On the Limits of Consensus under Dynamic Availability and Reconfiguration

Javier Nieto[1], Joachim Neu[2], and Ling Ren[1]

[1]University of Illinois Urbana-Champaign, {jmnieto2,renling}@illinois.edu
[2]a16z Crypto Research, jneu@a16z.com

Abstract

Proof-of-stake blockchains require consensus protocols that support Dynamic Availability and Reconfiguration (which we call the DAR model). Here, Dynamic Availability means that the consensus protocol should remain live even if a large number of nodes temporarily crash, and Reconfiguration means it should be possible to change the set of operating nodes over time. State-of-the-art protocols inspired by the DAR model, such as Ethereum, Cardano's Ouroboros, or Snow White, require additional model features comprising external mechanisms or node capabilities that are difficult to justify, such as social consensus or the requirement that key evolution be performed even when nodes have crashed. The key result of this paper is the necessary and sufficient adversarial condition under which consensus can be achieved in the *plain* DAR model, *without any extra features*. We then introduce an additional feature to the model that is justified by proof-of-stake blockchain designs: honest nodes complete a sign-off procedure the moment they express intent to exit from the set of operating nodes. This additional feature reduces the power of the adversary relative to the plain DAR model and helps us obtain a bootstrapping gadget that is particularly simple and efficient in the common optimistic case of few reconfigurations and no double spending.

1 Introduction

Today's distributed systems are evolving from static, tightly coordinated deployments into open, decentralized environments with dynamic availability and reconfigurable membership. For instance, classical consensus protocols such as PBFT [CL99] assume a fixed set of always-online nodes (also called servers, replicas, or validators). This is in stark contrast to modern proof-of-stake (PoS) blockchains, where nodes may frequently go offline, join, or leave the system. To this end, current literature has studied two orthogonal dimensions of changing participation.

First, the idea of *reconfigurable* membership in distributed systems came early on to support adding, removing, or replacing nodes after initialization of the protocol. Whereas classical consensus is defined on a single fixed *membership set*, reconfigurable systems consist of a sequence of membership sets where a set of nodes makes consensus decisions for some time before the next set of nodes takes over [LMZ09, LMZ10, SRMJ12, OO14, BSA14, JM14].

Second, and orthogonal to reconfiguration, is the idea of *dynamic availability*, which was pioneered by the Bitcoin protocol [Nak09]. Bitcoin does not require a fixed quorum of nodes to make decisions. Its fault tolerance is thus inherently determined by the relative levels of participation, i.e., the proportion of honest versus adversarial computational power, at any given time. As a result, Bitcoin remains operational even if a large number of nodes temporarily stop participating. Pass and

Table 1: A comparison of protocols in the DAR model and its variants

Protocol	**Model**	**Adversarial condition**	**No external mechanism**	**No added node capability**
Bitcoin [Nak09]	PoW	Honest majority	✔	✔
Snow White [DPS19]	DAR with social consensus	Honest majority	✘	✔
Ethereum [BHK+20]	DAR with social consensus	Honest majority	✘	✔
Ouroboros Genesis [BGK+18]	DAR with continuous key evolution	Honest majority	✔	✘
Alg. 1	DAR	SR-HM (Defs. 3.3 and 3.4)*	✔	✔
Alg. 2	DAR with sign-off	SR-HM (Defs. 4.1 and 3.4)*	✔	✘

* SR-HM stands for simulation-resistant honest majority. The definition of simulation-resistant differs slightly under the plain DAR model and the DAR with sign-off model (see Defs. 4.1, 3.3 and 3.4).

Shi [PS17] first formalized this notion as the *sleepy model*, which is also referred to as the dynamic availability model in the literature [NTT21, DNTT24]. The model considers a *fixed* set of nodes, where, throughout execution, some honest nodes may become *asleep* and not participate in the protocol. Asleep nodes can be viewed as crashed nodes that may recover later. However, the sleepy model is not to be confused with mixed fault models [GP92, CMSK07, ADKS22, GDCL22], because the sleepy model allows for an arbitrary number of asleep nodes, whereas mixed fault models put a bound on the total number of Byzantine and crash-recovery faults.

It is then natural to combine the two orthogonal aspects into the *Dynamic Availability and Reconfiguration* (DAR) model. While prior works have explored models that appear similar to DAR [DPS19, BGK+18, BHK+20], a closer inspection reveals that these works incorporate additional external mechanisms or node capabilities into the model that are difficult to justify, as we elaborate below. Consequently, these works do not operate in what we call, for clarity, the *plain* DAR model, but rather in *variants* of the DAR model *with extra features*.

Snow White [DPS19] and PoS Ethereum [BHK+20] rely on an extra external mechanism called *social consensus*, which states that newly joining nodes obtain a recent checkpoint of the system through some mechanism. In practice, this mechanism must continuously achieve consensus on an up-to-date sequence of checkpoints so that outdated nodes can determine the latest state to correctly participate in the consensus protocol. The common rationale for social consensus is that newly joining nodes must anyway obtain a correct protocol implementation from some trusted source; thus, obtaining a recent checkpoint of the blockchain state appears to impose little additional requirement. However, this seemingly minor requirement is at odds with the core principles and established motivation of distributed consensus. While all consensus protocols require some initial common knowledge [HM84] (i.e., agreed-upon states), such as the protocol description itself and a few public parameters (like the genesis block in Bitcoin), the role of the consensus protocol is to subsequently process impromptu inputs and to continuously expand the agreed-upon state. Social consensus is an external mechanism that provides consensus on a growing sequence of checkpoints to external observers. It therefore seems unsatisfactory, or even circular, for a consensus protocol to assume the existence of an external consensus mechanism in order to solve consensus.

Ouroboros Genesis [BGK+18] is a longest-chain consensus protocol that appears to be designed

for the DAR model[1], and does not rely on social consensus. However, their model deviates from the plain DAR model by requiring asleep nodes to *continuously* perform *key evolution* to maintain forward security. Asleep nodes that fail to do so are counted as adversarial. However, this extra capability of asleep nodes seems to be at odds with the motivation of dynamic availability. The main goal of dynamic availability is to model nodes (stakeholders) that may not be interested in taking part in running the consensus protocol. It seems questionable to assume that these nodes *continuously* update their local state despite not actively participating in the protocol.

Thus, existing protocols achieve consensus only in variants of the DAR model that are augmented with powerful extra features that are hard to justify. This leaves open the question of under what condition consensus can be achieved in the *plain* DAR model.[2] In this work, as the first key contribution, we resolve the open question, and precisely characterize the *adversarial condition* that is both necessary and sufficient for achieving consensus in the plain DAR model. Carrying the classic threshold adversary to the DAR model, the adversarial condition covers the fraction of adversarial nodes, as well as nodes' sleep schedule and the reconfiguration schedule. In the process, we rigorously characterize and tackle the core difficulty of the DAR model: the uncertainty in who *can* participate (reconfiguration) and who *does* participate (availability). Prior works dodge this difficulty by relying on extra model features. Concretely, social consensus alleviates the uncertainty in who can participate by providing outdated nodes with oracle access to a recent state of the system, and key evolution while asleep alleviates the uncertainty in who has participated by ensuring that compromised keys cannot be used to retroactively forge participation for asleep nodes.

In the second half of this work, we propose our own feature to add to the plain DAR model: nodes are required, one time only, to execute a *sign-off* procedure when exiting the membership set, even if the node is asleep. While this can be regarded as an extra node capability, we contend it is realistic and representative of PoS blockchains where nodes leave the membership set by signing a transaction to transfer their stake to another node. We refer to this stronger variant of the DAR model as the *DAR with sign-off* model. Under this model, the power of the adversary is weakened, which allows consensus to be achieved under a weaker adversarial condition than the one required in the plain DAR model. We also give a protocol with much better efficiency than the one in the plain DAR model when only a small fraction of members is reconfigured per epoch and no double-spending transfers of stake are observed, which is often the case in real-world PoS blockchains.

We compare our models, adversarial condition, and protocols with those of the aforementioned prior works in Tab. 1.

Main Results.

- We formalize the necessary adversarial condition for solving consensus in the plain DAR model as *simulation-resistant honest majority* (SR-HM, Def. 3.4, see Thm. 3.9).

- We propose a generic bootstrapping gadget that transforms any consensus protocol secure in the dynamically available and non-reconfigurable model into a consensus protocol secure in the plain DAR model under the SR-HM condition (see Thm. 3.5), thereby establishing it as the sufficient adversarial condition for solving consensus in the plain DAR model.

[1] Ouroboros Genesis [BGK+18] uses the term dynamic availability to refer to what we define as the DAR model. We use the term dynamic availability to only refer to the availability/sleepy status of nodes, and we treat reconfiguration (membership changes) as a separate aspect of the model.

[2] One could run the protocols from prior works [DPS19, BGK+18, BHK+20] in the plain DAR model rather than their intended models and investigate the conditions under which they achieve consensus. We do not pursue this direction, as it is unlikely to uncover the tight condition.

