two such $\mathsf{V2}$ certificates.

2. If a decision certificate $\mathsf{V2}_a$ formed in view v, then no certificate V_b can form in any view $v' > v$, for any value $b \neq a$.

These properties suffice for proving the correctness of Tendermint. Alas, we aim to design a protocol for the $f \geq \frac{n}{3}$ regime. At which point, *all* of these exclusion properties can be violated. And of course, in the standard partial synchrony model, with no SigT, this is not accidental, the seminal DLS result established the impossibility of solving BA in partial synchrony in the $f \geq \frac{n}{3}$ regime. While we are equipped with a SigT setup, it might seem unclear at first how exactly they could assist us in circumventing the DLS result.

Arguably the first clue comes from a key observation stemming from a closer look at the properties of the Tendermint certificates in the case when $f \geq \frac{n}{3}$. While we can no longer hope that the above exclusion conditions hold as is, we can hope to have an *accountable* version of the exclusion conditions. Namely, that if in some execution of Tendermint, any of the exclusion conditions are violated, then there exists at least $n - 2f$ corrupt parties S that *provably* misbehaved. In other words, for each party $q \in S$, the transcript of the execution contains a pair of incriminating messages, both signed by q, which no correct party could ever send. Intuitively, the $n-2f$ figure stems from each certificate containing contributions from $n-f$ parties, thus, a pair of certificates violating the exclusion conditions must have in their intersection at least $n - 2f$ parties that contributed to both.

For the sake of argument, suppose that by divine intervention, all correct parties were able to collectively identify the same set of $n-2f$ parties as corrupt. Then the correct parties could simply ostracize those parties from the protocol, and run an instance of BA in which $n' = n - (n - 2f) = 2f, f' = f - (n - 2f) = 3f - n$. When $f < \frac{3n}{7}$, we get that $3 \cdot (3f - n) < 9f - 7f = 2f$. Thus $f' < \frac{n'}{3}$, and so with (n', f'), the correct parties can simply achieve consensus by running your garden-variety partial synchrony BA protocol.

One may begin to perceive the emerging win-win situation coalescing at this point, though it is still far from tangible. If the exclusion conditions hold, then Tendermint provides consensus. If they don't,then $n - 2f$ parties are *provably* corrupt, and their removal from the protocol allows to reach consensus in partial synchrony. So far, of course, we haven't really said anything directly related to the SigT setup model. The above win-win observation holds even in standard partial synchrony. The gap lies not in the *existence* of $n - 2f$ provably corrupt parties, but in the correct parties' ability (or lack thereof) to *detect* and identify those parties. Evidently enough, the DLS result essentially shows that in standard partial synchrony, correct parties may never detect the corrupt parties, leading to an agreement violation. Here, precisely, is where SigT steps into the field.

SigT quorum intersection. While we can't apply a quorum intersection argument to obtain the exclusion conditions on certificates when $f \geq \frac{n}{3}$, we can apply it on certificates constructed from SigTs! Namely, consider defining a notion of a SigT-certificate for a value a. A SigT-certificate for a value a, amongst other conditions, includes SigT contributions for a from at least $n - f$ parties. Then, since $n - f > \frac{n}{2}$, and each party has exactly a single use of its SigT, we could confidently conclude that a SigT-certificate can exist for at most a single value. The road to obtaining such a certificate, however, is paved with several pitfalls, around which we must gingerly maneuver.

Challenges. The first obstacle we encounter is that, unlike with a PKI, the correct parties only have a *single* use of their SigT. The quorum intersection argument above for SigT-certificates is adequate for safety, but does nothing for *liveness*. In fact, the SigT being single-use is a significant challenge from a liveness point of view. Consider a toy execution between $n = 3$ parties A, B, C

with $f = 1$, A is corrupt. B has input 1, and C has input 0. Suppose that the adversary delays all communication between B, C while equivocating: Talking to B as if it is correct and has input 1 and not hearing from C (which eventually leads to B deciding 1), and talking to C as if it is correct with input 0 and not hearing from B (which eventually leads to C deciding $0 \neq 1$). For the sake of the example, suppose that in these interactions, B and C use their SigT *before* A. One can make the intuition formal, but you should be envisaging that B's SigT is contributing to a SigT-certificate for 1, while C's SigT is contributing to a SigT-certificate for 0. If A were to simply crash when both B, C have used their SigT, B, C would need to reconcile the situation between them, and in particular, without being able to form a SigT certificate for any value! Notice, however, that in order to reach this situation, A had to equivocate, and inject into the transcript of the protocol a proof of it being corrupt. Our first observation is that one can construct a protocol in which this is *not a coincidence*: We tie the use of the SigT to the accountable exclusion conditions discussed above. In essence, the goal is to achieve a situation in which every correct party p reaches one of the following states.

- p obtains a SigT-certificate for a value a. In our protocol, we refer to this state as allgood.
- p identifies a set S of size *at least* $n-2f$ of corrupt parties, along with a provable equivocation by each party in S. In our protocol, we refer to this state as equiv.

Our protocol for $k = 1$ consists of three parts (see Figure 1). The first part, which brings correct parties to converge on one of the above states, is the *Tendermint stage*.

The Tendermint stage. The skeletal structure of the Tendermint stage is that of Tendermint: A leader-based protocol, partitioned into views, during which the leader of the view proposes a value, on which the parties then perform multiple rounds of voting, and move on to the next view if sufficient time has passed with no successful completion of the view. Leaders rotate in a round-robin fashion. In more detail, each view v consists of a proposal by the leader of view v, leader(v), followed by three rounds of voting vote1, vote2, vote3. vote1, vote2 mimic their analogous Tendermint behavior, including the locks, and are issued upon receiving a valid proposal from leader(v), and upon observing $n - f$ vote1 messages for the same value, respectively. A vote3 message is also issued upon observing $n - f$ valid vote2 messages for the same value. Note that in the language of Tendermint, a vote3 message signals observance of a decision certificate. For that reason, a vote3 message carries an additional inter-view dependency: Once a correct party issues a vote3 for a value, it *never* issues a vote3 message for any other value. The final action in the logic of a view is precisely where the SigT comes in: If leader(v) collects $n - f$ vote3 messages for the same value, it uses its SigT on this certificate of vote3 messages, and broadcasts it. This is the first type of *valid* use of a SigT.

When a party collects $n - f$ SigT messages for the same value, i.e., a SigT-certificate, it reaches the allgood state. A key observation is the following, which precisely brings us to the desired convergence described above: If a correct party receives two valid SigT messages as above for two values $a \neq b$, then it can identify $n - 2f$ provably corrupt parties! This follows from the fact that each valid SigT for a value contains $n - f$ vote3 messages for a value, and a correct party only ever sends vote3 messages for at most one value.

An astute reader might notice that what one should worry about in the above logic is in fact not safety, but liveness! Namely, with the restriction on casting a vote3 message for at most a single value, it is entirely unclear that a correct party ever collects $n - f$ valid SigT messages. For this

reason, we augment the logic of the Tendermint stage with an additional *escape hatch* logic: If a correct party observes two conflicting certificates that provably implicate $n - 2f$ corrupt parties, they use their SigT to sign this pair of certificates, if they haven't used their SigT before, and forward them. This is the second type of a *valid* use of a SigT. They then reach the equiv state. This logic suffices to guarantee that every correct party eventually reaches one of the two states, as any two valid vote3 messages for different values imply the existence of conflicting certificates. Correct parties store the set of corrupt parties they identified in a variable denoted by blame.

This concludes the description of the Tendermint stage. At its end, every correct party converges to either allgood, or equiv. Alas, presently, there is nothing linking the two states between correct parties. In other words, if some correct party is in the allgood state with a value a, it could very well be that all other correct parties are in the equiv state, with no preference for a whatsoever. The next part of our protocol, the *conciliation stage*, precisely endows the states of allgood and equiv with a guarantee of *consistency* between correct parties.

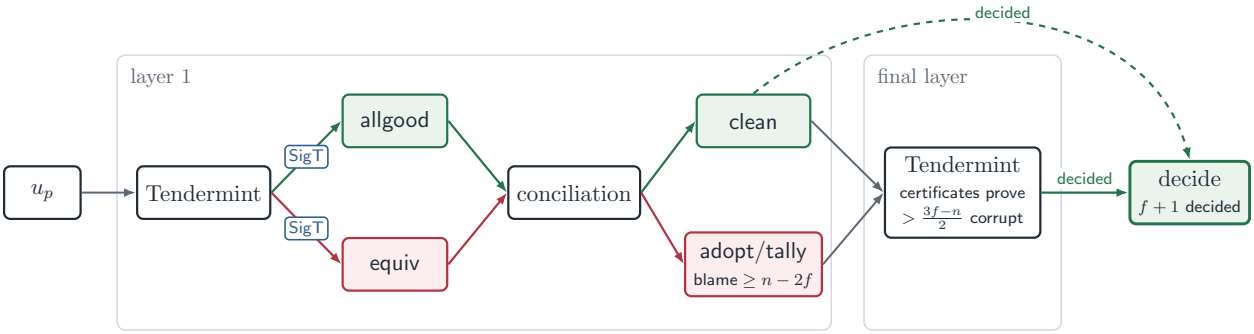

Figure 1: The flow of our protocol for $k = 1$, from the viewpoint of a single correct party. Each party uses its signature token on its way to the allgood or the equiv state; green marks the path on which no misbehavior is detected.

The conciliation stage. Correct parties start the conciliation stage in either the allgood state, or the equiv state. The first implies that it has collected a SigT-certificate for a value, while the second implies that it detected at least $n-2f$ provably corrupt parties. At the end of the conciliation stage, a correct party exits with an output of one of three types: clean, if it believes that all correct parties are on the allgood path, and adopt or tally otherwise. In any case, all correct parties advance to the last part of the protocol with the value they output. The conciliation stage endows the correct parties with two crucial properties.

- There cannot be a clean and a tally output for different values, nor, by SigT quorum intersection, a clean and an adopt output.
- Any correct party that outputs adopt or tally leaves with a blame set of size at least $n - 2f$. I.e., it detected at least $n - 2f$ provably corrupt parties.

Obtaining these guarantees involves a key technical idea, of favoring parties from which *you haven't heard from*, when choosing the value of a tally output. We now describe the logic of the conciliation stage.

In terms of communication between parties, the logic of the conciliation stage is very simple: Each party simply multicasts whether it is in the allgood or equiv state, with accompanying proof: A SigT-certificate for a value, or a valid use of a SigT along with a set of at least $n - 2f$ provably corrupt parties, respectively. From here, the rest of the logic of the conciliation stage is solely local computation, and proceeds as follows. First, a party waits to receive $n - f$ messages(either allgood or equiv) from parties in $[n] \setminus$ blame. If it receives $n - f$ allgood messages (with SigT-certificates for the same value a), and no equiv message, it outputs a as clean, and multicasts a decided message for a.

Note that in all other cases, a correct party has received at least one equiv message, and as such has received proof of the corruptness of at least $n - 2f$ parties, which it updates in its own blame variable. If a correct party receives at least one valid allgood message, it outputs the value of that message as adopt.

The remaining case is when all $n - f$ messages from parties in $[n] \setminus$ blame received by a correct party p are equiv messages, in which case p outputs tally. Now there are two cases for its value. The first is easy: if among the received equiv messages, all the SigT's received by p are either on a vote3 certificate for the same value, or all are on a pair of conflicting certificates, then p outputs that value (or its own input value).

The interesting case is when p receives at least two SigT messages, each signing a certificate of vote3 messages, for different values $a \neq b$. We denote by v3signers(a), v3signers(b) the sets of $n - f$ parties that cast a vote3 message in each of the SigT messages. p first filters from both sets the parties that appear in both, as they provably cast vote3 messages for two different values. It then also filters from each set those parties from which p *received an* equiv *message*. The intuition here is that p must be maximally alert to the possibility that another correct party output a clean value, which means that p must output the same value. As such, it must treat parties that it does not know to be corrupt & haven't heard from in the conciliation stage as if they have sent an allgood message which is still in flight. Parties that p has heard from that have sent an equiv message have essentially admitted to forfeiting the allgood path. After these two filtering steps of v3signers(a), v3signers(b), it outputs the value of the largest remaining set. The proof of correctness for this logic is technical, and can be found in Lemmas 4.9 and 4.12.

The final layer. The last part of the protocol is where we make good on the promise for the win-win intuition we described earlier: If all parties identify $n - 2f$ corrupt parties, they can essentially run any partially synchronous protocol secure against $\frac{1}{3}$-fraction of faults, once they ostracize these parties. Note that by the guarantees at the end of the conciliation stage, we are just shy of that condition. Every correct party has either output clean, or output adopt or tally and identified a set of at least $n - 2f$ provably corrupt parties. Had all parties agreed on the set of corrupt parties, as we explained before, they could simply exclude them from the protocol, and run e.g. Tendermint to finish the protocol. So essentially, the last hurdle we must deal with is that we have no guarantee that correct parties *agree* on the set of corrupt parties they identified. Our solution is a second and final layer, which runs the Tendermint stage once more, without any SigT, and in which every certificate must carry proofs of corruption against more than $\frac{3f-n}{2}$ parties, and count only the votes of other parties, an idea inspired by the recent work of [LPR25]. Two conflicting certificates would then implicate more than $(n - 2f) + (3f - n) = f$ corrupt parties, so they cannot exist. A party that outputs a clean value, or receives a decision certificate $\mathsf{V2}_a$ of the final layer, multicasts a decided message for its value, and a party decides a once it receives $f + 1$ decided messages for

a. If all correct parties output clean values, they all decide. Otherwise, some correct party holds proofs against $n - 2f > \frac{3f-n}{2}$ parties, as $f < \frac{3n}{7}$, and its proposals in the final layer carry these proofs, so a view it leads after GST produces valid certificates, and every correct party decides. Finally, every party enters the final layer with the value it output, together with a *justification*: evidence of how it obtained that value, which all its messages carry. If any correct party output a clean value a, then a is the only value with a valid justification, so every decision certificate of the final layer is for a. This maintains agreement between parties that output a clean value, and parties that decide in the final layer.

This concludes the technical description of the three parts of our protocol for $k = 1$: the Tendermint stage, the conciliation stage, and the final layer. See Section 4 for the full details and proofs.

Generalizing to $k > 1$. Now that we have described the case $k = 1$, let us briefly describe how we generalize our protocol to $k > 1$. With k tokens, a party runs k *layers* before the final layer, each consisting of a Tendermint stage and a conciliation stage as above, where layer i uses the party's i-th token, and the party's output value and justification from layer i serve as its input to layer $i+1$. Intuitively, layer 1 ends either in convergence to a decision, or in the detection of $n-2f$ corrupt parties, and one should think of every subsequent layer in the same way: It either converges to a decision, or detects further corrupt parties. Crucially, later layers detect more corrupt parties. Certificates carry the proofs of corruption gathered in their value's history, and count only the votes of the remaining parties, so if D corrupt parties were detected so far, a conflict detects $n' - 2f'$ further ones among the remaining $n' = n - D$ parties, of which $f' = f - D$ are corrupt, that is, $n - 2f + D$. The number of detected corrupt parties thus grows as $n - 2f$, $3(n - 2f)$, $7(n - 2f)$, and so on, reaching $(2^k - 1)(n - 2f)$ after layer k, which exceeds the $\frac{3f-n}{2}$ that the final layer requires precisely when $f < \frac{2^{k+1}-1}{2^{k+2}-1}n$. Of course, additional complications arise, e.g., when some correct parties converge to a decision while others detect equivocations, and handling them requires further ideas. We defer the full details to Section 4.1, and the proofs to Section 4.2.

3.2 Lower bound

A generic attack on the layer protocol. Write $q = n-2f > 0$, and suppose both input values occur among correct parties. In layer 1, q corrupt parties suffice for a DLS-style attack: partition the remaining parties into two sets of size f, and let the q parties equivocate towards both sets, creating conflicting V certificates with $n - f$ signers each. Before GST, the adversary delivers both certificates to every correct party while delaying further votes, so all correct parties use their tokens to send equiv messages and enter the next layer with their inputs unchanged. In layer 2, the q identified corrupt parties are excluded, and the same attack requires $2q$ additional corrupt parties; in layer 3, it requires $4q$, and so on. Thus, by revealing $q + 2q + \cdots + 2^{k-1}q = (2^k - 1)q$ corrupt parties, the adversary can make all correct parties exhaust their tokens without forming an allgood certificate. If $(2^k - 1)q \leq (3f - n)/2$, equivalently $f \geq \frac{2^{k+1}-1}{2^{k+2}-1}n$, the corrupt parties can then remain silent and GST can occur. No final-layer certificate can form, since it requires proofs against more than $(3f - n)/2$ corrupt parties, and termination fails.