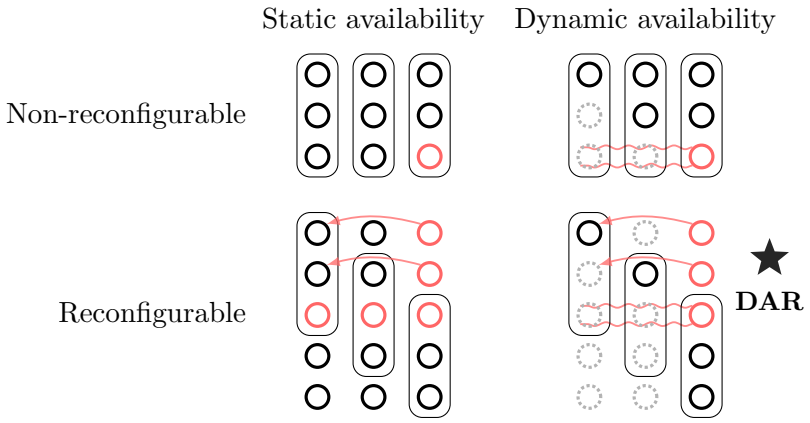

Figure 1: Models of reconfiguration and availability across three rounds. Here, "○" depicts awake and honest nodes, "○" are Byzantine nodes, and "○" are asleep nodes. "⟵" depicts long-range attacks, and "≈" depicts backward simulation.

- We introduce the *DAR with sign-off* model and propose a generic bootstrapping gadget with a simple and efficient good-case path that leverages signed membership-change transactions to achieve consensus in the DAR with sign-off model (see Thm. 4.6).

1.1 Overview

We briefly describe the difficulties in solving consensus under the different combinations of availability and reconfiguration. This will aid in our reasoning in the DAR model. We categorize models along two orthogonal dimensions: availability (static vs. dynamic), and membership (non-reconfigurable vs. reconfigurable). We label each model accordingly, such as "Dynamic Availability and Reconfiguration (DAR)" or "Dynamic Availability and Non-Reconfiguration (Sleepy Model)." We summarize the scenarios in Fig. 1.

Static Availability and Non-Reconfiguration These systems consider a fixed set of nodes where honest nodes are always available. Classic protocols such as PBFT [CL99] operate in this setting. Many protocols in this setting utilize digital signatures to bind nodes to the messages they send. Once a node receives sufficiently many signed messages, the node can construct a quorum *certificate*, a transferable proof that a particular event, or decision, has occurred with sufficient backing.

Static Availability and Reconfiguration These systems largely operate in the same manner as the classic protocols above. There is only a slight tweak in that the certificates produced throughout the protocol must be verified with respect to the corresponding membership set. Therefore, verification usually starts from the last known membership set and *iteratively* verifies every consensus decision and reconfiguration up to the most current output of the protocol.

Reconfiguration brings a subtlety known as the *long-range attack* [DPP19]. It may be unrealistic to expect an honest node that has since exited the system to maintain its honesty indefinitely. Thus, it is most desirable to place fault bounds only on the latest set of members. However, nodes that have exited the system still possess their cryptographic signing keys. If enough of them are corrupted by the adversary, the adversary can generate certificates for an alternative sequence of past decisions, essentially *rewriting history* in a way that appears valid to a newly joining node.

To mitigate this problem, Algorand [CM19] and Ouroboros [KRDO17, DGKR18, BGK+18] make use of forward-secure digital signatures [BM99, IR01, DGNW20, WTW+24] to prevent the adversary from forging signatures over past time periods after corrupting a node and obtaining its private state.

Dynamic Availability and Non-Reconfiguration (Sleepy Model) These systems operate from the same fixed set of nodes throughout the entire execution, but nodes may "go to sleep" for an arbitrary amount of time. As such, nodes that are awake must maintain liveness without the presence of all nodes. The sleepy model, therefore, assumes synchrony because it is impossible to solve consensus in the sleepy model if the network can be partitioned [PS17, NTT21, LR23] (see also Sec. 2.1).

Unlike reconfiguration, all nodes are assumed members, so there is no need to distinguish nodes that left or joined as members since the start of execution. Sleepy consensus protocols [PS17, MR22] assume an honest majority in the sense that awake and honest nodes outnumber adversarial nodes at any given time. Adversarial nodes are assumed to always be awake. Therefore, as participation grows, the adversary may corrupt more nodes. This becomes increasingly problematic since the adversary may *simulate* as if the newly corrupted nodes were already awake in the past. Consider the execution shown in Fig. 1 in the sleepy model (top-right), where there is one awake node at time 0, and three nodes awake at time 2. If we adapted techniques from the static availability setting, certificate sizes would be relative to the number of available nodes at the time of the event.

Thus, similarly to the long-range attack, the adversary may simulate alternative certificates and rewrite history in a way that appears valid to nodes waking up from their sleep—an attack known as *backward simulation* or *costless simulation* [DPS19, MMR23]. Forward-secure digital signatures do not resolve the problem in this setting, since, as discussed above, asleep nodes cannot perform key evolution. Recent works [DKT21, MR22, DNTT24, DSTZ25] solve this issue by completely ignoring past messages and only listening to the latest messages on what decisions to follow.

Dynamic Availability and Reconfiguration (DAR) Naturally, establishing a secure protocol in the DAR model inherits all the difficulties from the two prior models: nodes may be asleep, and the membership set is not fixed. Whenever a node is outdated, either because it has been asleep for some time or has recently joined the system, it must determine the latest state of the system using only the protocol logic and the initial membership set. We call this process *bootstrapping*. After reasoning about the prior two models, one can recognize the main challenge for bootstrapping: *current* active nodes alone dictate the state of the system (cf. Dynamic Availability and Non-Reconfiguration), but nodes bootstrapping into the system must verify *past* reconfigurations to know who to listen to at present (cf. Static Availability and Reconfiguration).

Bitcoin, a longest-chain proof-of-work (PoW) protocol, has a simple bootstrapping procedure with minimum assumptions: a newly joining node only requires the protocol logic, the initial genesis block, and connection to at least one honest and up-to-date node. Even if all other connections are to adversarial nodes, the newly joining node can distinguish the true chain by simply following the one with the most work. Our bootstrapping gadgets in the DAR model (both plain and with sign-off) also require minimum common knowledge and trust assumptions that are conceptually similar to Bitcoin's bootstrapping procedure, though we show that a stronger variant of the honest majority condition is required for any consensus protocol to be secure in the plain DAR model.

Organization. In Sec. 2, we outline basic aspects of our models that are used throughout the paper. In Sec. 3.1, we formalize reconfigurable consensus. In Sec. 3.2, we present our bootstrapping

gadget for constructing secure atomic broadcast protocols in the DAR model under the SR-HM condition, and prove that the SR-HM condition is necessary in Sec. 3.3. We consider signed membership changes (DAR with sign-off model) and give a bootstrapping gadget with a simple and efficient good-case path in Sec. 4. We discuss further related work in Sec. 5.

2 Model and Preliminaries

2.1 Model

We assume a public key infrastructure (PKI), where each node's public key is known to all others.

Synchrony. We consider a lock-step synchronous system in which nodes proceed in globally synchronized discrete *rounds* and are assumed to have synchronized clocks. Messages seen by *any* honest (and awake) node by the start of a round are seen by *all* honest (and awake) nodes by the end of the round. Subject to this constraint, the adversary can selectively reveal messages it sends to honest nodes. Note that all prior works on dynamic availability assume synchrony [PS17, NTT21, MR22, MMR23, DSTZ25, DNTT24, DPS19, BGK$^+$18]. It is easy to see that consensus with dynamic availability is impossible with partial synchrony or asynchrony [LPR24]. We now give a proof sketch. In a partially synchronous or asynchronous network, the adversary can partition awake and honest nodes into two disjoint sets and delay all messages between the two sets. To each set, the other set appears to be asleep. Due to dynamic availability and liveness, both sets must eventually decide, even though they never communicate with each other. The decisions will conflict, violating safety.

Adversary. The adversary is a probabilistic polynomial-time (PPT) algorithm that can fully adaptively *corrupt* nodes and put nodes to *sleep*. Corrupted nodes (also called *adversarial* or *Byzantine*) may deviate from the protocol arbitrarily in a manner coordinated by the adversary. We denote the corrupted nodes at a round t as $\mathcal{A}_t$. Once a node is corrupted, the node will never become honest again, so $\mathcal{A}_t \subseteq \mathcal{A}_{t+1}$. Uncorrupted nodes (called *honest*) follow the protocol when they are not asleep (as discussed below).

Sleepiness. Besides corruption, the adversary can fully adaptively put nodes to *sleep* at each round. *Asleep* nodes do not receive or send messages and do not execute the protocol. When an asleep node wakes up, it enters a state of *booting up* and receives any critical messages required to be up-to-date. During this time, the node initiates a *bootstrapping* process, after which it becomes fully *awake*. The process mirrors Bitcoin, where nodes must receive all critical block data to reconstruct the history. In practice, this bootstrapping is interactive and may span many rounds, with nodes querying neighbors for critical messages on demand and potentially re-bootstrapping later (e.g., when discovering a longer chain in Bitcoin). For simplicity, we model this interactive retrieval as instantaneous, as if sleeping nodes had buffered all necessary messages—though in practice, no such buffer is needed since nodes retrieve messages on demand upon waking. An awake and honest node executes the specified protocol. We denote awake and honest nodes at a round t as $\mathcal{H}_t$. Adversarial nodes are always considered awake and cannot be put to sleep.

2.2 Atomic Broadcast

Log. We define a *log* as a sequence of values x_i, denoted as $\Lambda = [x_0, x_1, x_2, ...]$. We say two logs are *compatible* if one is a prefix of the other. Otherwise, we say they are *conflicting*. At every round,

each node has an output log.