Extending the attack to arbitrary protocols. Our lower bounds establish that a similar attack applies beyond the layer protocol. With sufficiently many corrupt parties, we can force a

set of parties to use their signature tokens while preserving two executions extending the same prefix with different correct decisions. Once these tokens are exhausted, a DLS-style equivocation attack yields a contradiction. For $k = 1$, Theorem 5.1 gives the tight threshold $f < 3n/7$. For general k, Theorem 5.2, together with our protocol, shows that $\Theta(\log n)$ signature tokens per party are necessary and sufficient to tolerate $f < n/2$ faults, although the exact resilience threshold for $k > 1$ remains open. We refer to Section 5 for a detailed overview of the lower bound arguments, executions, and the proofs.

4 k-SigT Protocol

In this section we present the k-SigT protocol and prove its correctness. Specifically, we prove the following theorem.

Theorem 4.1. *Let $k \geq 1$ and $f < \frac{2^{k+1}-1}{2^{k+2}-1}n$. Then, assuming a PKI and a k-SigT setup, there exists a deterministic f-secure* BA *protocol in partial synchrony.*

Algorithm 1 is the main protocol. It runs Layer (Algorithm 2) for layers $1, \ldots, k+1$ in order, feeding each layer's output into the next layer, and handles the decision and termination logic.

4.1 Protocol Overview

Protocol flow & structure. Layers $1, \ldots, k$ share the same structure, and layer i uses each party's i-th signature token, which we call its token in that layer. A party enters layer i with an input value and an input *justification* for it, which certifies how the party obtained that value in layer $i-1$. A layer $i \leq k$ consists of a Tendermint stage, followed by an allgood/equiv stage, at the end of which the party outputs a value with a justification for it, and enters layer $i+1$ with them. Layer $k+1$ uses no signature token, and brings the parties to a decision, if they have not reached one in layer k. We next describe each of these components.

1. **The Tendermint stage.** Each layer starts with a view-based, leader-based protocol that proceeds in views, and whose logic resembles Tendermint [BKM18]. Each view v has a designated leader leader(v), with leaders rotating in a fixed round-robin order. Every party holds a *lock*, a value together with evidence that it is safe: initially, its input value with its input justification, and later, the most recent V certificate it received. When a party enters a view, it sends its lock to the leader in a suggest message. A leader whose token is still unused proposes the value of the most recent lock it received, and a party casts a vote1 for the proposal if the proposal's lock is at least as recent as its own. A V_a certificate, consisting of $n - f$ vote1 messages for a, makes a party lock a and cast a vote2, and a $\mathsf{V2}_a$ certificate makes it cast a vote3, unless it has already cast a vote3 for, or used its token on, the other value. Upon collecting a $\mathsf{V3}_a$ certificate, the leader uses its token, if still unused, to sign it, producing a value-$\mathsf{SigT}^{(i)}$ for a, and $n - f$ value-$\mathsf{SigT}^{(i)}$ messages for a form a $\mathsf{SigT}^{(i)}$ certificate for a. Since each token signs only once, $\mathsf{SigT}^{(i)}$ certificates exist for at most one value (Observation 4.7). A view ends when a party receives a V certificate of the view, or when $n - f$ parties time out on it. Locks also carry the history of values across layers: since they start from input justifications, and every vote1 carries the leader's proposal with its lock, every certificate of layer i for a value contains a justification for that value from layer $i-1$, and, through it, the value's history in earlier layers (Lemma 4.8).

2. **The allgood/equiv stage.** The endgame of the Tendermint stage is for each correct party to reach one of two states, and to announce it in an allgood or equiv message, using its token if still unused. We prove in Lemma 4.21 and Observation 4.6 that, in every layer, every correct party eventually reaches exactly one of these two states.

   (a) **The allgood state.** This is the happy path, in which no misbehavior has been detected. A party reaches it upon receiving a $\mathsf{SigT}^{(i)}$ certificate for a value a, provided it has received no conflicting certificates, and has not cast a vote3 for, or used its token on, the other value. Its allgood message carries the certificate and its own value-$\mathsf{SigT}^{(i)}$ for a.

   (b) **The equiv state.** A party reaches it upon receiving conflicting certificates, such as two V or two V2 certificates for different values of the same view. Their common signers provably misbehaved, and the two certificates identify at least $n - 2f$ corrupt parties (Claim 4.4). Its equiv message carries the conflicting certificates and its $\mathsf{SigT}^{(i)}$. Throughout the protocol, each party stores proofs against every party it finds corrupt in its blame set.

3. **The output.** Once a party has announced its state and received allgood/equiv messages from $n - f$ parties outside its blame, it outputs a value with a justification of one of three types.

   (a) clean**.** The party received no equiv message, so the $n - f$ messages are allgood messages, all for the same value a. It outputs a, and in layer k, it also multicasts $\langle \mathsf{decided}, a \rangle$.

   (b) adopt**.** The party received an allgood message for some value a, as well as an equiv message. It outputs a.

   (c) tally**.** The party received only equiv messages. If it received value-$\mathsf{SigT}^{(i)}$ messages for both values, it compares their support: for each value, it takes the signers of a V3 certificate for it, and discards those who also signed the other value's certificate or sent one of the $n - f$ equiv messages. The value with more remaining signers wins. If it received a value-$\mathsf{SigT}^{(i)}$ for a single value, it outputs that value, and if for none, its input value. In layer k, a party that received an allgood certificate from layer $k - 1$ outputs that certificate's value instead.

   No two clean or adopt outputs in the same layer are for different values, by quorum intersection: each carries a $\mathsf{SigT}^{(i)}$ certificate for its value, and two such certificates for different values would require some party to use its token twice (Observation 4.7). A tally output, however, may be for a different value than a clean output. We prove in Lemma 4.9 that this comes at a dire cost to the adversary: it reveals at least $(3n - 5f)/2$ corrupt parties.

4. **Justifications and the next layer.** A justification certifies how its party obtained its output value. It contains the party's own allgood/equiv message, its blame, and the messages that determined the output's type and value. The party enters layer $i + 1$ with its output value as input and the justification as input justification, which all its messages in layer $i + 1$ carry. A value can be proposed in layer $i + 1$ only together with a justification for it from layer i. Two features make the layers compound. First, a party's own allgood/equiv message from layer i carries its $\mathsf{SigT}^{(i)}$, which signs the party's input justification for layer i, a justification from layer $i - 1$. Hence all of the party's messages from layer $i + 1$ on carry the same justification from layer $i - 1$. Second, since certificates carry the histories of their values, conflicting certificates of layer $i + 1$ identify not only the at least $n - 2f$ parties that signed

both, but also the corrupt parties that each of them identifies through earlier layers. Thus, as long as no allgood certificate forms, the number of identified corrupt parties grows from layer to layer as $T_1 = 0$, $T_{i+1} = n - 2f + 2T_i$, and conflicting certificates of layer i identify at least $\min\{(3n - 5f)/2, T_{i+1}\}$ corrupt parties (Lemma 4.11).

5. **The last two layers.** Layer k follows the same structure, except that a clean output also triggers a decided message, and a tally output follows an allgood certificate from layer $k - 1$ if the party received one. Together with the histories carried by certificates, this ensures that once an allgood certificate for a exists in layer k, every justification from layer k is for a (Lemma 4.12), since a justification for another value would identify more than f corrupt parties. This is where the bound on f comes from: it is equivalent to $(3n - 5f)/2 + 2T_k > f$.

   Layer $k + 1$ brings the remaining parties to a decision. If every correct party outputs clean in layer k, they all multicast decided messages for the same value and decide. Otherwise, for the sake of intuition, suppose for a moment that all correct parties held the same set B of more than $(3f - n)/2$ provably corrupt parties. Then $3(f - |B|) < n - |B|$, and the proper course of action becomes obvious: remove the parties in B, and run Tendermint, which tolerates fewer than a third of corrupt parties, among the rest. The correct parties cannot be assumed to agree on their blame sets, so instead, layer $k + 1$ runs only the Tendermint stage, without tokens or vote3 messages, and a certificate is valid only if it carries proofs against more than $(3f - n)/2$ parties. Two conflicting certificates would then identify more than f corrupt parties, so none exist (Lemma 4.13). A correct party that did not output clean in layer k holds such a set in its blame (Lemma 4.11), and its propose messages in layer $k + 1$ carry the proofs, so the views it leads produce valid certificates (Lemma 4.24). A party that receives a V2 certificate of layer $k + 1$ multicasts a decided message for its value. If an allgood certificate for a exists in layer k, this value is a, since every justification from layer k, and hence every input to layer $k + 1$, is then for a.

4.2 Protocol & Proof

Algorithm 1 k-SigT protocol (for party p)

Input: u_p.
$J_p \leftarrow \perp$.
for $i = 1, \ldots, k + 1$ **do**
    $(\cdot, u_p, J_p) \leftarrow \mathsf{Layer}(i, u_p, J_p)$.
end for

upon received $f + 1$ $\langle \mathsf{decided}, a \rangle$:
    Decide a and terminate.

Inputs. Party p invokes $\mathsf{Layer}(i, u_p, J_p)$ with a layer index $i \in \{1, \ldots, k+1\}$, an input $u_p \in \{0, 1\}$, and an input justification J_p for u_p.

Views and signatures. Each layer proceeds in views $v = 0, 1, \ldots$, with $\mathsf{leader}(v)$ rotating through the parties in a fixed round-robin order. $\mathsf{SigT}^{(i)}$ denotes a signature using a party's i-th signature

token; all other messages carry classical signatures. In layer $k+1$, parties use no SigT and send no vote3, allgood, or equiv messages.

Message forwarding. Throughout the k-SigT protocol, correct parties forward all received messages (which they didn't forward beforehand). We omit the explicit mention of this forwarding in the protocol pseudocode to maintain simplicity and clarity of exposition.

Nomenclature. For the analysis of the k-SigT protocol, we require defining the notion of a message *existing* in a given execution. For a protocol Π, a message m, and an execution E of Π, which is entirely determined by the adversary and their choices of corrupt parties, their behavior, GST, and network delays, we say that m exists in execution E if there exists a correct party that receives m during E before terminating. When referring to membership, cardinality, or set operations on blame, we implicitly refer to the set of blamed parties represented by the stored proofs of maliciousness.

Certificates. A certificate σ consists of $n-f$ valid messages from distinct parties outside its corrupt set D_σ, the set identified by all corruption proofs in σ. A V_a, $\mathsf{V2}_a$, or $\mathsf{V3}_a$ certificate of view v consists of vote1, vote2, or vote3 messages, respectively, for a of view v; a $\mathsf{SigT}^{(i)}$ or allgood certificate for a consists of value-$\mathsf{SigT}^{(i)}$ or allgood messages for a. In layer $k+1$, a certificate is valid only if, in addition, $|D_\sigma| > (3f-n)/2$. The validity of a certificate thus depends only on its contents. Conflicting certificates are two V certificates, or two $\mathsf{V2}$ certificates, for different values of the same view, or a $\mathsf{V2}_a$ certificate of view v and a V_b certificate for $b \neq a$ of a later view whose vote1 messages contain locks from views before v, all of the same layer. All instructions to form a certificate require that the resulting certificate be valid.

Corruption proofs. A party is proved corrupt by its messages in both of two conflicting certificates, or by two of its messages from one layer that are: vote1 messages, or vote2 messages, for different values in the same view; vote3 or value-SigT messages for different values; an allgood message for a and a vote3 or value-SigT message for $b \neq a$; or an allgood and an equiv message.

Justifications. Validity is defined recursively, and parties ignore invalid messages. In layer 1, $\perp$ is an input justification for either value. For $i > 1$, an input justification of party p for a in layer i contains p's own allgood/equiv message from layer $i-1$, its blame, which contains every party the justification proves corrupt, and one of the following from layer $i-1$:

1. clean: $n-f$ allgood messages for a from parties outside its blame.
2. adopt: an allgood message for a and an equiv message, one of which is p's own, and $n-f$ allgood/equiv messages from parties outside its blame.
3. tally: $n-f$ equiv messages from parties outside its blame, and the evidence and choices with which lines 54–65 give a, where u_p on line 59 is the value justified by p's input justification for layer $i-1$ in p's own allgood/equiv message (either value if it is $\perp$).

Every message of layer i except decided carries its sender's input justification for layer i. In addition, the following messages carry the listed justification:

- propose for a of view v: a lock for a from a view $v' < v$, that is, an input justification for a if $v' = -1$, and a V_a certificate of view v' otherwise. vote1 for a of view v: a valid propose for a of view v from leader(v). suggest of view v: a lock from a view before v.
- vote2 and vote3 for a of view v: a V_a and a $\mathsf{V2}_a$ certificate of view v, respectively.
- Value-$\mathsf{SigT}^{(i)}$ for a: a $\mathsf{V3}_a$ certificate or a $\mathsf{SigT}^{(i)}$ certificate for a.
- allgood for a: a $\mathsf{SigT}^{(i)}$ certificate for a and its sender's value-$\mathsf{SigT}^{(i)}$ for a.
- equiv: conflicting certificates of layer i and its sender's $\mathsf{SigT}^{(i)}$.

Party p outputs from layer i a tuple (type, a, J), where J is an input justification for a in layer $i+1$ of type type.

Algorithm 2 Layer(i, u_p, J_p) (for party p)

Local variables: lock $\leftarrow (u_p, J_p, -1)$, castVoteThree $\leftarrow \perp$, ownSigT $\leftarrow \perp$. Let $C \leftarrow 20\Delta$. blame $\leftarrow$ all parties proved corrupt by J_p.

Enter view 0.

upon received evidence proving a party q corrupt:
    blame $\leftarrow$ blame $\cup \{q\}$.

upon entering view v:
    Start local timer T_v, and send $\langle \mathsf{suggest}, i, \mathsf{lock}, v, J_p \rangle$ to leader(v).
    if $p =$ leader(v) and ownSigT $= \perp$ **then**
        Wait 2Δ. Let L be a lock with the highest view among the received suggest messages for view v, and $a \leftarrow \mathsf{val}(L)$.
        If at least $n - f$ such messages were received, multicast $\langle \mathsf{propose}, i, a, L, v, J_p \rangle$.
    end if

upon received the first valid propose P from leader(v) for a, with lock L of view v':
    if $v' \geq$ View(lock) **then**
        Multicast $\langle \mathsf{vote1}, i, a, v, P, J_p \rangle$.
    end if

upon received a valid V_a certificate in view v:
    Set lock $\leftarrow (a, \mathsf{V}_a, v)$ if $v >$ View(lock).
    if have not sent a vote2 message in view v **then**
        Multicast $\langle \mathsf{vote2}, i, \mathsf{V}_a, a, v, J_p \rangle$.
    end if
    Exit view v and enter view $v + 1$.

upon received $n - f$ valid vote2 messages for a in view v, and previously sent vote2 for a in v:
▷ Monitor even after exiting view v.
    if castVoteThree $\in \{\perp, a\}$ and ownSigT is either unused or for a **then**

22: Form $\mathsf{V2}_a$; set $\mathsf{castVoteThree} \leftarrow a$.
23: Multicast $\langle \mathsf{vote3}, i, \mathsf{V2}_a, a, v, J_p \rangle$.
24: **end if**

25: **upon** $i = k + 1$ and received a valid $\mathsf{V2}_a$ certificate of view v: ▷ Monitor even after exiting view v.
26: Multicast $\langle \mathsf{decided}, a \rangle$ and **return** .

27: **upon** received $n - f$ valid viewchange messages for view v:
28: Exit view v and enter view $v + 1$.

29: **upon** $T_v = C$:
30: Multicast $\langle \mathsf{viewchange}, i, v, J_p \rangle$.

31: **upon** received $n - f$ valid vote3 messages for a in view v, and $p = \mathsf{leader}(v)$: ▷ Monitor even after exiting view v.
32: **if** $\mathsf{ownSigT} = \perp$ and $\mathsf{castVoteThree} \in \{\perp, a\}$ **then**
33: Form $\mathsf{V3}_a$; set $\mathsf{ownSigT} \leftarrow \mathsf{SigT}^{(i)}(\mathsf{value}, a, \mathsf{V3}_a, J_p)$.
34: Multicast ownSigT.
35: **end if**

36: **upon** received conflicting certificates σ_a, σ_b, and have not sent an allgood or equiv message:
37: **if** $\mathsf{ownSigT} = \perp$ **then**
38: $\mathsf{ownSigT} \leftarrow \mathsf{SigT}^{(i)}(\mathsf{equiv}, \sigma_a, \sigma_b, J_p)$.
39: **end if**
40: Multicast $\langle \mathsf{equiv}, i, \mathsf{ownSigT}, \sigma_a, \sigma_b, J_p \rangle$.

41: **upon** received a $\mathsf{SigT}^{(i)}$ certificate σ for a, no conflicting certificates, $\mathsf{castVoteThree} \in \{\perp, a\}$, ownSigT unused or for a, and have not sent an allgood or equiv message:
42: **if** $\mathsf{ownSigT} = \perp$ **then**
43: $\mathsf{ownSigT} \leftarrow \mathsf{SigT}^{(i)}(\mathsf{value}, a, \sigma, J_p)$.
44: **end if**
45: Multicast $\langle \mathsf{allgood}, i, a, \mathsf{ownSigT}, \sigma, J_p \rangle$.

46: **upon** sent an allgood or equiv message, and received valid allgood/equiv messages from $n - f$ parties outside blame:
47: $J \leftarrow p$'s own allgood/equiv message, blame, and these $n - f$ messages.
48: **if** no equiv message was received **then**
49: $\mathsf{type} \leftarrow \mathsf{clean}$, and $a^* \leftarrow$ the value of these messages.
50: If $i = k$, multicast $\langle \mathsf{decided}, a^* \rangle$.
51: **else if** an allgood message for some a was received **then**
52: $\mathsf{type} \leftarrow \mathsf{adopt}$, $a^* \leftarrow a$, and add this allgood message and a received equiv message to J.
53: **else**
54: Let $\mathcal{V}$ be the values of received valid value-$\mathsf{SigT}^{(i)}$ messages.

55: Let $\mathsf{equivSenders}$ be the senders of these $n - f$ messages.
56: **if** $i = k$ and p received an $\mathsf{allgood}$ certificate for some a from layer $k - 1$ **then**
57: $a^* \leftarrow a$.
58: **else if** $\mathcal{V} = \emptyset$ **then**
59: $a^* \leftarrow u_p$.
60: **else if** $\mathcal{V} = \{a\}$ **then**
61: $a^* \leftarrow a$.
62: **else**
63: For each $a \in \mathcal{V}$, choose any received valid value-$\mathsf{SigT}^{(i)}$ for a and a layer-i $\mathsf{V3}_a$ certificate in its evidence, with signers P_a.
64: $S_a \leftarrow P_a \setminus \left(\bigcup_{b \in \mathcal{V} \setminus \{a\}} P_b \cup \mathsf{equivSenders}\right)$ for each $a \in \mathcal{V}$.
65: $a^* \leftarrow \arg\max_{a \in \mathcal{V}} |S_a|$, breaking ties in favor of 0.
66: **end if**
67: $\mathsf{type} \leftarrow \mathsf{tally}$, and add the evidence and choices used on lines 54–65 to J.
68: **end if**
69: **return** (type, a^*, J).

We next prove Theorem 4.1 by showing that Algorithm 1 satisfies agreement, termination, and validity. Throughout the analysis, fix an f-bounded adversary $\mathcal{A}$, for $f < \frac{2^{k+1}-1}{2^{k+2}-1}n$, and let F be the set of corrupt parties. Define $T_1 = 0$ and $T_{i+1} = n - 2f + 2T_i$, so that $T_i = (2^{i-1} - 1)(n - 2f)$; the bound on f is equivalent to $7f < 3n + 4T_k$, and it implies $f < n/2$. For a certificate or justification X, its corrupt set D_X is the set of parties proved corrupt by X. A *justification from layer j* is an input justification for layer $j + 1$. The *input conditions* of layer i are that every correct party p invokes $\mathsf{Layer}(i, u_p, J_p)$ with J_p a valid input justification for u_p and, if $i \leq k$, its $\mathsf{SigT}^{(i)}$ unused. Statements about a layer i assume its input conditions, which hold in every layer by Lemma 4.23 and the paragraph after it.

4.2.1 Agreement

Claim 4.2. *Let p be a correct party. In any execution of* Layer*, p multicasts* vote3 *messages for at most a single value. If p multicasts an* allgood *message for a, then it does not multicast a* vote3 *message for any $b \neq a$ in this layer.*

Proof. A vote3 can only be cast at line 23. The first time this occurs, castVoteThree is set to the value of this vote3, after which the condition for casting a vote3 fails for every other value. Furthermore, the condition on line 41 requires $\mathsf{castVoteThree} \in \{\perp, a\}$, so no vote3 for $b \neq a$ precedes the allgood message, and it requires ownSigT to be unused or for a, after which ownSigT is for a, so no vote3 for $b \neq a$ follows it. □

Claim 4.3. *Let p be a correct party, and let v be a view during* Layer*. Then p multicasts at most a single* vote2 *message in view v and at most a single* vote1 *message in view v.*

Proof. For vote2, the claim follows from the condition guarding line 17. For vote1, the claim follows from the first-valid-proposal condition on line 10. □

Claim 4.4. *Let σ_a, σ_b be conflicting certificates of layer i, with corrupt sets $D_{\sigma_a}, D_{\sigma_b}$. Then σ_a and σ_b together identify $B = D_{\sigma_a} \cup D_{\sigma_b} \cup (\mathsf{signers}(\sigma_a) \cap \mathsf{signers}(\sigma_b)) \subseteq F$, and $|B| \geq n - 2f + |D_{\sigma_a}| + |D_{\sigma_b}|$.*

Proof. No correct party appears in $\mathsf{signers}(\sigma_a) \cap \mathsf{signers}(\sigma_b)$. For certificates of the same view, this follows from Claim 4.3. Otherwise, they are a $\mathsf{V2}$ certificate of a view w and a V certificate of a later view whose $\mathsf{vote1}$ messages contain locks from views before w. A correct party sets its lock to view w before multicasting a $\mathsf{vote2}$ in w, and its lock view never decreases, so by line 11 it multicasts no such $\mathsf{vote1}$. The other parties in B lie in $D_{\sigma_a} \cup D_{\sigma_b}$. Let $P_a = \mathsf{signers}(\sigma_a)$ and $P_b = \mathsf{signers}(\sigma_b)$; then $|P_a| = |P_b| = n - f$, $P_a \cap D_{\sigma_a} = \emptyset$, and $P_b \cap D_{\sigma_b} = \emptyset$. Thus $P_a \cap P_b$ is disjoint from $D_{\sigma_a} \cup D_{\sigma_b}$, and $P_a \cup P_b$ is disjoint from $D_{\sigma_a} \cap D_{\sigma_b}$. Consequently, $|P_a \cap P_b| \geq n - 2f + |D_{\sigma_a} \cap D_{\sigma_b}|$. Adding $|D_{\sigma_a} \cup D_{\sigma_b}|$ proves the claimed bound. □

Lemma 4.5. *Let σ_a, σ_b be valid $\mathsf{V2}$ certificates of layer i for values $a \neq b$ and views $w \leq w'$, respectively. Then σ_a and σ_b contain conflicting certificates.*

Proof. If $w = w'$, then σ_a and σ_b conflict. Otherwise, let τ be a V_b certificate contained in σ_b with the least view greater than w; one exists, since σ_b contains a V_b certificate of view w'. Each $\mathsf{vote1}$ in τ contains a lock that is itself contained in σ_b, so by minimality the lock is from a view at most w. If some lock is a V_b certificate of view w, it conflicts with a V_a certificate of view w in σ_a. Otherwise, all of these locks are from views before w, and σ_a and τ conflict. □

Observation 4.6. *In a layer $i \leq k$, a correct party multicasts at most one* $\mathsf{allgood}/\mathsf{equiv}$ *message, and before returning it has multicast one and used its* $\mathsf{SigT}^{(i)}$.

Proof. The conditions on lines 36 and 41 both require that no $\mathsf{allgood}/\mathsf{equiv}$ message has been sent, and each of these two handlers uses $\mathsf{SigT}^{(i)}$ if it is unused. For $i \leq k$, every return requires that such a message has been sent. □

Observation 4.7. *In any execution of* Layer, $\mathsf{SigT}^{(i)}$ *certificates can exist for at most a single value. In particular,* clean *and* adopt *outputs cannot be for different values in the same layer.*

Proof. A $\mathsf{SigT}^{(i)}$ certificate for a contains $n - f > n/2$ valid value-$\mathsf{SigT}^{(i)}$ messages for a. Two such certificates for different values would require a party to use its $\mathsf{SigT}^{(i)}$ twice. A clean or adopt output for a includes an $\mathsf{allgood}$ message for a, which contains a $\mathsf{SigT}^{(i)}$ certificate for a. □

Lemma 4.8. *Let X be a valid certificate for a of layer $m \leq k$, or a valid justification for a from layer $m < k$.*

1. *For every $j < m$, X contains a justification for a from layer j. If there exists an* $\mathsf{allgood}$ *certificate from layer j for a value other than a, every justification for a from layer j in X is of type* tally.

2. *If X contains an* adopt *or* tally *justification from layer $m-1$ and no* $\mathsf{allgood}$ *certificate from a layer before $m-1$, then $|D_X| \geq T_m$.*

Proof. We first prove that X contains a justification for a from every layer $j < m$. We use induction on m; for $m = 1$ there is nothing to prove. For $m > 1$, we prove it for certificates of layer m, by reducing it to a justification from layer $m-1$, and then for justifications from layer m, by reducing it to a certificate of layer m or a justification from layer $m - 1$.

Let X be a certificate for a of layer m. Then X either is a V_a certificate of layer m or contains one. For an $\mathsf{allgood}$ or $\mathsf{SigT}^{(m)}$ certificate, this holds because it contains value-$\mathsf{SigT}^{(m)}$ messages for a, each carrying a $\mathsf{V3}_a$ certificate or a $\mathsf{SigT}^{(m)}$ certificate for a, and following these nested

certificates ends at a $\mathsf{V3}_a$ certificate. Each vote1 in this V_a certificate contains a lock for a from an earlier view: either an input justification for a, that is, a justification for a from layer $m-1$, or a V_a certificate of an earlier view of layer m. Following such locks back through earlier views ends at a justification for a from layer $m-1$. Since $m-1<k$, the induction hypothesis applies to this justification, so it contains a justification for a from every layer $j<m-1$. Hence X contains a justification for a from every layer $j<m$.

Now let X be a justification for a from layer $m<k$. Since $m<k$, X does not use line 56. If X is formed by a party q using line 59, then q's own allgood/equiv message in X contains q's justification for a from layer $m-1$, to which the induction hypothesis applies. Otherwise, X contains a value-$\mathsf{SigT}^{(m)}$ for a: in an allgood message for a if X is of type clean or adopt, and in the evidence used on lines 54–65 if X is of type tally. Following its nested certificates as in the previous paragraph, X contains a $\mathsf{V3}_a$ certificate of layer m, to which the previous paragraph applies.

Next, if there exists an allgood certificate from layer j for $b\neq a$, then by Observation 4.7 no allgood message from layer j is for a, so no justification for a from layer j is of type clean or adopt, as each of these contains an allgood message for a.

Finally, we prove the second part, the bound $|D_X|\geq T_m$, by induction on m. The adopt or tally justification contains an equiv message, and hence conflicting certificates σ,σ' of layer $m-1$. By Claim 4.4, $|D_X|\geq n-2f+|D_\sigma|+|D_{\sigma'}|\geq T_2$, which proves the bound for $m\leq 2$. For $m>2$, each of σ,σ' contains a message whose justification from layer $m-2$ is not of type clean, since a clean justification from layer $m-2$ contains an allgood certificate from that layer and X contains none; hence it is of type adopt or tally. By the induction hypothesis, $|D_\sigma|,|D_{\sigma'}|\geq T_{m-1}$, so $|D_X|\geq n-2f+2T_{m-1}=T_m$. □

Lemma 4.9. *Let σ be an* allgood *certificate for a, and let J be a valid* tally *justification for $b\neq a$ that does not use line 56, both from layer i. Let $G=\mathsf{signers}(\sigma)$, let O be the set* equivSenders *in J, and let τ_b be the $\mathsf{V3}_b$ certificate selected in J. Then $|G\cap(\mathsf{signers}(\tau_b)\cup O)|\geq(3n-5f)/2$, and σ and J together identify a set B with $G\cap(\mathsf{signers}(\tau_b)\cup O)\subseteq B\subseteq F$ such that*

$$|B|\geq\frac{3n-5f}{2}+|D_\sigma|+|D_{\tau_b}|.$$

Proof. The $n-f$ parties in O are outside the blame of J, which contains D_J. Thus $O\cap D_J=\emptyset$.

The certificate σ contains a $\mathsf{SigT}^{(i)}$ certificate with $n-f$ value-$\mathsf{SigT}^{(i)}$ messages for a. Since this certificate and O each have $n-f>n/2$ parties, some $z\in O$ signed this certificate. The equiv message of z in J contains z's $\mathsf{SigT}^{(i)}$, which, since z can use it only once, is z's value-$\mathsf{SigT}^{(i)}$ for a. Thus a is in the set $\mathcal{V}$ of J. Since J is for b and does not use line 56, $\mathcal{V}=\{a,b\}$, and J selects b on line 65 by comparing certificates for a and b.

Let τ_a be the $\mathsf{V3}_a$ certificate selected in J, $P_a=\mathsf{signers}(\tau_a)$, and $P_b=\mathsf{signers}(\tau_b)$. Every party in $P_a\cap P_b$ sent vote3 messages for both values, which J contains, so $P_a\cap P_b\subseteq D_J$, and $(P_a\cap P_b)\cap O=\emptyset$. The sets S_a and S_b computed in J therefore have $|S_a|=n-f-|P_a\cap P_b|-|P_a\cap O|$ and $|S_b|=n-f-|P_a\cap P_b|-|P_b\cap O|$. Since J selects b, $|S_b|\geq|S_a|$, that is, $|P_b\cap O|\leq|P_a\cap O|$. These two intersections are disjoint subsets of O, so $|P_b\cap O|\leq(n-f)/2$. Hence $|P_b\cup O|\geq n-f+(n-f)/2$, and $|G\cap(P_b\cup O)|\geq|G|+|P_b\cup O|-n\geq n-2f+(n-f)/2=(3n-5f)/2$.

Every party in $G\cap O$ sent both an allgood and an equiv message, contrary to Observation 4.6, and every party in $G\cap P_b$ sent an allgood message for a and a vote3 message for b, contrary to Claim 4.2; σ and J contain these messages. Hence σ and J together identify the set $B=D_J\cup D_\sigma\cup(G\cap(P_b\cup O))$ of corrupt parties.

To bound $|B|$, let $H = [n] \setminus (D_J \cup P_b \cup O)$. Then B contains $G \setminus H$, and $G \cap D_\sigma = \emptyset$, so $|B| \geq |D_\sigma| + n - f - |H|$. Since $O \cap D_J = \emptyset$, we have $|H| = n - |P_b \cup O| - |D_J \setminus P_b|$. Since τ_b is in J, $D_{\tau_b} \subseteq D_J$, and $D_{\tau_b} \cap P_b = \emptyset$. Consequently, $|H| \leq f - (n-f)/2 - |D_{\tau_b}|$, and $|B| \geq (3n - 5f)/2 + |D_\sigma| + |D_{\tau_b}|$. □

Lemma 4.10. *Let A contain an* allgood *certificate σ for a and B contain a* tally *justification J for $b \neq a$, both from layer j. Let τ_b be the* $\mathsf{V3}_b$ *certificate selected in J, O the set* equivSenders *in J, and $z \in \mathsf{signers}(\sigma) \cap (\mathsf{signers}(\tau_b) \cup O)$. If A and B each contain a message of z from a layer $i \geq j + 2$, then one of A and B, on its own, proves z corrupt.*

Proof. Each message of z from layer i contains z's justification for layer i, and thus z's $\mathsf{SigT}^{(i-1)}$, which signs z's justification for layer $i-1$. Since z can use its $\mathsf{SigT}^{(i-1)}$ only once, all of z's messages from layer i contain the same justification for layer $i-1$. Repeating the argument down to z's $\mathsf{SigT}^{(j+1)}$, they contain the same allgood/equiv message r of z from layer j. If r is an allgood message, it is for a by Observation 4.7, and together with z's vote3 or equiv message in J it proves z corrupt; B contains all of these. Otherwise, r is an equiv message, and together with z's allgood message in σ it proves z corrupt; A contains all of these. □

Lemma 4.11. *Conflicting certificates of layer $i \leq k$ identify a set $B \subseteq F$ with $|B| \geq \min\{(3n - 5f)/2, T_{i+1}\}$. If $7f < 3n + 4T_i$, then $3(f - |B|) < n - |B|$. Every correct party that outputs* adopt *or* tally *in layer i has such a set in* blame *at its output.*

Proof. Let the certificates be σ_a, σ_b, for values $a \neq b$. Suppose one of them contains an allgood certificate from a layer $j < i$. By Lemma 4.8, if this certificate is for b, then σ_a contains a tally justification for a from layer j, and if it is for a, then σ_b contains one for b; either way, this tally justification does not use line 56, since $j < k$. By Lemma 4.9, this allgood certificate and this tally justification, which σ_a and σ_b contain, identify at least $(3n - 5f)/2$ corrupt parties. Otherwise, $|D_{\sigma_a}|, |D_{\sigma_b}| \geq T_i$: trivially for $i = 1$, and for $i \geq 2$ by Lemma 4.8, since each certificate contains an adopt or tally justification from layer $i-1$ and no allgood certificate. By Claim 4.4, the certificates then identify at least $n - 2f + 2T_i = T_{i+1}$ corrupt parties.