Atomic Broadcast. An atomic broadcast protocol [CASD95] allows nodes to agree on a log. Nodes may input requests into the protocol at each round. Logs that are output across honest nodes and across time satisfy the following properties:

- *Safety:* If two honest nodes (possibly the same node) at two points in time (possibly the same time) decide logs Λ and Λ', then Λ and Λ' are compatible.
- *Liveness:* If an awake and honest node inputs a value x, then eventually all awake and honest nodes decide a log containing x.
- *Validity:* If an honest log contains the value x, then x is input by some node.

2.3 Forward-Secure Signatures

We make use of a forward-secure signature scheme for signing messages in our protocols. *Forward security* ensures unforgeability of messages in the past—prior to key exposure—such that only messages after key exposure are vulnerable. Bellare and Miner's [BM99] forward-secure signature construction consists of a single public key $\textsc{pk}$ and an initial signing key denoted $\textsc{sk}_0$. They consider T time periods in which the signature scheme is used. The signer uses a different signing key in each period: $\textsc{sk}_1$ in period 1, $\textsc{sk}_2$ in period 2, and so on. At the beginning of period i, $\textsc{sk}_i$ is derived as a one-way function on $\textsc{sk}_{i-1}$. This is the process of *evolving* the key, and, once complete, the signer deletes the previous key $\textsc{sk}_{i-1}$. Thus, an adversary that corrupts the signer during period i cannot forge messages in prior periods. Let $\mathcal{F}_{\text{FS}}$ be a key-evolving signature scheme parametrized by security parameter λ and the total number of time periods T. We define $\mathcal{F}_{\text{FS}}$ with the following four algorithms taken from Bellare and Miner [BM99] for a signer v:

- **Key Generation:** $(v.\textsc{pk}, \textsc{sk}_0) \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Gen}()$. Signer v runs the key generation algorithm to generate a public verification key $\textsc{pk}$ and an initial secret signing key $\textsc{sk}_0$.
- **Key Update:** $\textsc{sk}_{i+1} \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\textsc{sk}_i)$. Signer v updates its secret key $\textsc{sk}_i$ to $\textsc{sk}_{i+1}$ for the next time period $i+1$. We may also use $\textsc{sk}_{i'} \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\textsc{sk}_i, i')$ to repeatedly apply the update to any time period $i' > i$.
- **Sign:** $\langle m \rangle_i^v \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Sign}(\textsc{sk}_i, m)$. Signer v uses the current signing key $\textsc{sk}_i$ and a message m to compute signature $\langle m \rangle_i^v$. For convenience, we assume the signature $\langle m \rangle_i^v$ carries the signer's identity v and the time period i in which the signature is produced.
- **Verification:** $b \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Verify}(\langle m \rangle_i^v)$. Anyone may verify the signature $\langle m \rangle_i^v$ over message m for time period i under the public key $v.\textsc{pk}$ by running the verification algorithm, which outputs 1 if the signature is valid and 0 otherwise.

As an example, we can use the Pixel forward-secure signature [DGNW20], which supports key update, signing, and verification in $O(1)$ cryptographic operations (i.e., group exponentiations and pairings). Both the signature size and public key size are constant. The secret key size is $O(\log^2 T)$ group elements, where T is an upper bound on the number of time periods for which the scheme is used. For instance, if each time period lasts one second, then $T = 2^{32} - 1$ periods would be 136 years, giving a secret key of 43KB in size [DGNW20].

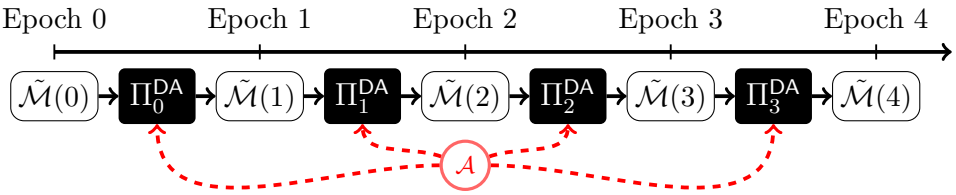

Figure 2: Overview of reconfiguration every epoch. "⇢" indicates the adversary's membership input into Π_e^{DA}.

3 Bootstrapping in Plain DAR

This section presents our detailed model for reconfiguration (Sec. 3.1), our atomic broadcast protocol (Sec. 3.2), and a matching impossibility result (Sec. 3.3) in the plain DAR model. We make no use of extra features through additional external mechanisms or node capabilities to solve the bootstrapping problem. In contrast, in Sec. 4, we will study a variant of the model where exiting nodes must authorize their withdrawal, as is the case in PoS systems.

3.1 Modeling Reconfiguration

The initial membership set $\mathcal{M}_0$, a finite set of nodes, is common knowledge. Subsequent membership sets $\mathcal{M}_t$, for rounds $t > 0$, are not known a priori to all nodes. Note that an honest node knows whether it is awake, but not whether it is a current member. We assume a fixed membership size, i.e., $|\mathcal{M}_0| = |\mathcal{M}_t|$ for all rounds $t \geq 0$.

Epochs. Time is divided into fixed-length epochs of R rounds, a parameter chosen beforehand that is common knowledge. Membership is fixed during each epoch and may change across epochs. Denote the *decided* membership for epoch e as $\tilde{\mathcal{M}}(e)$. Then, we have $\tilde{\mathcal{M}}(e) = \mathcal{M}_{eR} = \mathcal{M}_{eR+1} = ... = \mathcal{M}_{eR+R-1}$.

Let Π_e^{DA} be the instance of a dynamically available, non-reconfigurable atomic broadcast protocol Π^{DA} associated with epoch e, which is run by nodes in $\tilde{\mathcal{M}}(e)$. At each epoch $e \geq 0$, the adversary may select a new set of nodes M and input it into the atomic broadcast protocol Π_e^{DA} through its own input channel. At the end of the epoch, awake and honest nodes can set $\tilde{\mathcal{M}}(e+1)$ as the latest decided membership set in their log and use it as the next epoch's membership. We illustrate the process in Fig. 2. An external mechanism that instead announces each membership to all nodes would amount to social consensus: an outdated node could learn the current membership by simply waiting for the next announcement. Instead, letting the adversary propose membership sets captures that many possible sets can exist, and a node not participating in Π_e^{DA} cannot tell which one took effect. Agreement through Π_e^{DA} guarantees that the awake and honest nodes in $\tilde{\mathcal{M}}(e)$ settle on a single membership set $\tilde{\mathcal{M}}(e+1)$ and are the only nodes guaranteed to learn of it.

The liveness of Π_e^{DA} ensures that $\tilde{\mathcal{M}}(e+1)$ is *eventually* included in the logs of all awake and honest nodes. However, liveness alone does not guarantee that $\tilde{\mathcal{M}}(e+1)$ is included *by all awake and honest nodes by the end of epoch* e. Specifically, the concern is that, at the end of epoch e, some honest nodes have $\tilde{\mathcal{M}}(e+1)$ in their output log, while other honest nodes do not. Thus, we require an additional property of Π_e^{DA} that ensures the membership set $\tilde{\mathcal{M}}(e+1)$ is in the decided logs of all awake and honest nodes at the end of epoch e. Such a property is easily attainable for common atomic broadcast implementations with simple transformations, albeit with a slight reduction in performance. For instance, given a dynamically available protocol with a worst-case latency of k rounds, a possible transformation is to require "empty proposals" during the last k rounds of

the epoch. The last k rounds do not contribute new inputs, decreasing the protocol's throughput. However, it provides adequate time for all awake and honest nodes to settle on a common output by the end of the epoch. Nodes then wait to apply any proposed membership set until the end of the epoch. Under this model, we define the standard honest majority of a schedule (see Def. 3.1) in Def. 3.2, which we show in Sec. 3.3 is insufficient to guarantee safety and liveness of an atomic broadcast protocol in the DAR model. Note that honest majority is the tight condition under which consensus is feasible in each of the two settings that the DAR model combines: static availability with reconfiguration [KRDO17, DGKR18] and dynamic availability without reconfiguration (i.e., the sleepy model [PS17, MR22]). The insufficiency of an honest majority in the DAR model thus formalizes the sense in which the combination is harder than its parts.

Definition 3.1 (Schedule)**.** A schedule K defines the sequences of sets $\mathcal{A}_t$, $\mathcal{H}_t$, and $\mathcal{M}_t$ of adversarial nodes, awake and honest nodes, and membership at every round $t \geq 0$. Note that the corruption, sleep, and membership schedule is chosen by the adversary.

Definition 3.2 (Honest Majority (HM))**.** A schedule satisfies *honest majority* if for every round $t \geq 0$,

$$|\mathcal{M}_t \cap \mathcal{A}_t| < |\mathcal{M}_t \cap \mathcal{H}_t|.$$

Simulation. To capture what is sufficient and necessary for solving consensus in the DAR model, we pivot to the notion of *simulation*. While asleep, a node does not perform protocol actions: it neither sends nor receives messages, nor evolves its keys. If such a node later becomes corrupted, the adversary can retroactively *simulate* its behavior, making it appear as though it had actively participated in past rounds despite being asleep. We call such nodes *simulatable*. Note that the node is only simulatable up to the last time it was awake and honest, since it may have performed a key evolution at that point, preventing the adversary from signing messages for earlier rounds. Def. 3.3 gives the precise definition of simulatable nodes.