Since $n > 2f$, $(3n - 5f)/2 > (3f - n)/2$, and if $7f < 3n + 4T_i$, then $T_{i+1} > (3f - n)/2$. Either bound gives $3(f - |B|) < n - |B|$. Both the adopt and the tally branch require a valid equiv message, which contains conflicting certificates of layer i, so the outputting party's blame contains the set that these certificates identify. □

Lemma 4.12. *If an* allgood *certificate for a from layer k exists, every valid justification from layer k that exists is for a.*

Proof. Let σ be the allgood certificate for a, and $K = \mathsf{signers}(\sigma) \setminus F$. Since $\mathsf{signers}(\sigma)$ lies outside $D_\sigma \subseteq F$, $|K| \geq n - 2f + |D_\sigma|$. Suppose there exists a valid justification J for $b \neq a$ from layer k. By Observation 4.7, J is of type tally. Let O be the set equivSenders in J; it lies outside the blame in J, which contains D_J, so $O \cap D_J = \emptyset$. Some $z \in O$ signed a $\mathsf{SigT}^{(k)}$ certificate in σ, since both have $n - f > n/2$ parties, so z's equiv message in J contains z's value-$\mathsf{SigT}^{(k)}$ for a, which is in σ. Following its nested certificates, J contains a $\mathsf{V3}_a$ certificate C contained in σ. The sets K, D_J, and O are pairwise disjoint: $K \cap D_J = \emptyset$ since $D_J \subseteq F$, $D_J \cap O = \emptyset$ as noted, and $K \cap O = \emptyset$ by Observation 4.6. Since $|O| = n - f$, $|K| + |D_J| \leq f$, and

$$n - 2f + |D_\sigma| + |D_J| \leq f. \tag{1}$$

Case 1: J contains an allgood *certificate σ' from layer $k-1$.* By line 56, σ' is for b. By Lemma 4.8, C contains a tally justification J' for a from layer $k-1$; let τ be its selected $\mathsf{V3}_a$ certificate. By the same argument for J' and σ', J' contains a $\mathsf{V3}_b$ certificate τ' contained in σ'. Since σ contains C and τ', by Lemma 4.8 it contains justifications for both values from every layer before $k-1$. Since J contains σ' and J', and J' is from layer $k-1$, Lemma 4.9 gives $|D_J| \geq (3n-5f)/2 + |D_{\sigma'}| + |D_\tau|$, where D_J contains the set $X = \mathsf{signers}(\sigma') \cap (\mathsf{signers}(\tau) \cup O')$ of size at least $(3n-5f)/2$, with O' the set equivSenders in J'.

- Suppose σ contains an allgood certificate σ_j from a layer $j \leq k-2$. Since σ contains justifications for both a and b from layer j, by Lemma 4.8 it contains a tally justification from layer j for the value other than that of σ_j, so $|D_\sigma| \geq (3n-5f)/2$ by Lemma 4.9. Since also $|D_J| \geq (3n-5f)/2$, (1) gives $n - 2f + (3n-5f) \leq f$, contradicting $n > 2f$.

- Suppose σ' contains an allgood certificate σ_j from a layer $j \leq k-3$. If σ_j is for a, then by Lemma 4.8 σ' contains a tally justification for b from layer j, so $|D_{\sigma'}| \geq (3n-5f)/2$ by Lemma 4.9, and $|D_J| \geq (3n-5f)/2 + |D_{\sigma'}| \geq 3n-5f$. If σ_j is for b, then by Lemma 4.8 τ contains a tally justification J_j for a from layer j; let τ_j be the $\mathsf{V3}_a$ certificate selected in J_j and O_j the set equivSenders in J_j. By Lemma 4.9, $X_j = \mathsf{signers}(\sigma_j) \cap (\mathsf{signers}(\tau_j) \cup O_j)$ has at least $(3n-5f)/2$ parties, and $X_j \subseteq D_J$ since J contains σ_j and J_j. Let $z \in X \cap X_j$. Then z has messages from layer $k-1$ in σ' and in τ or J', so by Lemma 4.10 for σ_j, J_j, and z, one of σ' and τ on its own proves z corrupt if $z \in \mathsf{signers}(\tau)$, and one of σ' and J' does otherwise. This contradicts $z \in X$, since a certificate's signers lie outside its corrupt set and O' lies outside $D_{J'}$. Hence $X \cap X_j = \emptyset$, and $|D_J| \geq |X| + |X_j| \geq 3n-5f$. In both cases, (1) gives $n - 2f + (3n-5f) \leq f$, contradicting $n > 2f$.

- Otherwise, σ contains no allgood certificate from a layer before $k-1$, and σ' none from a layer before $k-2$. Since σ contains J', $|D_\sigma| \geq T_k$ by Lemma 4.8. For $k \geq 3$, the justifications from layer $k-2$ in τ and τ' are in σ, which contains no allgood certificate from layer $k-2$, so they are of type adopt or tally, and Lemma 4.8 gives $|D_{\sigma'}|, |D_\tau| \geq T_{k-1}$; for $k = 2$ this is trivial. Then (1) gives $n - 2f + T_k + (3n-5f)/2 + 2T_{k-1} = (3n-5f)/2 + 2T_k \leq f$, contradicting $7f < 3n + 4T_k$.

Case 2: J contains no allgood *certificate from layer $k-1$.* Then J does not use line 56; let τ_b be its selected $\mathsf{V3}_b$ certificate. By Lemma 4.9, at least $(3n-5f)/2 + |D_\sigma| + |D_{\tau_b}|$ parties are corrupt, including the set $X = \mathsf{signers}(\sigma) \cap (\mathsf{signers}(\tau_b) \cup O)$ of size at least $(3n-5f)/2$.

- Suppose σ or τ_b contains an allgood certificate σ_j from a layer $j \leq k-2$. By Lemma 4.8, if σ_j is for b, then σ contains a tally justification J_j for a from layer j, and if σ_j is for a, then τ_b contains a tally justification J_j for b from layer j. Let τ_j be the V3 certificate selected in J_j for the value of J_j, and O_j the set equivSenders in J_j. By Lemma 4.9, $X_j = \mathsf{signers}(\sigma_j) \cap (\mathsf{signers}(\tau_j) \cup O_j)$ is a set of at least $(3n-5f)/2$ corrupt parties. If σ_j and J_j lie in the same one of σ, τ_b, then $|D_\sigma|$ or $|D_{\tau_b}|$ is at least $(3n-5f)/2$. Otherwise, let $z \in X \cap X_j$. Then z has messages from layer k in σ and in τ_b or J, so by Lemma 4.10 for σ_j, J_j, and z, one of σ and τ_b on its own proves z corrupt if $z \in \mathsf{signers}(\tau_b)$, and one of σ and J does otherwise. This contradicts $z \in X$, since a certificate's signers lie outside its corrupt set and O lies outside D_J. Hence $X \cap X_j = \emptyset$. Either way, at least $3n-5f$ parties are corrupt, which exceeds f since $n > 2f$.

- Otherwise, C and τ_b contain no allgood certificate from a layer before $k-1$, and, as they lie in J, none from layer $k-1$. Hence every justification from layer $k-1$ in them is of type adopt

or tally. By Lemma 4.8, $|D_\sigma| \geq |D_C| \geq T_k$ and $|D_{\tau_b}| \geq T_k$, trivially for $k = 1$. Hence at least $(3n - 5f)/2 + 2T_k$ parties are corrupt, which exceeds f since $7f < 3n + 4T_k$.

□

Lemma 4.13. *No two valid* V2 *certificates of layer* $k + 1$ *are for different values.*

Proof. Otherwise, by Lemma 4.5, the two certificates contain conflicting certificates σ, σ' of layer $k + 1$; if they are of the same view, σ and σ' are the two certificates themselves. Since validity is recursive, σ and σ' are valid, so $|D_\sigma|, |D_{\sigma'}| > (3f - n)/2$, and by Claim 4.4, they identify more than $n - 2f + (3f - n) = f$ corrupt parties, a contradiction. □

Lemma 4.14. *All* decided *messages multicast by correct parties are for the same value.*

Proof. A correct party multicasts $\langle \mathsf{decided}, a \rangle$ only upon a clean output for a in layer k or a valid $\mathsf{V2}_a$ certificate of layer $k+1$. Two clean outputs in layer k are for the same value by Observation 4.7. A clean output for a in layer k contains an allgood certificate for a. Let a correct party receive a valid $\mathsf{V2}_b$ certificate of layer $k + 1$. Following the locks of its vote1 messages back through earlier views reaches an input justification for b in layer $k + 1$, that is, a valid justification for b from layer k, which exists; hence $b = a$ by Lemma 4.12. Two valid V2 certificates of layer $k + 1$ are for the same value by Lemma 4.13. □

The following claim concludes the agreement proof.

Claim 4.15. *Algorithm 1 satisfies agreement.*

Proof. A correct party decides a only after receiving $f + 1$ $\langle \mathsf{decided}, a \rangle$ messages, one of which is from a correct party. By Lemma 4.14, no two correct parties decide different values. □

4.2.2 Termination

We now prove that all correct parties eventually exit each layer $i \leq k$. Fix such i, and assume that no correct party decides and that every correct party eventually enters layer i.

Observation 4.16. *Let p be a correct party that receives a valid message m at time t. Then eventually, all correct parties receive m.*

Proof. Correct parties forward every received valid message. The network guarantees delivery by time $\max\{\mathrm{GST}, t\} + \Delta$. □

Claim 4.17. *If a correct party receives conflicting certificates of layer i, then eventually, every correct party multicasts an* allgood *or* equiv *message in this layer.*

Proof. By Observation 4.16, all correct parties eventually receive the certificates. Every correct party that has not already sent one of the two messages then satisfies the condition on line 36, uses its $\mathsf{SigT}^{(i)}$ if needed, and multicasts an equiv message. □

Claim 4.18. *Suppose no correct party sends an* allgood *or* equiv *message. Every correct party eventually enters and exits every view. Exiting a view requires receiving either a V_a certificate or $n - f$* viewchange *messages of that view.*

Proof. We prove the first statement by induction on the view. All correct parties eventually enter view 0. Suppose all eventually enter view v. If any correct party receives a V_a certificate or $n-f$ viewchange messages of view v, it forwards them, and all correct parties eventually exit v. Otherwise, every correct party's timer expires, and every correct party multicasts viewchange for v. Thus all eventually exit v and enter $v+1$. The second statement follows from the two view-exit conditions. □

Observation 4.19. *Suppose no correct party sends an* allgood *or* equiv *message. If a correct party enters view v at a time $t >$ GST by which all correct parties have entered the layer, every correct party reaches view v or a later view by time $t + \Delta$.*

Proof. By Claim 4.18, the party received the messages for exiting each preceding view. It forwards them. By time $t + \Delta$, every correct party receives them and can exit each preceding view. □

Observation 4.20. *If a correct party receives a $\mathsf{SigT}^{(i)}$ certificate in layer i, then eventually, every correct party multicasts an* allgood *or* equiv *message in this layer.*

Proof. All correct parties eventually receive the certificate. Let its value be a, and consider a correct party that has not sent either message. If it has received conflicting certificates of layer i, Claim 4.17 applies. Otherwise, it sends an allgood message unless it previously cast a vote3 or used its $\mathsf{SigT}^{(i)}$ for some $b \neq a$.

In the latter case, that party holds a $\mathsf{V2}_b$ certificate, and the $\mathsf{SigT}^{(i)}$ certificate contains a $\mathsf{V2}_a$ certificate. By Lemma 4.5, these contain conflicting certificates, which that party receives, and Claim 4.17 concludes the proof. □

Lemma 4.21. *Eventually, every correct party multicasts an* allgood *or* equiv *message in layer i.*

Proof. If any correct party receives conflicting certificates of layer i or a $\mathsf{SigT}^{(i)}$ certificate, the proof follows from Claim 4.17 and Observation 4.20. Assume otherwise. No correct party sends an allgood or equiv message, and by Claim 4.18, all correct parties enter arbitrarily large views.

First, any two vote3 messages received by correct parties are for the same value. Otherwise, their V2 certificates contain conflicting certificates by Lemma 4.5, which some correct party receives by Observation 4.16, contrary to the assumption.

Next, let v be a sufficiently late view whose leader is correct and has its $\mathsf{SigT}^{(i)}$ unused when it enters v. We show that the leader then uses its $\mathsf{SigT}^{(i)}$ on a value-$\mathsf{SigT}^{(i)}$. Let t be the time the first correct party enters v; since v is late enough, t is after GST and after all correct parties enter the layer. By Observation 4.19, all correct parties reach v by time $t + \Delta$. Before time $t + 20\Delta$, no correct party exits v except upon a V certificate of v, since $n - f$ viewchange messages for v include one from a correct party, sent only when its timer expires.

Let $t_\ell \leq t + \Delta$ be the leader's entry time. By time $t + 2\Delta \leq t_\ell + 2\Delta$, the leader receives the suggest messages of all correct parties, and it then proposes a lock L for some value a whose view is at least that of each of their locks. A correct party changes its lock only upon a V certificate of its current view, and every valid V certificate in view v is for a, since each of its vote1 messages carries a propose of the correct leader, which proposes only a. If a correct party exits v before casting its own vote1, it exits upon a V_a certificate of view v and forwards it. Otherwise its lock is the one it suggested, so the condition on line 11 holds, and it casts vote1 for a. In either case, by time $t_\ell + 5\Delta$, all correct parties receive a V_a certificate and multicast vote2 for a. The certificates

of view v here and below can be formed from messages of correct parties alone; their corrupt sets lie in F, so they are valid.

By time $t_\ell + 6\Delta$, all correct parties receive a $\mathsf{V2}_a$ certificate. A correct party's earlier vote3 or value-$\mathsf{SigT}^{(i)}$ for $b \neq a$ contains a $\mathsf{V2}_b$ certificate, which together with this one contains conflicting certificates, by Lemma 4.5, contrary to the assumption. Thus no correct party's previous vote3 or value-$\mathsf{SigT}^{(i)}$ prevents it from casting vote3 for a. By time $t_\ell + 7\Delta$, the leader receives the vote3 messages for a of all correct parties and uses its $\mathsf{SigT}^{(i)}$ for a, unless it has already used it for a while processing an earlier view. Since $t_\ell + 7\Delta \leq t + 8\Delta < t + 20\Delta$, no correct party's timer for v expires during these steps.

By Claim 4.18, all correct parties enter arbitrarily large views. Since leaders rotate in round-robin order, every correct party whose $\mathsf{SigT}^{(i)}$ remains unused eventually leads such a view and uses it. All these uses are for the same value, since all vote3 messages received by correct parties are. Any correct party that used its $\mathsf{SigT}^{(i)}$ earlier did so on line 33, since it sent no allgood or equiv message, and hence also for that value. Eventually, therefore, all correct parties receive $n - f$ value-$\mathsf{SigT}^{(i)}$ messages for the same value, forming a $\mathsf{SigT}^{(i)}$ certificate. This contradicts the assumption and concludes the proof. □

Lemma 4.22. *Every correct party eventually exits layer i with a* clean*,* adopt*, or* tally *output and the corresponding justification.*

Proof. By Lemma 4.21, every correct party multicasts an allgood or equiv message. Since $\mathsf{blame}_p \subseteq F$ for every correct party p, eventually p receives at least $n - f$ such messages from parties outside blame_p, and returns. In the clean branch, these are all allgood messages for the same value, by Observation 4.7. In the adopt branch, J contains p's own allgood/equiv message and a message of the other type. Each branch supplies the evidence required in its justification. □

Lemma 4.23. *Let p be a correct party that outputs (type, a, J) in layer $i \leq k$. Then J is a valid input justification for a in layer $i + 1$, and, if $i < k$, p's $\mathsf{SigT}^{(i+1)}$ is unused.*

Proof. By Observation 4.6, J contains p's own allgood/equiv message, which carries p's $\mathsf{SigT}^{(i)}$ and J_p. The return block adds the rest of the evidence that a justification of type type requires, so J is a valid input justification for a in layer $i + 1$. Each layer $i' \leq k$ uses only $\mathsf{SigT}^{(i')}$, so, if $i < k$, p's $\mathsf{SigT}^{(i+1)}$ is unused. □

In layer 1, each correct party starts with $J_p = \bot$, a valid input justification for either value, and its $\mathsf{SigT}^{(1)}$ unused. Together with Lemma 4.23, this shows that the input conditions hold in every layer.

Lemma 4.24. *Suppose no correct party decides. Then every correct party receives a valid* V2 *certificate of layer $k + 1$.*

Proof. No correct party terminates, so by Lemmas 4.22 and 4.23, every correct party completes layers $1, \ldots, k$ and enters layer $k + 1$. If every correct party outputs clean in layer k, then the $n - f \geq f + 1$ correct parties multicast $\langle \mathsf{decided}, a \rangle$ for the same a, by Observation 4.7, and every correct party decides. Hence some correct party p outputs adopt or tally in layer k. By Lemma 4.11 for $i = k$, since $7f < 3n + 4T_k$, its blame at output contains a set B with $3(f - |B|) < n - |B|$, that is, $|B| > (3f - n)/2$. This blame is contained in J_p, so every message p sends in layer $k + 1$ contains proofs for B.

Suppose no correct party receives a valid V2 certificate of layer $k+1$. Then no correct party returns from this layer, and since the proof of Claim 4.18 uses only that no correct party returns, every correct party enters every view. Since leaders rotate in round-robin order, p leads arbitrarily late views. The view argument in the proof of Lemma 4.21, up to the $\mathsf{V2}_a$ certificates, applied to a sufficiently late view v led by p, shows that every correct party receives a valid V2 certificate of v, a contradiction. Here every vote1 of view v carries p's propose, which carries J_p, so the corrupt set of every certificate of view v contains B, and $|D_\sigma| > (3f-n)/2$. Thus some correct party receives a valid V2 certificate of layer $k+1$. It forwards the vote2 messages that this certificate consists of, and since the validity of a certificate depends only on its contents, every correct party forms the same valid certificate. □

The following claim concludes the termination proof.

Claim 4.25. *Algorithm 1 satisfies termination.*

Proof. A correct party that decides has forwarded $f+1$ $\langle \mathsf{decided}, a\rangle$ messages, so every correct party decides. Suppose no correct party decides. By Lemma 4.24, every correct party receives a valid V2 certificate of layer $k+1$ and multicasts a decided message. By Lemma 4.14, these $n-f \geq f+1$ messages are for the same value, so every correct party decides, a contradiction. □

4.2.3 Validity

Claim 4.26. *Algorithm 1 satisfies validity.*

Proof. Suppose all parties are correct with input a. Since $F = \emptyset$, Claim 4.4 implies that no conflicting certificates exist, so there are no equiv messages. By induction on layers and views, every lock, proposal, vote, value-SigT, and certificate is for a, every output is clean for a, and every party enters the next layer with input a. Hence every $\langle \mathsf{decided}, b\rangle$ message has $b = a$, and no correct party decides a value other than a. □

Proof of Theorem 4.1. Algorithm 1 is deterministic. By Claims 4.15, 4.25 and 4.26, it satisfies agreement, termination, and validity against every f-bounded adversary. Hence it is an f-secure BA protocol, which proves Theorem 4.1. □

5 Lower Bound

In this section, we prove the following two theorems.

Theorem 5.1. *Let n, f be positive integers such that $\frac{3n}{7} \leq f < n$. There is no deterministic f-secure protocol for BA in the partially synchronous model with a PKI and a 1-SigT setup.*

Theorem 5.2. *Let $k \geq 2$ be an integer and let n, f be positive integers such that $\frac{2(4^k-1)}{4^{k+1}-1}n \leq f < n$. There is no deterministic f-secure protocol for BA in the partially synchronous model with a PKI and a k-SigT setup.*

Intuition for Theorem 5.1. The starting point is the obstacle that SigT introduces to a DLS-style attack [DLS88]. Ordinarily, a corrupt party can behave toward one set of correct parties as in an execution leading to 1, and toward another set as in an execution leading to 0. With SigT, however, these two behaviors may be incompatible: they may require the same party to use its SigT on two different messages. This suggests a different approach. What if we first make the parties that will eventually equivocate use their SigT in exactly the same way in both executions? Their subsequent equivocation would no longer require a second use. The question is whether the adversary can arrange this while leaving enough freedom to obtain different decisions. As we will see, it can—but doing so requires an additional set of $n - 2f$ corrupt parties. This is also the number of parties whose equivocation the $k = 1$ protocol seeks to establish before removing them and proceeding.

The prefix π_0. To see how this can happen, consider the boundary case $f = 3n/7$, and write $q = n - 2f = n/7$. Take a set S of $4q = n - f$ parties, give all of them input 1, and let them communicate with GST = 0, while the remaining parties are corrupt and silent. Eventually, the parties in S must decide. Moreover, their decision must be 1: as far as they can tell, the silent parties could also be correct with input 1, with their messages merely delayed until after the decision.

Already, this execution gives us two possibilities. If a party decides before f parties have used their SigT, then the protocol has left enough SigT resources unused for a simpler DLS-style attack, as we prove in Lemma 5.3. The interesting case, therefore, is when the protocol does use sufficiently many SigTs. Let B consist of the first $2q$ parties to use theirs, and stop immediately after these uses, at a prefix denoted π_0. There are still $2q$ parties in S whose SigTs are unused. Divide them into sets A and R, each of size q, and divide the parties outside S into sets K and C, of sizes q and $2q$, respectively.

Executions E_1 and G. What freedom does this leave the adversary? First, the parties in R need not continue participating in order for A to decide 1. Consider execution E_1, in which R reproduces the preceding behavior toward A, B, but stops at π_0. At that point, allow K, whose communication has so far been delayed and whose input is 1, to communicate with A, B. These three sets contain $n - f$ parties, so they must eventually decide even if R, C remain silent. Their deciding histories are again consistent with an all-correct execution with input 1. Thus a party in A decides 1, despite R's departure.

But the unused SigTs in R also give the adversary another possibility. Instead of continuing toward A, the parties in R can behave toward K, C as if they were correct with input 0. This time, give K, C input 0, and delay communication involving A, B after π_0. The parties in R can reproduce the required input-0 behavior, including its SigT uses, precisely because they did not use their SigTs while communicating with A, B. The parties in K, C then see an execution consistent with R, K, C being correct with input 0, while everybody else is corrupt and silent. Consequently, a party in C eventually decides 0. Denote this execution by G; only R is corrupt in G.

Two features of this construction are worth emphasizing. The continuation leading to 0 does not need B, whose SigTs have already been used. And the parties in R can support that continuation because their earlier input-1 behavior required no SigT use. These are exactly the two kinds of freedom we were looking for.

There is a catch, however. We have not yet obtained a contradiction: K has different inputs in the two executions and communicates with different parties. We cannot simply run both continu-

ations together and declare that A decides 1 while C decides 0. Instead, we must understand how the possibility of deciding 1 gives way to the decision of 0 as G proceeds.

Checking the decision with A. For this purpose, let G unfold one local step or message delivery at a time. At each resulting prefix π_i, the adversary asks whether it can still make A decide 1 by allowing A, B, K to communicate from that point. To examine this possibility, consider two separate executions. In the first, G_i^A, the parties in A, B, K continue communicating, while the remaining parties are corrupt and stop. In the second, G_i^C, the parties in B, C continue instead, while the remaining parties are corrupt and stop. In both G_i^A and G_i^C, the correct parties continue from their respective local states at π_i, consistently with all the information available to them at that point. Each execution has $n - f$ correct parties and must therefore terminate.

Why consider these particular two executions? Because, unlike the two executions we started with, these two executions can be realized simultaneously in one execution via equivocation by B. Corrupt $B \cup R$, and have B behave toward A, K as in G_i^A, and toward C as in G_i^C. All parties in B used their SigT before these executions differed, so reproducing both subsequent behaviors requires no second SigT. With communication between A, K and C delayed, we obtain the execution H_i shown in the figure. Agreement in H_i forces the decision in G_i^A to equal the decision in G_i^C. Call this value d_i.

The first change in decision. Now consider what this value must be at the beginning. At π_0, the parties in C have heard nothing from K. Choose G_0^C so that they still receive no message from K before deciding. From their point of view, K could therefore have input 1 and be helping A complete execution E_1. Indeed, in that separate execution, B can simultaneously reproduce the E_1 behavior toward A, K and the G_0^C behavior toward C. Since A decides 1, agreement forces C to decide 1 as well. Hence $d_0 = 1$.

On the other hand, this cannot remain true throughout G. Eventually, C has already decided 0 through its communication with R, K. At that point, its decision is retained in G_N^C, and therefore $d_N = 0$. Somewhere between these two endpoints, there must be a first step at which the decision changes from 1 to 0.

It is precisely here that the earlier SigT uses become decisive. The two consecutive prefixes differ at only one party. Depending on which party performs the step, we combine the deciding history from before the step on one side with the deciding history from after the step on the other. The direction is chosen so that no correct party must reproduce two different histories. If the step occurs within B itself, its parties can reproduce both behaviors because their SigTs have already been used. Thus, in either case, the corrupt parties in B can present the appropriate behavior to each side, producing an execution in which correct parties in A and C decide differently.

The set sizes and the threshold $3n/7$. It remains to explain where the set sizes, and hence the threshold $3n/7$, come from. Rather than fixing them in advance, write $r = |R|$ and $b = |B|$, and ask what the executions require. First, the initial execution on $S = A \cup B \cup R$ must contain at least $n - f$ correct parties for termination to apply. The same is true of the input-0 execution on $R \cup K \cup C$. These two sets together contain all n parties and intersect exactly in R. Consequently, $n + r = |A \cup B \cup R| + |R \cup K \cup C| \geq 2(n - f)$, which forces $r \geq n - 2f$. Thus the need for both executions to terminate already imposes a minimum size on R.

Next, the executions G_i^A and G_i^C must also each contain at least $n - f$ correct parties. Their correct sets intersect exactly in B, and their union contains every party except those in R. It follows that $n - r + b = |A \cup B \cup K| + |B \cup C| \geq 2(n - f)$, and hence $b \geq n - 2f + r$. In particular, making R larger also forces us to make B larger: both executions must still terminate without relying on R.

Finally, to realize these two executions simultaneously, the adversary corrupts $B \cup R$. This must fit within its budget of f corrupt parties. Combining the three requirements gives $f \geq b + r \geq (n - 2f) + 2r \geq 3(n - 2f)$. Rearranging yields $f \geq 3n/7$. At equality, the requirements force $r = n - 2f$ and $b = 2(n - 2f)$.

5.1 Proof of Theorem 5.1

For both theorems, we may assume $f < n/2$; otherwise, the results follow from folklore indistinguishability arguments.

We first prove a preliminary lemma showing that too few SigT uses before a decision already allow a DLS-style attack.

Lemma 5.3. *Let $n/3 \leq f < n/2$, and let Π be a deterministic f-secure BA protocol in the partially synchronous model with a PKI and a 1-SigT setup. Consider an execution in which a set S of $n - f$ parties is correct, all parties in S have the same input, all remaining parties are corrupt and silent, and $\mathrm{GST} = 0$. Before any party in S decides, at least f parties in S must have used their SigT.*

Proof. Suppose, toward a contradiction, that the claim is false. By symmetry, assume that the common input is 1. We describe the following executions of Π.

Execution F_1. The parties in S are correct with input 1, the remaining parties are corrupt and silent, and $\mathrm{GST} = 0$. A party $p \in S$ decides at time t_1, by which fewer than f parties in S have used their SigT. Write $q = n - 2f$. At least $q + 1$ parties in S have not used their SigT by t_1, so choose $R \subseteq S \setminus \{p\}$ of size q among these parties. Let $A = S \setminus R$ and $C = [n] \setminus S$. Thus $p \in A$, $|A| = |C| = f$, and $|R| = q \leq f$.

Execution F_0. The parties in $R \cup C$ are correct with input 0, the parties in A are corrupt and silent, and $\mathrm{GST} = 0$. By termination, a party $c \in C$ eventually decides; denote its decision time by t_0.

Execution F. Only the parties in R are corrupt. The parties in A are correct with input 1, and the parties in C are correct with input 0. The parties in R behave toward A as in F_1 through time t_1, and toward C as in F_0 through time t_0. All messages between A and C are delayed beyond $\max\{t_0, t_1\}$, choosing a sufficiently large GST. No party in R uses its SigT in F_1 through t_1, so it can use its SigT as required by F_0 toward C.

Indistinguishability. The parties in $A \cup R$ cannot distinguish F_1 through t_1 from an execution in which all parties are correct with input 1, choosing a sufficiently large GST to delay messages from C beyond t_1. Thus, by validity, party p decides 1 in F_1. Similarly, the parties in $R \cup C$ cannot distinguish F_0 through t_0 from an execution in which all parties are correct with input 0, choosing

a sufficiently large GST to delay messages from A beyond t_0. Thus, by validity, party c decides 0 in F_0.

The parties in A cannot distinguish F from F_1 through t_1, because they receive the same messages from the same parties with the same timings. Thus p decides 1 in F. The parties in C cannot distinguish F from F_0 through t_0, because they receive the same messages from the same parties with the same timings. Thus c decides 0 in F. Both are correct in F, contradicting agreement. □

Proof of Theorem 5.1. We next formally define the executions used in the proof of Theorem 5.1. See Figures 2 and 3 for a visual description of the executions and the chain of indistinguishabilities establishing the lower bound. Let $3n/7 \leq f < n/2$, and suppose toward a contradiction that Π is a deterministic f-secure BA protocol as in the theorem statement. Write $q = n - 2f > 0$; then $3q \leq f$.

Execution steps and prefixes. For the analysis, we use a global time that is not available to the parties. We describe an execution as a sequence of steps, each involving one party: a message delivery or a local computation, including a timeout or a use of SigT. A prefix consists of the first finitely many steps. We say that a prefix π' extends a prefix π if π is a prefix of π'. The arbitrary clock skew and drift permitted before GST allow us to schedule these steps while preserving each party's observed clock readings.

The sets of parties. Choose a set S of $n-f$ parties. Consider an execution in which the parties in S are correct with input 1, the remaining parties are corrupt and silent, and GST $= 0$. Since Π is f-secure, Lemma 5.3 implies that at least f parties in S use their SigT before any party decides. Let B be the first $2q$ parties to do so. Choose $R \subseteq S \setminus B$ of size q, and let $A = S \setminus (B \cup R)$. Partition $[n] \setminus S$ into sets K, C of sizes $q, f-q$, respectively. Thus $|A| = f - 2q$, $|B| = 2q$, $|R| = |K| = q$, and $|C| = f - q$. Fix parties $a \in A$ and $c \in C$.

The prefix π_0. Consider an execution in which only R is corrupt, the parties in A, B are correct with input 1, and the parties in K, C are correct with input 0. The parties in R behave toward A, B as in the input-1 execution above, and messages within $A \cup B \cup R$ are delivered as in that execution, until all parties in B have used their SigT. Denote by π_0 the prefix ending immediately after these uses. All messages with a sender or recipient in $K \cup C$ are delayed beyond π_0, including messages between these parties. No party in $A \cup R$ has used its SigT in π_0.

Execution E_0. In execution E_0, the parties in R, K, C are correct with input 0, and the parties in A, B are corrupt and silent. Up to the end of π_0, all messages between correct parties are delayed. Set GST at the end of this prefix and let the execution conclude.

Execution E_1. In execution E_1, the parties in A, B, K are correct with input 1, and the parties in R, C are corrupt. The parties in C are silent. Up to the end of π_0, R behaves toward A, B as in π_0, and messages within $A \cup B \cup R$ are delivered as in that prefix. All messages with a sender or recipient in K are delayed beyond this prefix. Set GST at the end of this prefix and let the execution conclude with R silent.

Execution G. In execution G, only R is corrupt. The parties in A, B are correct with input 1, and the parties in K, C are correct with input 0. The execution begins with π_0. From that prefix onwards, R stops communicating with A, B and behaves toward K, C as in E_0. This is well-defined because no party in R has used its SigT in π_0. Messages to K, C are delivered as in E_0. All other messages are delayed until after c decides, choosing a sufficiently large GST.