Definition 3.3 (Simulatable Nodes)**.** Denote the set of nodes that have been awake and honest in any round during the range of rounds $[s, t]$ as

$$\mathcal{H}(s,t) \triangleq \bigcup_{t' \in [s,t]} \mathcal{H}_{t'}.$$

Then, denote the set of nodes that are simulatable at round s by the adversary at round t as

$$\mathcal{S}(s;t) \triangleq \mathcal{A}_t \setminus \mathcal{H}(s,t).$$

Using Def. 3.3, we introduce the *simulation-resistant honest majority* (SR-HM) in Def. 3.4, which strengthens the standard honest majority (HM, Def. 3.2). In Sec. 3.3, we prove that the standard honest majority over every round t is insufficient for consensus.

Definition 3.4 (Simulation-Resistant Honest Majority (SR-HM))**.** A schedule (Def. 3.1) satisfies *simulation-resistant honest majority* if for all rounds $t \geq 0$ and $s \leq t$, the adversarial and simulatable nodes at round s are fewer than the awake and honest nodes since round s, i.e.,

$$|\mathcal{M}_s \cap \mathcal{S}(s;t)| < |\mathcal{M}_s \cap \mathcal{H}(s,t)|\,.$$

Algorithm 1 Reconfigurable atomic broadcast for node v

1 **during** epoch e, **if** awake
2 Execute Π^{DA} among members in M_e and store output in Λ
3 **at** the last round of epoch e
4 $\langle \Lambda \rangle_e^v \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Sign}(\text{SK}_e, \Lambda)$ // Sign with forward-secure signature
5 Send $(\mathsf{LVOTE}, \langle \Lambda \rangle_e^v)$ to all // Send out vote for latest log
6 $\mathsf{M}_{e+1} \leftarrow \Lambda.\textsc{newMembers}()$ // Get membership set for next epoch from log
7 $\text{SK}_{e+1} \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\text{SK}_e)$ // Evolve secret key to latest epoch
8 **upon** boot up during epoch e
9 **if** first time booting up
10 $(\text{PK}, \text{SK}_0) \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Gen}()$ // Generate keys
11 $\mathsf{M}_0 \leftarrow \mathcal{M}_0$ // Initialize as initial membership set
12 $\ell \leftarrow 0$
13 **else**
14 $\ell \leftarrow$ last epoch v was awake
15 Receive LVOTE messages from network with valid forward-secure signatures
16 **for** $e' \leftarrow \ell, \ell+1, ..., e-1$
17 $\mathsf{votes} \leftarrow$ oldest LVOTE messages at or after epoch e' from members in $\mathsf{M}_{e'}$
18 $\Lambda_{e'} \leftarrow$ most voted log up to epoch e' in votes
19 $\mathsf{M}_{e'+1} \leftarrow \Lambda_{e'}.\textsc{newMembers}()$ // Get membership set for next epoch from log
20 $\text{SK}_e \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\text{SK}_\ell, e)$

3.2 Bootstrapping Gadget in Plain DAR with Simulation-Resistant Honest Majority

In this section, we present an atomic broadcast protocol in the DAR model under SR-HM by layering a bootstrapping gadget atop the dynamically available, non-reconfigurable protocol Π^{DA}. The bootstrapping gadget enables outdated nodes to learn the current membership set and join the current epoch's instance of Π^{DA}. At the end of each epoch, the gadget has awake nodes produce certification votes for the latest decided log, which also behaves as a vote for every prefix of the log. Booting nodes collect these votes upon awakening. They follow the log with the most votes from their last known membership set at each epoch until they reach the latest epoch. Although the decided log at each epoch may not have an absolute quorum of votes due to dynamic availability, we prove it safe to follow the log with the "heaviest votes" under SR-HM, since, for every round, the number of awake and honest members is greater than the number of simulatable members. The bootstrapping gadget is detailed in Alg. 1.

Awake Participation. Awake nodes send inputs to Π_e^{DA} during epoch e and output the decided log (ln. 2). At the end of the epoch, nodes produce a forward-secure signature over the log output Λ to send out as a *vote* over the decided log (lns. 4 and 5). Then, nodes obtain the next membership set from Λ, denoted as M_{e+1} for membership at epoch $e+1$ (ln. 6). Lastly, awake nodes evolve their secret keys to the next epoch (ln. 7).

Bootstrapping. The bootstrapping process starts at ln. 8 when a node v boots up during an epoch e. If this is the first time v is awake, it initializes its keys for the forward-secure signature scheme $\mathcal{F}_{\text{FS}}$ and sets its last known membership set to the initial membership set $\mathcal{M}_0$. The booting

node v then receives all LVOTE messages and verifies their signatures (ln. 15). The bootstrapping process proceeds from v's last known membership set M_ℓ at epoch ℓ. For every epoch $e' \in [\ell, e)$, v collects the oldest votes at or after epoch e' for logs from nodes in $\mathsf{M}_{e'}$ as the set votes (ln. 17). We are interested in the oldest vote since an honest node may be corrupted after issuing a vote, and we want to ensure that we capture all honest votes concerning epoch e'. At ln. 18, v tallies the number of votes for logs up to epoch e' and selects the log with the most votes as $\Lambda_{e'}$. Then, v extracts the next membership set $\mathsf{M}_{e'+1}$ from $\Lambda_{e'}$ (ln. 19). After iterating through every epoch, v has identified the latest membership set and joins the atomic broadcast protocol Π_e^{DA}.

Efficiency Overhead. Beyond its participation in Π^{DA}, each awake node additionally sends a LVOTE message to all nodes at the end of every epoch. Any node booting up at epoch e verifies the LVOTE messages for each missed epoch since it was last awake.

Theorem 3.5. *Under the SR-HM condition (Def. 3.4), the protocol shown in Alg. 1 implements atomic broadcast in the plain DAR model.*

Proof of Thm. 3.5. For every epoch e, since the membership is fixed for the entire epoch, the protocol is safe, live, and valid for awake and honest nodes due to the safety, liveness, and validity of Π_e^{DA}. It only remains to show that if an honest node v booting up during epoch e participates, it does so with the same membership set $\tilde{\mathcal{M}}(e)$ as the awake and honest nodes in that epoch.

We will inductively show that lns. 8 to 19 identify the decided membership sets from 0 to e. Let t be the round in epoch e in which node v is booting up. This is the same round and epoch as current awake nodes due to synchronized clocks. For the base case, we have that the initial membership set $\mathcal{M}_0$ is common knowledge. For the inductive step, assume that v has selected the decided membership set $\tilde{\mathcal{M}}(e')$ for epoch $e' < e$. We now show that v selects $\tilde{\mathcal{M}}(e'+1)$ as the confirmed membership set for epoch $e'+1$ at ln. 19.

Consider a false log $\widehat{\Lambda}$ that is conflicting with the decided log $\Lambda_{e'}$. Let c and $\widehat{c}$ be the total count of votes for $\Lambda_{e'}$ and $\widehat{\Lambda}$, respectively, in votes (ln. 17). Let s be the last round of epoch e'. By the induction hypothesis, all observed votes from round s are from members in $\tilde{\mathcal{M}}(e')$. Thus, we must have that $c \geq |\mathcal{M}_s \cap \mathcal{H}(s,t)|$, because every honest member in $\mathcal{M}_s$ that was awake at some round at or after s signed and sent out a vote for a log that extends $\Lambda_{e'}$. We also have that $\widehat{c} \leq |\mathcal{M}_s \cap (\mathcal{A}_s \cup \mathcal{S}(s;t))|$ since, by the definition of the forward-secure signature scheme, only adversarial and simulatable nodes at round s have the secret keys available to sign a vote for a false log. Since $\mathcal{A}_s \subseteq \mathcal{S}(s;t)$, by the SR-HM condition (Def. 3.4), we have that

$$\widehat{c} \leq |\mathcal{M}_s \cap \mathcal{S}(s;t)| < |\mathcal{M}_s \cap \mathcal{H}(s,t)| \leq c.$$

Thus, we must have that the most voted log for epoch e' is $\Lambda_{e'}$, which v uses to retrieve the next decided membership set $\tilde{\mathcal{M}}(e'+1)$. By induction, we have that node v identifies the latest membership set $\mathsf{M}_e = \tilde{\mathcal{M}}(e)$. □

3.3 Necessity of Simulation-Resistant Honest Majority

We now show that SR-HM is necessary for any atomic broadcast protocol to be safe, live, and valid in the DAR model. Importantly, we restrict our attention to *identity-agnostic* conditions, meaning that relabeling nodes in a schedule does not change whether the condition holds. We prove that no *identity-agnostic* condition that fails to imply SR-HM can suffice for an atomic broadcast protocol. Note that both the HM and SR-HM conditions are identity-agnostic.