The prefixes π_i. Starting from π_0, follow the steps of G in order until party c decides. Let N denote the number of these steps, with $N = 0$ if c has already decided in π_0. For each $0 \le i < N$, define π_{i+1} by appending the next step of G to π_i, and denote by p_i the party taking that step. Thus π_i consists of π_0 followed by the first i subsequent steps of G, and π_N includes c's decision. Between π_i and π_{i+1}, only p_i takes a step; every other party has received the same messages and taken the same local steps in both prefixes. For $0 \le i \le N$, write G_i for G through π_i.

Execution G_i^A. For $0 \le i \le N$, the parties in A, B, K are correct, with input 1 in A, B and input 0 in K, while R, C are corrupt. Through π_i, messages to A, B, K are delivered as in G. The parties in R, C reproduce exactly their messages delivered to A, B, K in that prefix, and send no other messages. Set GST at the end of π_i and let the execution conclude. Denote party a's decision by d_i^A.

Execution G_i^C. For $0 \le i \le N$, the parties in B, C are correct, with input 1 in B and input 0 in C, while A, R, K are corrupt. Through π_i, messages to B, C are delivered as in G. The parties in A, R, K reproduce exactly their messages delivered to B, C in that prefix, and send no other messages. Set GST at the end of π_i and let the execution conclude. Denote party c's decision by d_i^C. In particular, K sends no messages in G_0^C.

Execution H_i. For $0 \le i \le N$, the parties in B, R are corrupt, and the parties in A, K, C are correct, with input 1 in A and input 0 in K, C. The execution begins with π_i. From that prefix onwards, R is silent, and B behaves toward A, K as in G_i^A and toward C as in G_i^C. This is well-defined because all parties in B have already used their SigT in π_0. Messages within $A \cup K$ are delivered as in G_i^A, and messages within C are delivered as in G_i^C. Every message not prescribed above, whether sent before or after the prefix, is delayed until after a, c decide, choosing a sufficiently large GST.

Execution $\widetilde{G}_0^C$. The parties in B, C are correct, with input 1 in B and input 0 in C, while A, R, K are corrupt. The parties in A, R behave toward B, C as in G_0^C, and messages to B, C are delivered as in that execution until c decides. Up to the end of π_0, the parties in K reproduce their isolated input-1 behavior from E_1, but send no messages to B, C. Beyond this prefix, A, R, K are silent.

Execution $\widetilde{H}_0$. The parties in B, R are corrupt, and the parties in A, K, C are correct, with input 1 in A, K and input 0 in C. Up to the end of π_0, B, R behave toward A, K as in E_1, and messages are delivered as in E_1. Beyond this prefix, R is silent, and B behaves toward A, K as in E_1 and toward C as in $\widetilde{G}_0^C$. Messages within $A \cup K$ are delivered as in E_1, and messages within C are delivered as in $\widetilde{G}_0^C$. All other messages are delayed until after a, c decide, choosing a sufficiently large GST.

Execution $H_{i,i+1}$. For $0 \leq i < N$ with $p_i \notin C$, the parties in B, R are corrupt, and the parties in A, K, C are correct, with input 1 in A and input 0 in K, C. The execution begins with π_{i+1}. From that prefix onwards, R is silent, and B behaves toward A, K as in G^A_{i+1} and toward C as in G^C_i. Messages within $A \cup K$ are delivered as in G^A_{i+1}, and messages within C are delivered as in G^C_i. Every message not prescribed above, whether sent before or after the prefix, is delayed until after a, c decide, choosing a sufficiently large GST.

Execution $H_{i+1,i}$. For $0 \leq i < N$ with $p_i \in C$, the parties in B, R are corrupt, and the parties in A, K, C are correct, with input 1 in A and input 0 in K, C. The execution begins with π_{i+1}. From that prefix onwards, R is silent, and B behaves toward A, K as in G^A_i and toward C as in G^C_{i+1}. Messages within $A \cup K$ are delivered as in G^A_i, and messages within C are delivered as in G^C_{i+1}. Every message not prescribed above, whether sent before or after the prefix, is delayed until after a, c decide, choosing a sufficiently large GST.

Indistinguishability Arguments. We now turn to the indistinguishability arguments between the executions constructed above, starting with E_1 and ending with E_0. For each pair, we identify a set of parties that are correct in both executions and cannot distinguish between them up to the decision under consideration. This decision must therefore have the same value in both executions. The comparisons follow the structure depicted at the bottom of Figure 3.

By termination, party a eventually decides in E_1 and each G^A_i, and party c eventually decides in E_0 and each G^C_i.

E_1 and $\widetilde{H}_0$. The parties in A cannot distinguish E_1 from $\widetilde{H}_0$ up to a's decision, because they receive the same messages from the same parties with the same timings. Moreover, the parties in A, B, K cannot distinguish E_1 through that decision from an execution in which all parties are correct with input 1, choosing a sufficiently large GST to delay any additional messages from R, C until after a decides. Thus, by validity, party a decides 1 in E_1 and in $\widetilde{H}_0$.

$\widetilde{H}_0$ and $\widetilde{G}^C_0$. The parties in C cannot distinguish $\widetilde{H}_0$ from $\widetilde{G}^C_0$ up to c's decision, because they receive the same messages from the same parties with the same timings. Since a, c are correct in $\widetilde{H}_0$, agreement implies that c decides 1 in both executions.

$\widetilde{G}^C_0$ and G^C_0. The parties in C cannot distinguish $\widetilde{G}^C_0$ from G^C_0 up to c's decision, because they receive the same messages from the same parties with the same timings. In particular, K sends no messages to B, C in either execution, so its different behavior is not observed by these parties. Hence $d^C_0 = 1$.

G^A_i and H_i. For every $0 \leq i \leq N$, the parties in A cannot distinguish G^A_i from H_i up to a's decision, because they receive the same messages from the same parties with the same timings. After π_i, B behaves toward A, K as in G^A_i, R is silent, and messages from C are delayed. Thus party a decides d^A_i in H_i.

H_i and G^C_i. For every $0 \leq i \leq N$, the parties in C cannot distinguish H_i from G^C_i up to c's decision, because they receive the same messages from the same parties with the same timings.

After π_i, B behaves toward C as in G_i^C, R is silent, and messages from A, K are delayed. Thus party c decides d_i^C in H_i. Since a, c are correct in H_i, agreement gives $d_i^A = d_i^C$; denote this value by d_i. In particular, $d_0 = 1$.

G_N^C and E_0. The parties in C cannot distinguish G_N^C from E_0 up to c's decision, because they receive the same messages from the same parties with the same timings: G_N^C includes π_N, in which c has already decided after receiving the messages prescribed by E_0. The parties in R, K, C cannot distinguish E_0 through this decision from an execution in which all parties are correct with input 0, choosing a sufficiently large GST to delay messages from A, B until after c decides. Thus, by validity, party c decides 0 in E_0, and hence $d_N = 0$.

We next establish the two comparisons between consecutive prefixes shown at the bottom of Figure 3.

The comparisons through $H_{i,i+1}$. Fix $0 \le i < N$ with $p_i \notin C$. No party in C takes a step between π_i and π_{i+1}. The parties in C cannot distinguish G_i^C from $H_{i,i+1}$ up to c's decision, because B reproduces toward C its messages from G_i^C and all other messages to C not prescribed by that execution are delayed; they therefore receive the same messages from the same parties with the same timings. The parties in A cannot distinguish $H_{i,i+1}$ from G_{i+1}^A up to a's decision, because they receive the same messages from the same parties with the same timings, including the messages prescribed after π_{i+1}. Thus c decides d_i^C and a decides d_{i+1}^A in $H_{i,i+1}$. Agreement gives $d_i = d_{i+1}$.

The comparisons through $H_{i+1,i}$. Fix $0 \le i < N$ with $p_i \in C$. No party in $A \cup K$ takes a step between π_i and π_{i+1}. The parties in A cannot distinguish G_i^A from $H_{i+1,i}$ up to a's decision, because B reproduces toward A, K its messages from G_i^A and all other messages to $A \cup K$ not prescribed by that execution are delayed; they therefore receive the same messages from the same parties with the same timings. The parties in C cannot distinguish $H_{i+1,i}$ from G_{i+1}^C up to c's decision, because they receive the same messages from the same parties with the same timings, including the messages prescribed after π_{i+1}. Thus a decides d_i^A and c decides d_{i+1}^C in $H_{i+1,i}$. Agreement again gives $d_i = d_{i+1}$.

In both cases, B can reproduce the two required behaviors because all its parties used their SigT in the common prefix π_0. The two cases cover every step, so $d_0 = d_N$, contradicting $d_0 = 1$ and $d_N = 0$. This concludes the proof of Theorem 5.1. □

In fact, the chain of executions constructed in the proof of Theorem 5.1 proves the following stronger corollary, which we use in the proof of Theorem 5.2.

Corollary 5.4. *Let Π be a deterministic f-secure* BA *protocol with a PKI and a k-*SigT *setup, and write $q = n - 2f > 0$. Consider an execution prefix π in which only a set F of f' parties is corrupt. Suppose that every party in a set $B \subseteq [n] \setminus F$ of size $q + f'$ has used all its* SigT*s. If $|F \cup B| \le f$, then there do not exist two executions extending π, each with only F corrupt, in which correct parties decide different values.*

Proof. Suppose such executions E_0, E_1 exist, with correct decisions $0, 1$, respectively. Partition $[n] \setminus (F \cup B)$ into sets A, C, each of size $f - f'$. Both $A \cup B$ and $B \cup C$ have size $n - f$.

At each prefix ρ extending π along either execution, consider executions G_ρ^A, G_ρ^C whose correct sets are $A \cup B, B \cup C$, respectively. The remaining parties are corrupt, reproduce their messages

delivered to the correct parties through ρ, and send no other messages. The correct parties proceed from their states at ρ, with GST at any finite time at or after that prefix. By termination, parties in A, C eventually decide in the respective executions.

Apply the comparisons through H_i above, using F, A in place of $R, A \cup K$, respectively. The combined execution has only $F \cup B$ corrupt, and B can reproduce both behaviors because it has no unused SigTs. The parties in A and C receive the same messages from the same parties with the same timings as in their respective executions. Agreement therefore gives a decision value $d(\rho)$, independent of the choices of G^A_ρ, G^C_ρ.

For consecutive prefixes ρ, ρ^+, the comparisons through $H_{i,i+1}$ and $H_{i+1,i}$ apply with the same substitutions. If the step occurs outside C, combine $G^A_{\rho^+}$ with G^C_ρ; otherwise, combine G^A_ρ with $G^C_{\rho^+}$. The parties in C in the first case, and in A in the second, have not taken that step. As above, agreement gives $d(\rho) = d(\rho^+)$.

By agreement and termination, each E_v has a finite prefix η_v at which all parties outside F have decided v. Hence $d(\eta_v) = v$, whereas the consecutive-prefix comparisons give $d(\eta_0) = d(\pi) = d(\eta_1)$, a contradiction. □

5.2 Proof of Theorem 5.2

We now prove Theorem 5.2. The following lemma constructs the prefixes and executions used in the induction.

Lemma 5.5. *Let Π be a deterministic f-secure BA protocol with a PKI and a k-SigT setup, and write $q = n - 2f > 0$. Let $1 \le t \le k$ and $b \ge 1$ be integers. Consider an execution prefix π in which only a set F of f' parties is corrupt, and put $s = q + f'$. Suppose that every party in a set $B \subseteq [n] \setminus F$ of size $(4b+1)s \le f$ has used at least $k - t$ signature tokens, and that two executions extending π, each with only F corrupt, have different correct decisions.*

Then there exist sets $I \subseteq B$ of size s and $B' \subseteq B \setminus I$ of size $2bs$, and a prefix σ extending π with only $F \cup I$ corrupt, such that every party in B' has used at least $k - t + 1$ signature tokens. Moreover, two executions extending σ, each with only $F \cup I$ corrupt, have different decision values.

Proof. Let E_0, E_1 be the two executions extending π, with correct decisions $0, 1$, respectively. We define the following executions and prefixes.

The sets of parties and (ρ, S)-executions. Put $r = n - f - (2b+1)s \ge 2bs + q > 0$. There are $2r$ parties outside $F \cup B$. We consider sets $S \subseteq [n] \setminus F$ with $|S \cap B| = (2b+1)s$ and $|S \setminus B| = r$, so $|S| = n - f$. For such a set and a prefix ρ extending π with only F corrupt, a (ρ, S)-execution has S correct; the remaining parties are corrupt, reproduce their messages delivered to S through ρ, and send no others. The parties in S proceed from their states at ρ, with GST at any finite time at or after that prefix.

The prefixes π_i. Choose a set $S_0 \subseteq [n] \setminus F$ with $|S_0 \cap B| = (2b+1)s$ and $|S_0 \setminus B| = r$, and fix a (π, S_0)-execution. By termination and agreement, its correct parties eventually decide a value v. Starting from $\pi_0 = \pi$, follow E_{1-v} one step at a time until all parties outside F decide, obtaining prefixes $\pi_0, \dots, \pi_N$. At π_0, the chosen execution decides v; at π_N, every (π_N, S)-execution for a set S of the stated sizes decides $1 - v$. Let $j < N$ be the last index for which some (π_j, S)-execution

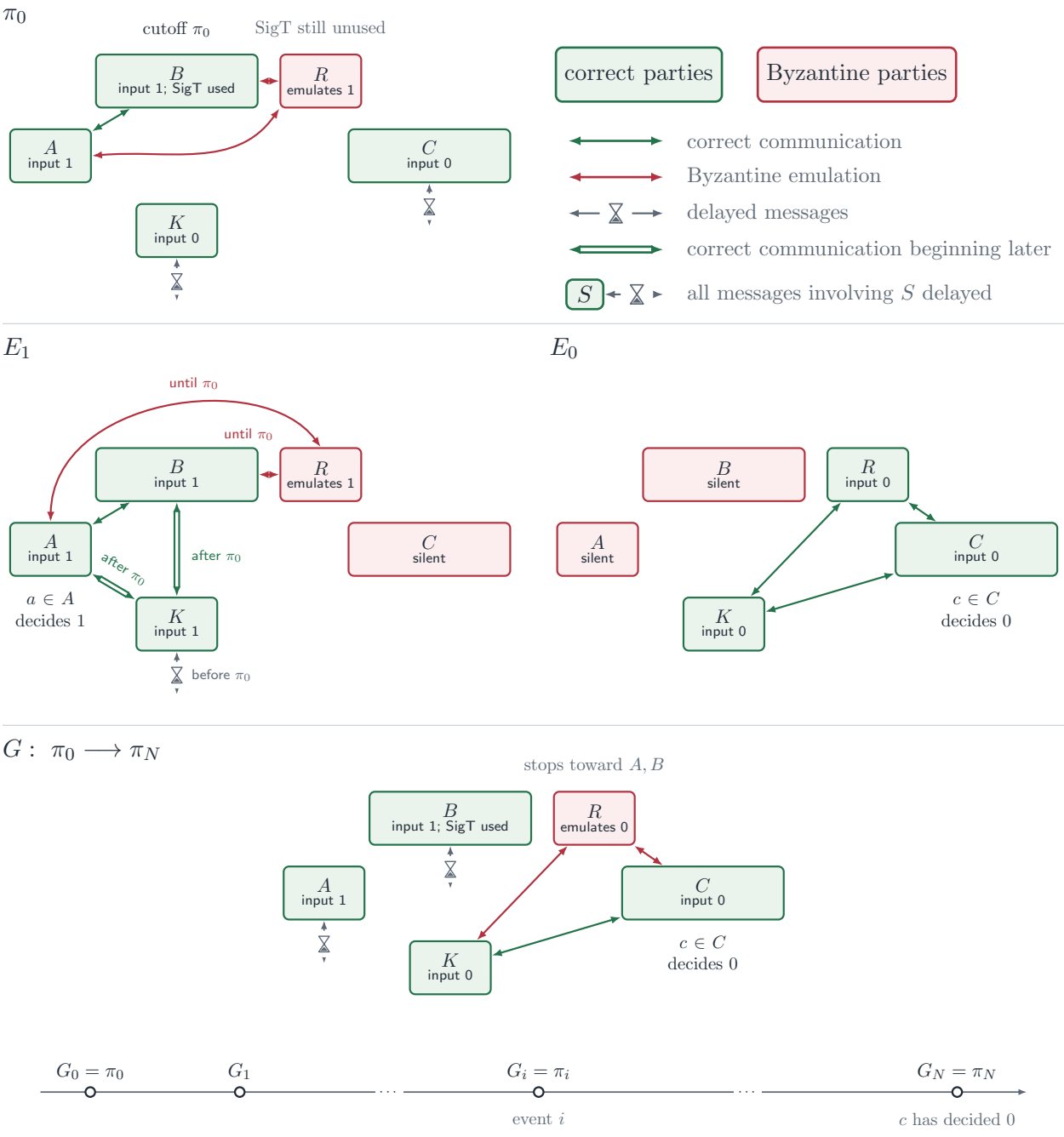

Figure 2: Executions E_1, E_0, and G. Here $n = 7t$, $f = 3t$, $|B| = |C| = 2t$, and $|A| = |K| = |R| = t$. Execution G reaches π_0 when all parties in B have used their SigT, and then continues one event at a time. In E_1, a party in A decides 1; in E_0, a party in C decides 0. Green and red boxes denote sets of correct and corrupt parties, respectively. Inputs and sets of corrupt parties are fixed within each execution. Hourglasses indicate finite message delays, not crashes.