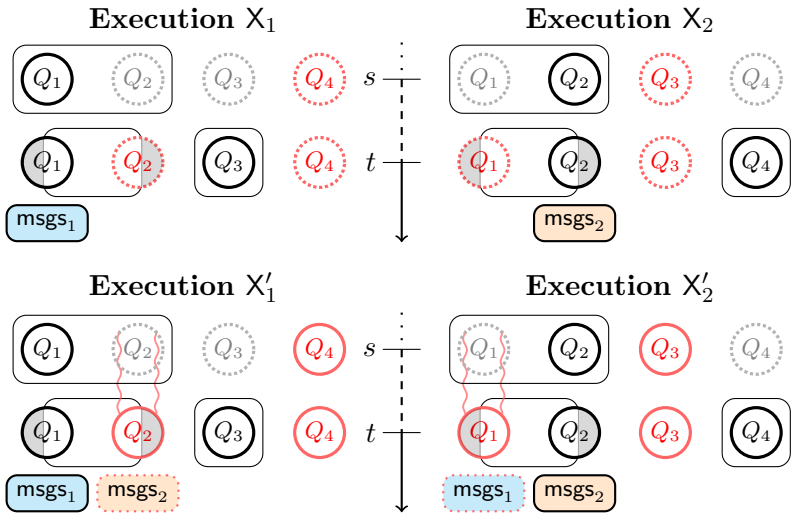

Figure 3: Four executions for possible schedules K_1 and K_2 under condition C. A node booting up in executions X_1 and X'_1 (left) expects to output Λ_1. A node booting up in executions X_2 and X'_2 (right) expects to output Λ_2. Here, msgs_1 denotes the messages of $Q_1 \cup Q_3$ and msgs_2 those of $Q_2 \cup Q_4$. At round t, an arbitrary portion of Q_1 and Q_2 may have withdrawn, where the shaded portion of each set indicates the members withdrawn since s. Recall that "○" depicts awake and honest nodes, "○" are Byzantine nodes, "○" are asleep nodes, "○" are Byzantine nodes that behave as if asleep, and "≈" depicts backward simulation.

Definition 3.6 (Identity-Permuted Schedule)**.** For a schedule K and a permutation ϕ of node identities, define the *identity-permuted schedule* $\phi(\mathsf{K})$ to be the schedule in which node $\phi(v)$ inherits the full local view and state (i.e., corruption, honesty, and membership) of node v across all rounds in K. We say two schedules K and K' are identity-permuted schedules if there exists a permutation ϕ such that $\mathsf{K}' = \phi(\mathsf{K})$.

Definition 3.7 (Identity-Agnostic Condition)**.** A condition C over schedules is said to be *identity-agnostic* if for all schedules K, whenever C holds for K, C also holds for all permuted schedules of K, i.e., for all schedules K and permutations ϕ, we have that

$$\mathsf{C}(\mathsf{K}) \iff \mathsf{C}(\phi(\mathsf{K})).$$

Definition 3.8 (Execution)**.** We define an *execution* X as a sequence of messages and states of all nodes under a schedule K during execution of a protocol Π.

Theorem 3.9. *No atomic broadcast protocol can be safe, live, and valid in the plain DAR model under an identity-agnostic condition (Def. 3.7) that does not imply the SR-HM condition (Def. 3.4).*

Proof of Thm. 3.9. Let C be an identity-agnostic condition that does not imply SR-HM. For the sake of a contradiction, suppose that Π is an atomic broadcast protocol that satisfies safety, liveness, and validity in the DAR model under C. Since C does not imply SR-HM, the adversary can find a schedule K_1 satisfying C with rounds $t \geq 0$ and $s \leq t$ where SR-HM does not hold, i.e.,

$$|\mathcal{M}_s \cap \mathcal{S}(s;t)| \geq |\mathcal{M}_s \cap \mathcal{H}(s,t)|\,.$$

Let $Q_1 = \mathcal{M}_s \cap \mathcal{H}(s,t)$ and $Q_2 \subseteq \mathcal{M}_s \cap \mathcal{S}(s;t)$ such that $|Q_1| = |Q_2|$. Let $Q_3 = \bigcup_{t' \in (s,t]} \mathcal{M}_{t'} \setminus \mathcal{M}_s$ be the new set of members from round $s+1$ to t and let $Q_4 \subseteq \mathcal{S}(s;t)$ such that $|Q_3| = |Q_4|$ and Q_4

is disjoint from $Q_2 \cup Q_3$. This allows us to define a permutation ϕ (Def. 3.6) that swaps the nodes in Q_1 with those in Q_2 and the nodes in Q_3 with those in Q_4, while leaving the remaining nodes unchanged. Because C is identity-agnostic, it also holds for $\mathsf{K}_2 = \phi(\mathsf{K}_1)$.

We now describe executions X_1 and X'_1 with schedule K_1 and executions X_2 and X'_2 with schedule K_2. Fig. 3 illustrates the executions for two example schedules. We have all adversarial nodes in executions X_1 and X_2 behave as if asleep. Then, let msgs_1 be the set of messages sent by nodes in $Q_1 \cup Q_3$ during X_1, and msgs_2 be the messages sent by $Q_2 \cup Q_4$ during X_2. Let Λ_1 be the decided log of nodes in $Q_1 \cup Q_3$ at round t during X_1, and let Λ_2 be the decided log of nodes in $Q_2 \cup Q_4$ at round t during X_2 such that Λ_2 conflicts with Λ_1. For nodes not in the sets Q_1, Q_2, Q_3, or Q_4, we have them behave identically in both executions. Let msgs^* be the set of messages sent by these nodes. For executions X'_1 and X'_2, the adversary behaves in the following manner for each execution.

Execution X'_1. By round t, the nodes in $Q_2 \cup Q_4$ must be simulatable at rounds $s, ..., t$. They then send the same set of messages msgs_2 sent by the awake and honest nodes in X_2, simulating a past view in this execution in which they were awake. By round t, nodes decide on log Λ_1. An honest booting node v will then receive messages $\mathsf{msgs}^* \cup \mathsf{msgs}_1 \cup \mathsf{msgs}_2$ and, by safety and liveness of Π, will output log Λ_1.

Execution X'_2. The execution proceeds symmetrically to X'_1 where nodes in $Q_1 \cup Q_3$ are simulatable at rounds $s, ..., t$. They send the same set of messages msgs_1 sent by the awake and honest nodes in X_1. By round t, nodes decide on a log Λ_2 that conflicts with Λ_1. An honest booting node v will then receive messages $\mathsf{msgs}^* \cup \mathsf{msgs}_1 \cup \mathsf{msgs}_2$ and, by safety and liveness of Π, will output log Λ_2.

Contradiction. The booting node v receives the same set of messages $\mathsf{msgs}^* \cup \mathsf{msgs}_1 \cup \mathsf{msgs}_2$ in both executions X'_1 and X'_2. Therefore, v cannot distinguish between the two executions, and we have a contradiction: either v outputs Λ_2 in X'_1 and violates safety, or v outputs Λ_1 in X'_2 and violates safety. Thus, no atomic broadcast protocol can be safe and live for any identity-agnostic condition that does not imply the SR-HM condition (Def. 3.4). □

4 Bootstrapping in DAR with Sign-Off

In this section, we detail our variant of the DAR model that incorporates a sign-off procedure before nodes exit the membership set. The plain DAR model captures general reconfigurations, including situations in which an asleep node is removed from the membership set without any action from the node. However, in real-world PoS systems, nodes typically sign a transaction to transfer their stake, even if they are not actively participating in the protocol. Therefore, the DAR with sign-off model is a natural augmentation of the plain DAR model that captures a node's participation before exiting the membership set, even if the node is asleep. This is not to be confused with the DAR model with continuous asleep participation required by the Ouroboros Genesis protocol [BGK$^+$18], as mentioned in Sec. 1. The sign-off procedure is only executed once before a node withdraws from the membership set and is not continuous. We discuss how the sign-off procedure is modeled in Sec. 4.1.

In Sec. 4.2, we present our concrete sign-off procedure and a more efficient bootstrapping gadget in this model. Naturally, we leverage sign-off to have nodes sign a transaction that transfers their membership to another node. We also have nodes delete their private keys, which we term *key disposal*. This weakens the sufficient adversarial condition required to solve the bootstrapping problem and also allows for more efficient bootstrapping than our prior gadget.

4.1 Modeling Sign-Off

We now extend the model described in Sec. 3.1 to incorporate sign-off behavior. Recall that during an epoch $e \geq 0$, the adversary may input a membership set into Π_e^{DA}, and the latest decided membership set at the end of the epoch becomes the next membership set $\tilde{\mathcal{M}}(e+1)$. With sign-off, this functionality remains unchanged, but we additionally assume that there exists a one-to-one mapping $\tau : \tilde{\mathcal{M}}(e) \setminus \tilde{\mathcal{M}}(e+1) \to \tilde{\mathcal{M}}(e+1) \setminus \tilde{\mathcal{M}}(e)$ from the set of withdrawing nodes to the set of replacement nodes. Then, for every $(v, v') \in \tau$, $v.\textsc{SignOff}(v')$ is executed by the end of epoch e. Notably, the mapping τ is only supplementary data that is unknown to honest nodes and is not input into Π_e^{DA}.[3] The sign-off function specifies the sign-off behavior of the protocol, such as signing a transaction and key disposal. The withdrawing node v, whether awake or not, is assumed to know the current epoch e and to be connected to at least one awake and honest node in $\tilde{\mathcal{M}}(e)$.

Disposal. With a weaker adversary in the DAR with sign-off model, we may weaken the SR-HM condition. In particular, consider that the adversary may no longer simulate past behaviors of withdrawn asleep nodes due to irreversible changes during the sign-off procedure, such as key disposal. We say a node is *disposed* if it has executed its sign-off function. Among nodes that have withdrawn, only corrupted nodes may avoid being disposed of. Therefore, we define the set of simulatable nodes in the DAR with sign-off model in Def. 4.1.