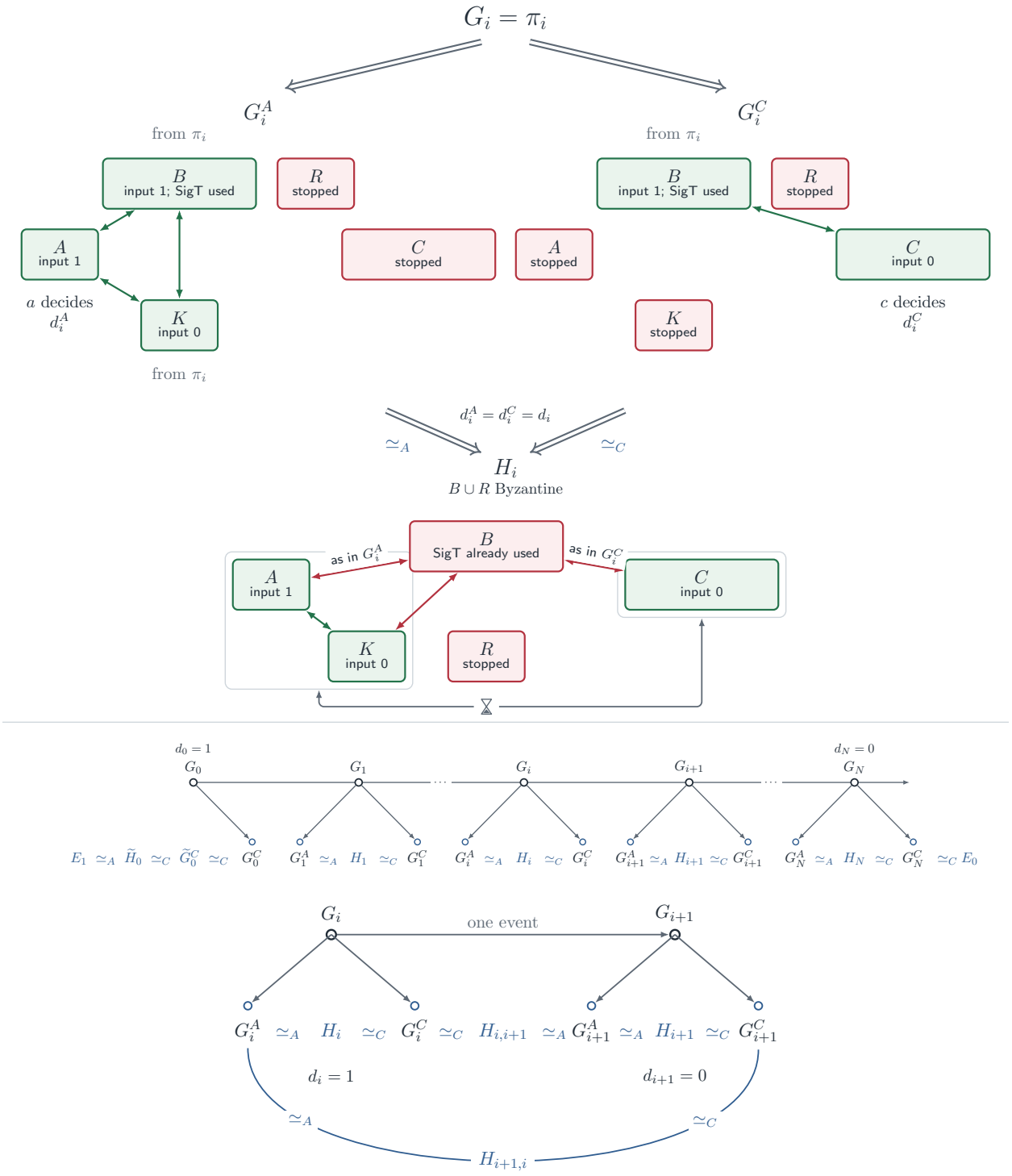

Figure 3: Executions and indistinguishability arguments. Double-line arrows indicate separate executions. Parties in S cannot distinguish executions related by $\simeq_S$ up to the indicated decisions. Agreement in H_i gives $d_i^A = d_i^C = d_i$. In executions marked with a tilde, parties in K input 1. Depending on the event, we use $H_{i,i+1}$ or $H_{i+1,i}$.

decides v. Write $\rho = \pi_j$, $\rho^+ = \pi_{j+1}$, and denote by p the party taking the step between ρ and ρ^+. Choose a set S of the stated sizes and a (ρ, S)-execution G^S deciding v. Every (ρ^+, T)-execution for such a set T decides $1 - v$. Moreover, $p \in S$: otherwise, the parties in S could receive the same messages with the same timings from either ρ or ρ^+ and still decide v.

Execution G. Only F is corrupt. The execution extends ρ, after which F is silent and messages to S are delivered as in G^S. Put $O = [n] \setminus (F \cup S)$. All messages with a sender or recipient in O are delayed until all parties in S decide, choosing a sufficiently large GST.

The prefix σ. If $2bs$ parties in $S \cap B$ have already used at least $k - t + 1$ signature tokens at ρ, take $\sigma = \rho$, choose B' to consist of $2bs$ such parties, and put $I = (S \cap B) \setminus B'$. Otherwise, follow G until the first prefix σ at which either $2bs$ parties in $S \cap B$ have used that many tokens or all parties in S have decided. Choose $I \subseteq S \cap B$ of size s among the parties that have not used that many tokens, and put $B' = (S \cap B) \setminus I$. In both cases, no party in I uses a signature token between ρ and σ. Set $T = O \cup I$; then $|T \cap B| = (2b + 1)s$ and $|T \setminus B| = r$.

Execution G^T. Choose a (ρ^+, T)-execution in which all messages with a sender or recipient in O remain delayed while O takes its local steps from G between ρ and σ. This is possible because $p \notin O$. Set GST at the end of these steps and let the execution conclude. Its correct parties decide $1 - v$.

Execution H^S. Only $F \cup I$ is corrupt from the beginning, reproducing G through σ. Beyond σ, F is silent and I behaves toward $S \setminus I$ as in G^S. Messages to $S \setminus I$ are delivered as in G^S, and all messages with a sender or recipient in O are delayed until the parties in $S \setminus I$ decide, choosing a sufficiently large GST.

Execution H^T. Only $F \cup I$ is corrupt from the beginning, reproducing the same prefix σ. Beyond σ, F is silent and I behaves toward O as in G^T. Messages to O are delivered as in G^T after its prescribed local steps, and all messages with a sender or recipient in $S \setminus I$ are delayed until the parties in O decide, choosing a sufficiently large GST. The parties in I have used no new signature tokens since ρ, so the additional token uses required by G^T remain available after σ, including any use in the step from ρ to ρ^+ if $p \in I$.

Indistinguishability arguments. The parties in $S \setminus I$ cannot distinguish H^S from G^S up to their decisions, because they receive the same messages from the same parties with the same timings. They therefore decide v in H^S.

The parties in O cannot distinguish H^T from G^T up to their decisions, because they receive the same messages from the same parties with the same timings. In particular, their steps and signature-token uses through σ are reproduced in G^T. They therefore decide $1 - v$ in H^T.

If G reached σ with all parties in S decided but fewer than $2bs$ parties having used the required tokens, then the correct parties in $S \setminus I$ would already have decided v in H^T, contradicting agreement. Thus every party in B' has used at least $k - t + 1$ signature tokens. The executions H^S, H^T extend the same prefix σ, have different correct decisions, and corrupt only $F \cup I$, whose size is $f' + s = 2s - q \leq f$. □

Proof of Theorem 5.2. Suppose, toward a contradiction, that such a protocol Π exists. Define $b_0 = 1$ and $b_t = 4b_{t-1} + 1$ for $t \geq 1$, so $b_t = (4^{t+1} - 1)/3$. Write $q = n - 2f > 0$. The assumed bound gives $f \geq 2b_{k-1}q$.

We first construct a prefix in which a set of correct parties has used at least one signature token each, together with two executions extending it with different correct decisions. We then repeatedly apply Lemma 5.5, adding corrupt parties and extending the prefix to obtain smaller sets of correct parties that have used at least two, then three, and eventually all k signature tokens each. Each application preserves two executions extending the new prefix with different correct decisions. After $k - 1$ applications, Corollary 5.4 gives a contradiction.

The sets of parties. Choose a set S of $n - f$ parties. Consider an execution in which these parties are correct with input 1, the remaining parties are corrupt and silent, and GST = 0. By an identical argument to the proof of Lemma 5.3, at least f parties in S must have used at least one SigT before a party in S decides. Let U be the first f distinct parties in S to use a SigT, let $R = S \setminus U$, and let $Z = [n] \setminus S$. Thus $|U| = |Z| = f$ and $|R| = q$.

The prefix π_0. Consider an execution in which only R is corrupt, the parties in U are correct with input 1, and the parties in Z are correct with input 0. The parties in R behave toward U as in the preceding execution, and messages within $U \cup R$ are delivered as in that execution until all parties in U have used at least one SigT. Denote this prefix by π_0. All messages with a sender or recipient in Z are delayed beyond π_0. No party in R has used a SigT in this prefix.

Execution E_1. Only R is corrupt. The execution extends π_0, with R completing its input-1 behavior toward U. Messages to U are delivered as in the initial execution on S, and all messages with a sender or recipient in Z are delayed until all parties in U decide, choosing a sufficiently large GST.

Execution E_0. Only R is corrupt. The execution extends π_0, after which R stops communicating with U and behaves toward Z as in an execution with $R \cup Z$ correct with input 0, U corrupt and silent, and all messages delayed through the initial prefix. In that execution, set GST at the end of the prefix. Deliver messages to Z accordingly, and delay all messages with a sender or recipient in U until all parties in Z decide, choosing a sufficiently large GST. This is well-defined because R has used no signature tokens in π_0.

Indistinguishability arguments. By the same indistinguishability and validity argument as in the proof of Theorem 5.1, the parties in U decide 1 in E_1, and the parties in Z decide 0 in E_0.

We now apply Lemma 5.5 to these two executions. Initially, take $F = R$, $t = k - 1$, and $s = 2q$, and choose $B \subseteq U$ of size $2b_{k-1}q$. Then $s = q + |F|$, $|B| = b_t s \leq f$, and every party in B has used at least $k - t$ signature tokens. The executions E_0, E_1 extend π_0 with only F corrupt and different correct decisions.

For $t \geq 1$, apply Lemma 5.5 with $b = b_{t-1}$. It gives a prefix σ, a corrupt set $F^+ = F \cup I$, and a set B' whose parties have used at least $k - t + 1$ signature tokens. Since $s' = q + |F^+| = 2s$ and $|B'| = 2b_{t-1}s = b_{t-1}s' \leq b_t s \leq f$, the two executions extending σ satisfy the same hypotheses with $t - 1$ in place of t.

After $k-1$ applications, all parties in the resulting B' have used all their signature tokens. In the final application, $b = b_0 = 1$, so $|B'| = 2s = q + |F^+|$ and $|F^+ \cup B'| = 4s - q \leq 5s \leq f$. Corollary 5.4 rules out the two executions extending σ, a contradiction. Finally,

$$f \geq 2b_{k-1}(n-2f) \quad \iff \quad f \geq \frac{2(4^k-1)}{4^{k+1}-1}\, n.$$

This concludes the proof of Theorem 5.2. □

6 Additional Related Work

Quantum consensus. The interplay between quantum information and consensus has been examined from several angles. Much of the existing literature on quantum consensus has focused on the power of quantum communication and entanglement against full-information, adaptive adversaries. In the synchronous setting, Bar-Joseph and Ben-Or [BJBO98] establish a lower bound of $\Omega(\sqrt{\frac{n}{\log n}})$ on the round complexity of *randomized* consensus against such an adversary, when $f = \Omega(n)$. In a follow-up work, Ben-Or and Hassidim [BOH05] establish quantum supremacy for consensus in this model, presenting a protocol employing quantum communication and entanglement, with $O(1)$ expected number of rounds. Subsequent work [HKO23] improved the concrete efficiency for crash faults. In the asynchronous setting, [LSZ24] improved the resilience from the $f < \frac{n}{4}$ bound of Ben-Or and Hassidim to $f < \frac{n}{3+\epsilon}$, by showing a black-box transformation turning classical private-channel consensus protocols into quantum full-information protocols. A second line of work explored the power of quantum information for the Byzantine Broadcast (BB) task with a relaxed validity notion, specifically one with *weak validity*: The sender's value must be decided only if *all* parties are correct. Beginning with Fitzi, Gisin, and Maurer [FGM01] and continuing in a series of works [Cab03; IG04; Cho22], this line of work shows that quantum communication can circumvent the seminal $f < \frac{n}{3}$ classical barrier of [FLM86], and boast protocols tolerating any number of failures (i.e. $f < n$) for the weak validity BB task. It turns out that the same resilience can be achieved classically, provided the communication channels are private [Fit+02].

Circumventing [DLS88]. Our result joins a broader effort to circumvent the Dwork-Lynch-Stockmeyer $f < \frac{n}{3}$ barrier by augmenting the standard partially synchronous setting. An impressive body of work has amassed in this direction, especially from the angle of the minimal amount of *synchrony* required to circumvent the barrier. Examples include the recently introduced granular synchrony model [Gir+24; ER25], which permits the synchrony of individual links to vary across the network, and recovers the $\frac{1}{2}$-resilience bound for Byzantine Agreement whenever the resulting communication graph is suitably connected.

In the same spirit is the work of [GPS19], in which, on top of the Byzantine parties, the adversary is allowed to temporarily delay the messages of a bounded number of honest parties. Other approaches include flexible Byzantine fault tolerance [MNR19], or multi-threshold consensus [MR21a]. Closest in spirit to our own approach is the work of [Yan+23]. In their paper, the authors use trusted hardware to prevent equivocation: This allows the model to reclaim the $f < \frac{n}{2}$ bound for consensus in asynchrony.

Acknowledgments. Yuval Efron's research is supported by the Institute for Advanced Study and a Kovner Foundation fellowship. We also thank the Columbia-Ethereum Research Center for

Blockchain Protocol Design for its support.

AI methodology.

Activity	AI used
Asking the research question	No
Coming up with the approach	No
Proof development	No
Writing and exposition	Yes
Checking for bugs	Yes
Other supporting tasks	Yes

Details. The research questions, overall approach, and main proof ideas are due to the authors. The protocol for $k = 1$ was already known to the authors before July 2026 [Efr+26], and the authors also developed the approach and proof ideas for its generalizations.

We used OpenAI Codex and Anthropic Claude primarily to sanity-check these ideas, identify bugs and gaps, and check that the protocol descriptions faithfully implemented the intended approach. These tools also contributed substantially to writing and exposition throughout the paper, preparing figures, and searching and checking the literature. The authors directed this process, examined the suggestions, and repeatedly reviewed and revised the protocol descriptions, arguments, and text.

References

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401. DOI: 10.1145/357172.357176.

[DS83] Danny Dolev and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM Journal on Computing* 12.4 (1983), pp. 656–666. DOI: 10.1137/0212045.

[Wie83] Stephen Wiesner. "Conjugate Coding". In: *ACM SIGACT News* 15.1 (1983), pp. 78–88. DOI: 10.1145/1008908.1008920.

[BB84] Charles H. Bennett and Gilles Brassard. "Quantum Cryptography: Public Key Distribution and Coin Tossing". In: *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing*. Archival reprint in Theoretical Computer Science 560, pp. 7–11, 2014, doi:10.1016/j.tcs.2014.05.025. Bangalore, India, 1984, pp. 175–179.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *Journal of the ACM* 32.2 (1985), pp. 374–382. DOI: 10.1145/3149.214121.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. "Easy Impossibility Proofs for Distributed Consensus Problems". In: *Distributed Computing* 1.1 (1986), pp. 26–39. DOI: 10.1007/BF01843568.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the Presence of Partial Synchrony". In: *Journal of the ACM* 35.2 (1988), pp. 288–323. DOI: 10.1145/42282.42283.