Definition 4.1 (Simulatable Nodes in the Sign-Off Model)**.** Denote the set of nodes that, while not corrupted, withdrew from a membership set during the range of rounds $[s, t)$ as

$$\mathcal{W}(s,t) \triangleq \bigcup_{t' \in [s,t)} (\mathcal{M}_{t'} \setminus \mathcal{M}_{t'+1}) \setminus \mathcal{A}_{t'}.$$

Then, denote the set of nodes that are *simulatable* (Def. 3.3) at round s by the adversary at round t as

$$\mathcal{S}_{\overline{\mathcal{W}}}(s;t) \triangleq (\mathcal{A}_t \setminus \mathcal{H}(s,t)) \setminus \mathcal{W}(s,t).$$

With the new definition of simulatable nodes in the DAR with sign-off model, the set $\mathcal{S}(s;t)$ in the SR-HM condition (Def. 3.4) is replaced by $\mathcal{S}_{\overline{\mathcal{W}}}(s;t)$ in this section.

4.2 Bootstrapping Gadget with Sign-Off

In this section, we present our bootstrapping gadget under the SR-HM condition (Def. 3.4) among simulatable nodes in the sign-off model (Def. 4.1). During sign-off, nodes are required to send a signed transaction that transfers their membership to another node and dispose of their private keys. The gadget augments our prior gadget by leveraging this sign-off procedure. In addition to votes, booting nodes may also utilize transactions (see Def. 4.2) to construct membership sets (see Def. 4.3).

Definition 4.2 (Transaction)**.** A transaction $\langle (v, v') \rangle_e^v$ for nodes v, v' signed by node v during epoch e transfers the membership of v to node v'. The transaction is *valid with respect to a membership set* M if node $v \in \mathsf{M}$ and $\mathcal{F}_{\text{FS}}.\textsc{Verify}(\langle (v, v') \rangle_e^v) = 1$.

Definition 4.3 (Transactions Constructing a Membership Set)**.** Let $\textsc{ApplyTxs}(\mathsf{M}, \mathcal{T})$ for a set of nodes M be the public function that outputs the membership set after applying transactions in $\mathcal{T}$.

[3] This models a slightly weaker constraint than typical PoS systems that reach consensus on each transaction for stake changes.

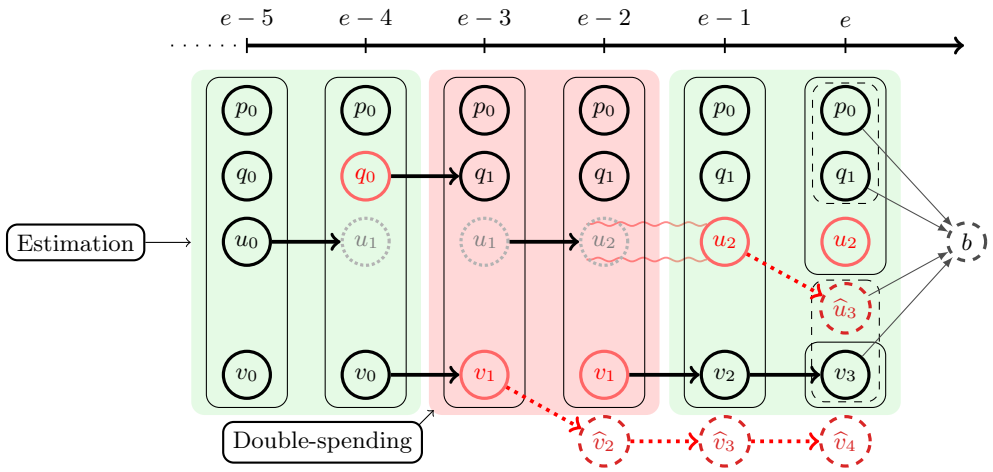

Figure 4: Example of estimating membership from the view of a booting node b. Here, "⟶" are decided transactions, "⇢" are undecided transactions, "⟶" are membership votes, and "▯" are non-members. The green regions indicate periods of estimation. The red region indicates a period with a double spender (v_1) that requires voting to identify the true membership change. The second estimation period has *hidden spending* from node u_2 to $\widehat{u}_3$. The dashed borders indicate node b's estimated membership set for epoch e that it listens to for the exact membership set (represented by the solid borders).

Specifically, starting from M and processing the transactions in $\mathcal{T}$ in increasing order of their epoch, it replaces node v with node v' for each transaction from v to v' that is valid with respect to the current set. We say $\mathcal{T}$ *constructs* a membership set M' if $\textsc{ApplyTxs}(\mathcal{M}_0, \mathcal{T}) = \mathsf{M}'$, and we denote $\mathsf{M}'.\mathsf{txs}$ as the transactions that construct M'.

These constructions are what allow for more efficient bootstrapping, since, often, there are far fewer membership-changing transactions than membership votes. We call the transactions that construct the decided membership sets, $\tilde{\mathcal{M}}(e).\mathsf{txs}$, *decided*. Naturally, the adversary may send additional *undecided* transactions to a booting node, which the node cannot immediately distinguish from decided ones. Furthermore, the adversary can confuse the node by *double-spending*, where the adversary creates conflicting transactions that transfer the same membership to different nodes (see Def. 4.4).

Definition 4.4 (Conflicting Transactions)**.** For the set of all transactions $\mathcal{T}$ received by some node, let $\mathcal{T}(\mathcal{D}) \subset \mathcal{T}$ be the set of transactions observed from double spenders (ln. 10). We denote $\mathcal{C} \subseteq \mathcal{T}(\mathcal{D})$ as the set of conflicting transactions from double spenders that were not decided as of the current epoch e, i.e.,

$$\mathcal{C} \triangleq \mathcal{T}(\mathcal{D}) \setminus \tilde{\mathcal{M}}(e).\mathsf{txs}.$$

See Fig. 4 for an example of a conflicting transaction from v_1 to $\widehat{v}_2$.

Therefore, during periods with double spending, our gadget follows along with the prior gadget to utilize votes to distinguish the true set of transactions that constructed the decided membership set. During periods without double spending, our gadget skips voting and directly constructs membership sets from the transactions received. We call this process *membership estimation* and illustrate the process in Fig. 4. The process is only an estimation since, besides double spending, the adversary may also include *hidden spending*, where it creates undecided transactions that do not

Algorithm 2 Reconfigurable atomic broadcast with sign-off for node v

during epoch e, **if** awake
  for every round in e except the last
    $\langle \mathsf{M}_e \rangle_e^v \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Sign}(\text{SK}_e, \mathsf{M}_e)$
    Send $(\mathsf{MVOTE}, \langle \mathsf{M}_e \rangle_e^v)$ to all
    Send transactions received to all
  same as lns. 2 to 7 in Alg. 1
upon boot up during epoch e
  same as lns. 9 to 14 in Alg. 1
  Receive $\mathsf{MVOTE}, \mathsf{LVOTE}$, and TX messages with valid forward-secure signatures
  $\mathcal{D} \leftarrow$ nodes that have signed transactions for conflicting receivers
  $\mathsf{E}_\ell \leftarrow \mathsf{M}_\ell$ // Initialize estimated membership to last known membership set
  for $e' \leftarrow \ell, \ell+1, ..., e-1$
    $\mathcal{T}_{e'} \leftarrow \mathsf{TX}$ messages signed during epoch e' from members in $\mathsf{E}_{e'}$
    if there exists a transaction from a double spender in $\mathcal{T}_{e'}$
      $\mathsf{votes} \leftarrow$ oldest LVOTE messages at or after epoch e' from members in $\mathsf{E}_{e'}$
      $\Lambda_{e'} \leftarrow$ most voted log up to epoch e' in votes
      $\mathsf{E}_{e'+1} \leftarrow \Lambda_{e'}.\textsc{newMembers}()$ // Get membership for next epoch from epoch e' log
    else // Otherwise directly apply $\mathcal{T}_{e'}$
      $\mathsf{E}_{e'+1} \leftarrow \textsc{ApplyTxs}(\mathsf{E}_{e'}, \mathcal{T}_{e'})$
  $\mathsf{votes} \leftarrow \mathsf{MVOTE}$ messages with signatures for epoch e from members in E_e
  $\mathsf{M}_e \leftarrow$ most voted membership set in votes
  $\text{SK}_e \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\text{SK}_\ell, e)$ // Evolve secret key to latest epoch
procedure $\textsc{SignOff}(v')$ // Sign-off procedure when called during an epoch e
  $\ell \leftarrow$ last epoch v was awake
  $\text{SK}_e \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Update}(\text{SK}_\ell, e)$
  $\langle (v, v') \rangle_e^v \leftarrow \mathcal{F}_{\text{FS}}.\textsc{Sign}(\text{SK}_e, (v, v'))$
  Delete SK_e
  Send $(\mathsf{TX}, \langle (v, v') \rangle_e^v)$ to all

conflict with decided transactions but still alter a booting node's view of the decided membership set. Crucially, however, hidden spending cannot remove honest nodes from an estimated membership set. We define estimated membership sets in Def. 4.5.

Definition 4.5 (Estimated Membership Set)**.** We say a set of nodes $\mathsf{E}_{e'}$ is an estimated membership set for an epoch $e' \leq e$ if it is constructed from a set of non-conflicting transactions that includes all decided transactions up to epoch e', i.e., such that

$$\tilde{\mathcal{M}}(e').\mathsf{txs} \subseteq \mathsf{E}_{e'}.\mathsf{txs} \subseteq \mathcal{T} \setminus \mathcal{C}.$$

We show that once a booting node finishes estimation up to the latest epoch, it can identify the latest decided membership set by following the set with the "heaviest votes" from members in the estimated set, since the estimated set is a superset of honest members that are a majority under the SR-HM condition (Def. 3.4). We give our bootstrapping gadget in Alg. 2.

Awake Execution. Awake nodes follow the same steps as the previous gadget by using Π_e^{DA} among the latest membership set and signing votes for the log outputted by Π_e^{DA} using the forward-

secure signature scheme. In addition, awake nodes send votes for the current membership and disseminate transactions at each round of the epoch except the last at lns. 2 to 5.

Sign-Off. The sign-off procedure is defined at ln. 23 given a replacement node v'. When a node v withdraws from the membership set during epoch e, it signs a transaction transferring its membership to node v' at ln. 26. Then, node v disposes of its secret key at ln. 27 and sends the transaction to at least one awake and honest node in the current membership set at ln. 28.

Bootstrapping. The bootstrapping process starts at ln. 7 when a node v boots up during an epoch e. When v wakes up for the first time, it follows the same initialization steps as the previous gadget. Then, v receives all valid votes and transactions signed with forward-secure signatures from the network at ln. 9. For every epoch e', v filters for all the transactions $\mathcal{T}_{e'}$ that were signed for epoch e' from its last estimated membership set (ln. 13). If there exist conflicting transactions in $\mathcal{T}_{e'}$, v follows the log with the "heaviest votes" (lns. 14 to 17) similar to the prior gadget.

In the absence of conflicting transactions in $\mathcal{T}_{e'}$, v applies all transactions in $\mathcal{T}_{e'}$ to its last estimated membership set (ln. 19). Once v has a membership set up to the latest epoch e, v identifies the decided membership set by following the set with the "heaviest votes" among its estimated members (lns. 20 and 21).

Efficiency Overhead. In addition to the LVOTE messages from Sec. 3.2, awake nodes forward at most $|\mathcal{T}_e|$ transactions and send at most one distinct MVOTE message in an epoch e. A booting node verifies the LVOTE and MVOTE messages and transactions for each missed epoch since it was last awake. For an epoch with no double spenders, the booting node performs only $|\mathcal{T}_e|$ verifications of transactions, where, typically in practice, $|\mathcal{T}_e| \ll |\tilde{\mathcal{M}}(e)|$.

Theorem 4.6. *Under the SR-HM condition (Def. 3.4), the protocol shown in Alg. 2 implements atomic broadcast in the DAR with sign-off model.*

Proof of Thm. 4.6. For every epoch e, the protocol is safe, live, and valid for awake and honest nodes due to the safety, liveness, and validity of Π_e^{DA}. It remains to show that an honest node v booting up during epoch e participates with the same membership set $\tilde{\mathcal{M}}(e)$ as the awake and honest nodes in that epoch.

We will prove by induction that the bootstrapping process defined by lns. 7 to 22 identifies *estimated membership sets* (Def. 4.5) from epoch 0 to e. For the base case, the initial membership set $\mathcal{M}_0$ is common knowledge and, by definition, is an estimated membership set. For the inductive step, assume that v has identified an estimated membership set $\mathsf{E}_{e'}$ for an epoch $e' < e$. We now show that the next membership set $\mathsf{E}_{e'+1}$ for v is an estimated membership set for epoch $e' + 1$.

Let $\mathcal{T}_{e'}$ be the set of valid transactions for epoch e' from nodes in $\mathsf{E}_{e'}$ as defined by ln. 13. Based on node v's view, there are two cases to consider: whether or not $\mathcal{T}_{e'}$ contains a transaction from a double spender.

Double Spender. In this case, the condition at ln. 14 is true, and node v proceeds into a voting-based procedure similar to our prior bootstrapping gadget in Sec. 3.2. Consider a log $\widehat{\Lambda}$ that is conflicting with the decided log $\Lambda_{e'}$ for epoch e'. Let c and $\widehat{c}$ be the total count of votes for $\Lambda_{e'}$ and $\widehat{\Lambda}$, respectively, in votes (ln. 15). Let s be the last round of epoch e', and t be the round node v is booting up in. We first observe that honest nodes that evolved or disposed of their keys must be in the estimated set. Notably, such nodes include all honest nodes, so we must have that

$c \geq |\mathcal{M}_s \cap \mathcal{H}(s,t)|$ since every honest member in $\mathcal{M}_s$ that was awake at some round at or after s signed and sent out a vote for a log that extends $\Lambda_{e'}$.

Any members in $\mathsf{E}_{e'}$ that are not in $\tilde{\mathcal{M}}(e')$ are constructed from undecided transactions in $\mathsf{E}_{e'}.\mathsf{txs}$. These members cannot be more than the adversarial and simulatable nodes in $\tilde{\mathcal{M}}(e')$. Then, we must have that the number of votes for $\widehat{\Lambda}$ is at most $\widehat{c} \leq |\mathcal{M}_s \cap (\mathcal{A}_s \cup \mathcal{S}_{\overline{\mathcal{W}}}(s;t))|$. Since $\mathcal{A}_s \subseteq \mathcal{S}_{\overline{\mathcal{W}}}(s;t)$, by the SR-HM condition (Def. 3.4) in the sign-off model, we then have

$$\widehat{c} \leq |\mathcal{M}_s \cap \mathcal{S}_{\overline{\mathcal{W}}}(s;t)| < |\mathcal{M}_s \cap \mathcal{H}(s,t)| \leq c.$$

Thus, it must be that $\Lambda_{e'}$ is the most voted log, which v uses to retrieve the next decided membership set $\tilde{\mathcal{M}}(e'+1)$.

No Double Spender. In this case, $\mathsf{E}_{e'+1}$ is directly constructed from $\mathcal{T}_{e'}$ over $\mathsf{E}_{e'}$ at ln. 19. We show that $\mathcal{T}_{e'}$ must contain all the decided transactions from epoch e'. Consider a sender q of a decided transaction in epoch e', who must be a member in $\tilde{\mathcal{M}}(e')$. The sender q must also remain a member in $\mathsf{E}_{e'}$ since q is not a double spender, implying no transaction of q exists in $\mathsf{E}_{e'}.\mathsf{txs}$. Thus, $\mathsf{E}_{e'+1}$ is an estimated set since it is constructed from all the decided transactions up to epoch $e'+1$ and does not contain any conflicting transactions.

By induction, the booting node v identifies estimated membership sets from epoch 0 to epoch e. Let E_e be the estimated membership set for epoch e. We now argue that v identifies the decided membership set $\tilde{\mathcal{M}}(e)$. Let $\widehat{\mathsf{M}} \neq \tilde{\mathcal{M}}(e)$ be an undecided membership set for epoch e, and let c and $\widehat{c}$ be the number of votes received for $\tilde{\mathcal{M}}(e)$ and $\widehat{\mathsf{M}}$, respectively, at ln. 20. We must have that E_e includes all honest members in the latest round t in which v is booting up since the adversary cannot forge their signatures and create undecided transactions for them, i.e., $\mathsf{E}_e \supseteq \mathcal{M}_t \setminus \mathcal{A}_t$. Thus, we must have that $c \geq |\mathcal{M}_t \cap \mathcal{H}_t|$. Furthermore, since $|\mathsf{E}_e| = |\mathcal{M}_t| = |\mathcal{M}_t \setminus \mathcal{A}_t| + |\mathcal{M}_t \cap \mathcal{A}_t|$, it must be that E_e has at most $|\mathcal{M}_t \cap \mathcal{A}_t|$ adversarial nodes that can sign votes, implying $\widehat{c} \leq |\mathcal{M}_t \cap \mathcal{A}_t|$. Therefore, by the SR-HM condition (Def. 3.4), we must have that

$$\widehat{c} \leq |\mathcal{M}_t \cap \mathcal{A}_t| < |\mathcal{M}_t \cap \mathcal{H}_t| \leq c.$$

Thus, $\tilde{\mathcal{M}}(e)$ is the most voted membership set, and node v stores it in M_e at ln. 21. Node v can then participate under the same configuration of Π^{DA} as the latest awake and honest nodes. Therefore, we have that Alg. 2 is a safe, live, and valid atomic broadcast protocol under the DAR with sign-off model. □

5 Related Work

Reconfiguration has been well studied, first for crash fault-tolerant consensus [LAB+06, LMZ09, SRMJ12] and later for Byzantine consensus [BSA14, DZ22]. Most of these early works focus on the partially synchronous and (hence inevitably) statically available setting. The work of [AKSW22] seems to operate in a setting similar to DAR, but they consider only crash faults without recovery and assume a known threshold over the number of crash faults (i.e., static availability).

Algorand [CM19] studies consensus with *player replaceability*, where each step of the protocol is executed by a different committee. Algorand's committees are subsampled from the membership set via VRF-based sortition [MRV99]. Algorand crucially relies on a fixed threshold of awake and honest nodes, so it does not tolerate adversarially controlled sleepiness and does not target dynamic availability. As a result, Algorand can change its membership set with standard reconfiguration techniques under static availability. YOSO [GHK+21], Fluid MPC [CGG+21], and Layered MPC [DDG+23]

extend player replaceability to MPC. Like Algorand, they do not target dynamic availability. They also do not target reconfiguration, as their membership set is either fixed or given by an external mechanism.