[BG93] Piotr Berman and Juan A. Garay. "Randomized Distributed Agreement Revisited". In: *23rd International Symposium on Fault-Tolerant Computing (FTCS)*. IEEE, 1993, pp. 412–419.

[BJBO98] Ziv Bar-Joseph and Michael Ben-Or. "A Tight Lower Bound for Randomized Synchronous Consensus". In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 1998, pp. 193–199. DOI: 10.1145/277697.277733.

[CL99] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 1999, pp. 173–186. URL: https://www.usenix.org/conference/osdi-99/presentation/practical-byzantine-fault-tolerance.

[FGM01] Matthias Fitzi, Nicolas Gisin, and Ueli Maurer. "Quantum Solution to the Byzantine Agreement Problem". In: *Physical Review Letters* 87.21 (2001), p. 217901. DOI: 10.1103/PhysRevLett.87.217901.

[Fit+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. "Detectable Byzantine Agreement Secure Against Faulty Majorities". In: *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2002, pp. 118–126. DOI: 10.1145/571825.571841.

[Cab03] Adán Cabello. "Solving the Liar Detection Problem Using the Four-Qubit Singlet State". In: *Physical Review A* 68.1 (2003), p. 012304. DOI: 10.1103/PhysRevA.68.012304.

[BOM04] Michael Ben-Or and Dominic Mayers. *General Security Definition and Composability for Quantum and Classical Protocols*. arXiv preprint arXiv:quant-ph/0409062. 2004.

[FHW04] Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. "Multi-party Computation with Hybrid Security". In: *Advances in Cryptology — EUROCRYPT 2004*. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 419–438. DOI: 10.1007/978-3-540-24676-3_25.

[IG04] S. Iblisdir and Nicolas Gisin. "Byzantine Agreement with Two Quantum-Key-Distribution Setups". In: *Physical Review A* 70.3 (2004), p. 034306. DOI: 10.1103/PhysRevA.70.034306.

[BOH05] Michael Ben-Or and Avinatan Hassidim. "Fast Quantum Byzantine Agreement". In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, 2005, pp. 481–485. DOI: 10.1145/1060590.1060662.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography". In: *Journal of Cryptology* 18.3 (2005), pp. 219–246. DOI: 10.1007/s00145-005-0318-0.

[KK06] Jonathan Katz and Chiu-Yuen Koo. "On Expected Constant-Round Protocols for Byzantine Agreement". In: *Advances in Cryptology — CRYPTO 2006*. Vol. 4117. Lecture Notes in Computer Science. Springer, 2006, pp. 445–462. DOI: 10.1007/11818175_27.

[Gar+07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. "Round Complexity of Authenticated Broadcast with a Dishonest Majority". In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2007, pp. 658–668. DOI: 10.1109/FOCS.2007.44.

[Nak08] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008.

[Aar09] Scott Aaronson. "Quantum Copy-Protection and Quantum Money". In: *Proceedings of the 24th Annual IEEE Conference on Computational Complexity*. IEEE Computer Society, 2009, pp. 229–242. DOI: 10.1109/CCC.2009.42. arXiv: 1110.5353 [quant-ph].

[FR09] Matthias Fitzi and Dominik Raub. *Tight Bounds for Protocols with Hybrid Security*. Cryptology ePrint Archive, Paper 2009/434. 2009. URL: https://eprint.iacr.org/2009/434.

[Unr10] Dominique Unruh. "Universally Composable Quantum Multi-party Computation". In: *Advances in Cryptology — EUROCRYPT 2010*. Vol. 6110. Lecture Notes in Computer Science. Springer, 2010, pp. 486–505. DOI: 10.1007/978-3-642-13190-5_25.

[KS11] Valerie King and Jared Saia. "Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine Agreement with an Adaptive Adversary". In: *Journal of the ACM* 58.4 (2011), 18:1–18:24. DOI: 10.1145/1989727.1989732.

[Gor+14] S. Dov Gordon, Jonathan Katz, Ranjit Kumaresan, and Arkady Yerukhimovich. "Authenticated Broadcast with a Partially Compromised Public-Key Infrastructure". In: *Information and Computation* 234 (2014), pp. 17–25. DOI: 10.1016/j.ic.2013.11.003.

[Bad+18] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. "Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 913–930. DOI: 10.1145/3243734.3243848.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The Latest Gossip on BFT Consensus*. arXiv preprint arXiv:1807.04938. 2018.

[JLS18] Zhengfeng Ji, Yi-Kai Liu, and Fang Song. "Pseudorandom Quantum States". In: *Advances in Cryptology – CRYPTO 2018*. Vol. 10993. Lecture Notes in Computer Science. Springer, 2018, pp. 126–152. DOI: 10.1007/978-3-319-96878-0_5. arXiv: 1711.00385 [quant-ph].

[Mah18] Urmila Mahadev. "Classical Verification of Quantum Computations". In: *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 2018, pp. 259–267. DOI: 10.1109/FOCS.2018.00033. arXiv: 1804.01082 [quant-ph].

[Abr+19] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. "Synchronous Byzantine Agreement with Expected $O(1)$ Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience". In: *Financial Cryptography and Data Security (FC)*. Vol. 11598. Lecture Notes in Computer Science. Springer, 2019, pp. 320–334. DOI: 10.1007/978-3-030-32101-7_20.

[GPS19] Yue Guo, Rafael Pass, and Elaine Shi. "Synchronous, with a Chance of Partition Tolerance". In: *Advances in Cryptology — CRYPTO 2019*. Vol. 11692. Lecture Notes in Computer Science. Springer, 2019, pp. 499–529. DOI: 10.1007/978-3-030-26948-7_18.

[MNR19] Dahlia Malkhi, Kartik Nayak, and Ling Ren. "Flexible Byzantine Fault Tolerance". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 1041–1053. DOI: 10.1145/3319535.3354225.

[Yin+19] 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 (PODC)*. ACM, 2019, pp. 347–356. DOI: 10.1145/3293611.3331591.

[Amo+20] Ryan Amos, Marios Georgiou, Aggelos Kiayias, and Mark Zhandry. "One-shot Signatures and Applications to Hybrid Quantum/Classical Authentication". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*. ACM, 2020, pp. 255–268. DOI: 10.1145/3357713.3384304.

[BFV20] Adam Bouland, Bill Fefferman, and Umesh Vazirani. "Computational Pseudorandomness, the Wormhole Growth Paradox, and Constraints on the AdS/CFT Duality". In: *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Vol. 151. Leibniz International Proceedings in Informatics (LIPIcs). Full version available as arXiv:1910.14646. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 63:1–63:2. DOI: 10.4230/LIPIcs.ITCS.2020.63. arXiv: 1910.14646 [quant-ph].

[Can20] Ran Canetti. "Universally Composable Security". In: *Journal of the ACM* 67.5 (2020), 28:1–28:94. DOI: 10.1145/3402457.

[Wan+20] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. "Expected Constant Round Byzantine Broadcast Under Dishonest Majority". In: *Theory of Cryptography Conference (TCC)*. Vol. 12550. Lecture Notes in Computer Science. Springer, 2020, pp. 381–411. DOI: 10.1007/978-3-030-64375-1_14.

[Col+21] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. "Hidden Cosets and Applications to Unclonable Cryptography". In: *Advances in Cryptology — CRYPTO 2021*. Lecture Notes in Computer Science. Springer, 2021, pp. 556–584. DOI: 10.1007/978-3-030-84242-0_20.

[Kre21] William Kretschmer. "Quantum Pseudorandomness and Classical Complexity". In: *16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*. Ed. by Min-Hsiu Hsieh. Vol. 197. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 2:1–2:20. DOI: 10.4230/LIPIcs.TQC.2021.2. arXiv: 2103.09320 [quant-ph].

[MR21a] Atsuki Momose and Ling Ren. "Multi-Threshold Byzantine Fault Tolerance". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2021, pp. 1686–1699. DOI: 10.1145/3460120.3484554.

[MR21b] Atsuki Momose and Ling Ren. "Optimal Communication Complexity of Authenticated Byzantine Agreement". In: *35th International Symposium on Distributed Computing (DISC)*. Vol. 209. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 32:1–32:16. DOI: 10.4230/LIPIcs.DISC.2021.32.

[Zha21] Mark Zhandry. "Quantum Lightning Never Strikes the Same State Twice. Or: Quantum Money from Cryptographic Assumptions". In: *Journal of Cryptology* 34.1 (2021). Article 6, pp. 1–56. DOI: 10.1007/s00145-020-09372-x. arXiv: 1711.02276 [cs.CR].

[Cho22] Vicent Cholvi. "Quantum Byzantine Agreement for Any Number of Dishonest Parties". In: *Quantum Information Processing* 21.4 (2022), p. 151. DOI: 10.1007/s11128-022-03492-y.

[Civ+22] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. "Byzantine Consensus Is $\Theta(n^2)$: The Dolev–Reischuk Bound Is Tight Even in Partial Synchrony!" In: *36th International Symposium on Distributed Computing (DISC 2022)*. Vol. 246. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 14:1–14:21. DOI: 10.4230/LIPIcs.DISC.2022.14.

[Shm22] Omri Shmueli. "Semi-Quantum Tokenized Signatures". In: *CRYPTO*. 2022, pp. 296–319.

[BDS23] Shalev Ben-David and Or Sattath. "Quantum Tokens for Digital Signatures". In: *Quantum* 7 (2023), p. 901. DOI: 10.22331/q-2023-01-19-901.

[CP23] Benjamin Y. Chan and Rafael Pass. "Simplex Consensus: A Simple and Fast Consensus Protocol". In: *Theory of Cryptography Conference (TCC)*. Vol. 14372. Lecture Notes in Computer Science. Cryptology ePrint Archive, Paper 2023/463. Springer, 2023, pp. 452–479. DOI: 10.1007/978-3-031-48624-1_17.

[HKO23] MohammadTaghi Hajiaghayi, Dariusz Rafal Kowalski, and Jan Olkowski. "Brief Announcement: Improved Consensus in Quantum Networks". In: *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2023, pp. 286–289. DOI: 10.1145/3583668.3594600.

[Yan+23] Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael K. Reiter. "Communication-Efficient BFT Using Small Trusted Hardware to Tolerate Minority Corruption". In: *26th International Conference on Principles of Distributed Systems (OPODIS 2022)*. Vol. 253. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 24:1–24:23. DOI: 10.4230/LIPIcs.OPODIS.2022.24.

[Bad+24] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. "Bitcoin as a Transaction Ledger: A Composable Treatment". In: *Journal of Cryptology* 37.2 (2024), p. 18. DOI: 10.1007/s00145-024-09493-7.

[Gir+24] Neil Giridharan, Ittai Abraham, Natacha Crooks, Kartik Nayak, and Ling Ren. "Granular Synchrony". In: *38th International Symposium on Distributed Computing (DISC)*. Vol. 319. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 30:1–30:22. DOI: 10.4230/LIPIcs.DISC.2024.30.

[LP+24] Andrew Lewis-Pye, Dahlia Malkhi, Oded Naor, and Kartik Nayak. "Lumiere: Making Optimal BFT for Partial Synchrony Practical". In: *Proceedings of the 2024 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2024, pp. 135–144. DOI: 10.1145/3662158.3662787.

[LSZ24] Longcheng Li, Xiaoming Sun, and Jiadong Zhu. "Quantum Byzantine Agreement Against Full-Information Adversary". In: *38th International Symposium on Distributed Computing (DISC)*. Vol. 319. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 32:1–32:22. DOI: 10.4230/LIPIcs.DISC.2024.32.

[Civ+25] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Manuel Vidigueira, and Igor Zablotchi. "Partial Synchrony for Free: New Upper Bounds for Byzantine Agreement". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 2025, pp. 4227–4291. DOI: 10.1137/1.9781611978322.144.

[Coh+25] Ran Cohen, Jack Doerner, Eysa Lee, Anna Lysyanskaya, and Lawrence Roy. "An Unstoppable Ideal Functionality for Signatures and a Modular Analysis of the Dolev-Strong Broadcast". In: *Theory of Cryptography Conference (TCC)*. Vol. 16271. Lecture Notes in Computer Science. Cryptology ePrint Archive, Paper 2024/1807. Springer, 2025, pp. 675–708. DOI: 10.1007/978-3-032-12290-2_22.

[CEK25] Daniel Collins, Yuval Efron, and Jovan Komatovic. "Juggernaut: Efficient Crypto-Agnostic Byzantine Agreement". In: *Advances in Cryptology — EUROCRYPT 2025*. Lecture Notes in Computer Science. Cryptology ePrint Archive, Paper 2024/1601. Springer, 2025. DOI: 10.1007/978-3-031-91092-0_3.

[ER25] Yuval Efron and Ling Ren. *Lifeline: Optimal Byzantine Agreement Under Minimal Synchrony*. Cryptology ePrint Archive, Paper 2025/1318. 2025. URL: https://eprint.iacr.org/2025/1318.

[KSW25] Quentin Kniep, Jakub Sliwinski, and Roger Wattenhofer. *Solana Alpenglow Consensus: Increased Bandwidth, Reduced Latency*. Anza white paper. Version 1.1, July 22, 2025. 2025. URL: https://www.anza.xyz/alpenglow-1-1.

[LPR25] Andrew Lewis-Pye and Tim Roughgarden. "Beyond Optimal Fault-Tolerance". In: *7th Conference on Advances in Financial Technologies (AFT)*. Vol. 354. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 15:1–15:23. DOI: 10.4230/LIPIcs.AFT.2025.15.

[MH25] Fermi Ma and Hsin-Yuan Huang. "How to Construct Random Unitaries". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*. STOC 2025. Association for Computing Machinery, 2025, pp. 806–809. DOI: 10.1145/3717823.3718254. arXiv: 2410.10116 [quant-ph].

[SZ25] Omri Shmueli and Mark Zhandry. "On One-Shot Signatures, Quantum vs Classical Binding, and Obfuscating Permutations". In: *Advances in Cryptology — CRYPTO 2025*. Lecture Notes in Computer Science. Cryptology ePrint Archive, Paper 2025/486. Springer, 2025. DOI: 10.1007/978-3-032-01878-6_12.

[Abr+26] Ittai Abraham, Sourav Das, Yuval Efron, and Jovan Komatovic. "Forget-IT: Optimal Good-Case Latency For Information-Theoretic BFT". In: *Proceedings of the 2026 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2026, pp. 478–488. DOI: 10.1145/3796701.3815965.

[CLPO26] Brendan Kobayashi Chou, Andrew Lewis-Pye, and Patrick O'Grady. *Minimmit: Fast Finality with Even Faster Blocks*. 2026. arXiv: 2508.10862v7.

[Col+26] Daniel Collins, Sisi Duan, Julian Loss, Charalampos Papamanthou, Giorgos Tsimos, and Haochen Wang. "Towards Optimal Parallel Broadcast Under a Dishonest Majority". In: *Financial Cryptography and Data Security (FC 2025)*. Springer, 2026. DOI: 10.1007/978-3-032-07024-1_20.

[Efr+26] Yuval Efron, Mahimna Kelkar, Saachi Mutreja, and Tim Roughgarden. *OSStracize: Quantum Cryptography in Distributed Computing*. Talk at the Foundations of Computer Science Seminar, Weizmann Institute of Science. July 6, 2026. URL: https://www.weizmann.ac.il/pages/event/31630.

[Jal+26] Mohammad Mussadiq Jalalzai, Kushal Babel, Jovan Komatovic, Tobias Klenze, Sourav Das, Fatima Elsheimy, Mike Setrin, John Bergschneider, and Babak Gilkalaye. *MonadBFT: Fast, Responsive, Fork-Resistant Streamlined Consensus*. 2026. arXiv: 2502.20692v4.

[Len+26] Christoph Lenzen, Julian Loss, Kecheng Shi, and Benedikt Wagner. "Byzantine Consensus in the Partially Authenticated Setting". In: *Proceedings of the 2026 ACM Symposium on Principles of Distributed Computing (PODC)*. ACM, 2026, pp. 282–292. DOI: 10.1145/3796701.3815929.

[ST26] Omri Shmueli and Ilan Tennenhouse. *Consensus in One Shot: Fast Agreement Beyond Classical Limits*. Cryptology ePrint Archive, Paper 2026/2269. 2026.