Reconfigurable consensus has seen a resurgence of interest due to the popularity of proof-of-stake (PoS) cryptocurrencies. PoS leverages the allocation of currency in the system to signify the voting power of users. PoS consensus is synonymous with reconfigurable consensus since the distribution of stake defines membership for participation.[4] There are many PoS proposals [DPS19, KRDO17, DGKR18, GHM+17, BGK+18, BHK+20, DZ22] used in practice today. Among those, only a few [DPS19, BGK+18, BHK+20] aim to support dynamic availability, and we have already compared them thoroughly in Sec. 1. We further note that the Ouroboros Genesis protocol [BGK+18] may work in the DAR model (in which asleep nodes do not perform key evolutions), under a condition that is similar in spirit to, but stronger than, our SR-HM condition.

Other recent works study dynamically available consensus in the non-reconfigurable setting, most notably sleepy consensus [PS17, MR22, MMR23, DNTT24, DSTZ25, ET25, GL23, FLPA26] and PoSAT [DKT21].

Lewis-Pye and Roughgarden [LPR24] provide a systematic overview and taxonomy of various models regarding availability and reconfiguration. In their taxonomy, the "quasi-permissionless" setting corresponds to reconfiguration with static availability (where they place Algorand, consistent with our discussion earlier), while their dynamically available setting corresponds to the DAR model. We do not call PoS "permissionless," because newcomer nodes in PoS need to acquire stake from current nodes to join the system. Although unlikely, one could imagine a scenario where initial stakeholders refuse to transfer their stake (at all or to certain individuals). In contrast, PoW protocols are considered permissionless since possession of computational resources is the only barrier to participation.

A notable result that is pertinent to our work is that of [LPR24, Theorem 10.1], which states that any protocol in the reconfiguration setting that only uses time-malleable cryptography (e.g., not forward-secure) is vulnerable to long-range attacks. They do not provide any positive result in the DAR model with Byzantine faults.

Forward-secure signatures were first proposed by Anderson [And97] and formalized by Bellare and Miner [BM99] as protection against the *key exposure problem*, where security of a digital signature scheme is lost when the underlying secret (signing) key is compromised. The recent Pixel multi-signature scheme [DGNW20], adopted by the Algorand blockchain [CM19], achieves forward security and practical efficiency.

6 Conclusion

We studied the fundamental limits of achieving consensus under the plain setting of Dynamic Availability and Reconfiguration (DAR). We showed that in the absence of any extra features added to the plain DAR model, consensus is only achievable under the *simulation-resistant honest majority* condition (Def. 3.4). We formalized this barrier with a tight lower bound and complemented it with a generic bootstrapping gadget that safely extends any protocol for the dynamically available, non-reconfigurable setting (i.e., the sleepy model) to the DAR setting under SR-HM. Our matching bounds characterize precisely under what circumstances consensus is feasible in the DAR model without extra features. We then introduced a variant of the DAR model with a sign-off requirement—typically satisfied in PoS systems. In the DAR with sign-off model, we can leverage two additional

[4] In practice, PoS membership is often weighted by stake. This difference mainly pertains to efficiency and not fundamental feasibility.

facts: honest withdrawals are unforgeable (even for asleep nodes that are later corrupted) due to key disposal, and double-spending is detectable due to signatures and network synchrony. In that setting, we weaken the adversary's ability to simulate. We proved SR-HM is sufficient for consensus in this setting, and presented a second bootstrapping gadget that has a more efficient good-case path of *membership estimation* with a fallback to voting-based resolution.

Acknowledgments

This work is funded in part by the National Science Foundation award #2143058.

References

[ADKS22] Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In *DISC*, volume 246 of *LIPIcs*, pages 38:1–38:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[AKSW22] Hagit Attiya, Sweta Kumari, Archit Somani, and Jennifer L. Welch. Store-collect in the presence of continuous churn with application to snapshots and lattice agreement. *Inf. Comput.*, 285(Part):104869, 2022.

[And97] Ross Anderson. Two remarks on public key cryptology. *Invited Lecture, ACM CCS'97*, 1997. `https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-549.pdf`.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS*, pages 913–930. ACM, 2018.

[BHK+20] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining GHOST and Casper, 2020.

[BM99] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, volume 1666 of *LNCS*, pages 431–448. Springer, 1999.

[BSA14] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *DSN*, pages 355–362. IEEE Computer Society, 2014.

[CASD95] Flaviu Cristian, Houtan Aghili, H. Raymond Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. *Inf. Comput.*, 118(1):158–179, 1995.

[CGG+21] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In *CRYPTO (2)*, volume 12826 of *LNCS*, pages 94–123. Springer, 2021.

[CL99] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *OSDI*, pages 173–186. USENIX Association, 1999.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[CMSK07] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *SOSP*, pages 189–204. ACM, 2007.

[DDG+23] Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In *CRYPTO (1)*, volume 14081 of *LNCS*, pages 360–392. Springer, 2023.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)*, volume 10821 of *LNCS*, pages 66–98. Springer, 2018.

[DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In *USENIX Security Symposium*, pages 2093–2110. USENIX Association, 2020.

[DKT21] Soubhik Deb, Sreeram Kannan, and David Tse. PoSAT: Proof-of-work availability and unpredictability, without the work. In *Financial Cryptography (2)*, volume 12675 of *LNCS*, pages 104–128. Springer, 2021.

[DNTT24] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on Ethereum?! In *FC (1)*, volume 14744 of *LNCS*, pages 3–23. Springer, 2024.

[DPP19] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

[DPS19] Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography*, volume 11598 of *LNCS*, pages 23–41. Springer, 2019.

[DSTZ25] Francesco D'Amato, Roberto Saltini, Thanh-Hai Tran, and Luca Zanolini. TOB-SVD: Total-order broadcast with single-vote decisions in the sleepy model. In *ICDCS*, pages 1033–1043. IEEE, 2025.

[DZ22] Sisi Duan and Haibin Zhang. Foundations of dynamic BFT. In *SP*, pages 1317–1334. IEEE, 2022.

[ET25] Yuval Efron and Ertem Nusret Tas. Dynamically available common subset. Cryptology ePrint Archive, Paper 2025/016, 2025.

[FLPA26] Ali Farahbakhsh, Giuliano Losa, Youer Pu, and Lorenzo Alvisi. Fast deterministically safe proof-of-work consensus. In *SP*, pages 2133–2151. IEEE, 2026.

[GDCL22] Mingyuan Gao, Hung Dang, Ee-Chien Chang, and Jialin Li. Mixed fault tolerance protocols with trusted execution environment, 2022.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO (2)*, volume 12826 of *LNCS*, pages 64–93. Springer, 2021.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68. ACM, 2017.

[GL23] Eli Gafni and Giuliano Losa. Brief announcement: Byzantine consensus under dynamic participation with a well-behaved majority. In *DISC*, volume 281 of *LIPIcs*, pages 41:1–41:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[GP92] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In *WDAG*, volume 647 of *LNCS*, pages 153–165. Springer, 1992.

[HM84] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *PODC*, pages 50–61. ACM, 1984.

[IR01] Gene Itkis and Leonid Reyzin. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, volume 2139 of *LNCS*, pages 332–354. Springer, 2001.

[JM14] Leander Jehl and Hein Meling. Asynchronous reconfiguration for Paxos state machines. In *ICDCN*, volume 8314 of *LNCS*, pages 119–133. Springer, 2014.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO (1)*, volume 10401 of *LNCS*, pages 357–388. Springer, 2017.

[LAB+06] Jacob R. Lorch, Atul Adya, William J. Bolosky, Ronnie Chaiken, John R. Douceur, and Jon Howell. The SMART way to migrate replicated stateful services. In *EuroSys*, pages 103–115. ACM, 2006.

[LMZ09] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical Paxos and primary-backup replication. In *PODC*, pages 312–313. ACM, 2009.

[LMZ10] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *SIGACT News*, 41(1):63–73, 2010.

[LPR24] Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus, 2024.

[LR23] Andrew Lewis-Pye and Tim Roughgarden. Byzantine generals in the permissionless setting. In *FC (1)*, volume 13950 of *LNCS*, pages 21–37. Springer, 2023.

[MMR23] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy BFT. In *CCS*, pages 490–503. ACM, 2023.

[MR22] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *CCS*, pages 2295–2308. ACM, 2022.

[MRV99] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130. IEEE Computer Society, 1999.

[Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. `https://bitcoin.org/bitcoin.pdf`.

[NTT21] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *SP*, pages 446–465. IEEE, 2021.

[OO14] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX ATC*, pages 305–319. USENIX Association, 2014.

[PS17] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT (2)*, volume 10625 of *LNCS*, pages 380–409. Springer, 2017.

[SRMJ12] Alexander Shraer, Benjamin C. Reed, Dahlia Malkhi, and Flavio Paiva Junqueira. Dynamic reconfiguration of primary/backup clusters. In *USENIX ATC*, pages 425–437. USENIX Association, 2012.

[WTW+24] Jianghong Wei, Guohua Tian, Ding Wang, Fuchun Guo, Willy Susilo, and Xiaofeng Chen. Pixel+ and Pixel++: Compact and efficient forward-secure multi-signatures for PoS blockchain consensus. In *USENIX Security Symposium*. USENIX Association, 2024.