Thunderella: Blockchains with Optimistic Instant Confirmation

Rafael Pass Elaine Shi

Abstract

State machine replication, or "consensus", is a central abstraction for distributed systems where a set of nodes seek to agree on an ever-growing, linearly-ordered log. In this paper, we propose a practical new paradigm called Thunderella for achieving state machine replication by combining a fast, asynchronous path with a (slow) synchronous "fall-back" path (which only gets executed if something goes wrong); as a consequence, we get *simple* state machine replications that essentially are as robust as the best synchronous protocols, yet "optimistically" (if a super majority of the players are honest), the protocol "instantly" confirms transactions.

We provide instantiations of this paradigm in both permissionless (using proof-of-work) and permissioned settings. Most notably, this yields a new blockchain protocol (for the permissionless setting) that remains resilient assuming only that a majority of the computing power is controlled by honest players, yet *optimistically*—if 3/4 of the computing power is controlled by honest players, and a special player called the "accelerator", is honest—transactions are confirmed as fast as the actual message delay in the network. We additionally show the 3/4 optimistic bound is tight for protocols that are resilient assuming only an honest majority.

Contents

Chapter 1

Introduction

State machine replication, also referred to as atomic broadcast, is a core distributed systems abstraction that has been investigated for three decades. In a state machine replication protocol, a set of servers seek to agree on an ever-growing, *linearly-ordered log*, such that two important properties are satisfied: 1) *consistency*, i.e., all servers must have the same view of the log; and 2) *liveness*, i.e., whenever a client submits a transaction, the transaction is incorporated quickly into the log. In this paper, we will also refer to state machine replication as *consensus* for short[1].

State machine replication is a fundamental building block for replicated databases. For more than a decade, companies such as Google and Facebook have deployed Paxos-style protocols [6, 28, 34] to replicate a significant part of their computing infrastructure. These classical deployment scenarios are typically relatively small scale, with fast local-area networking, where crash (rather than byzantine) faults are usually of concern.

Fuelled by decentralized cryptocurrencies, recently the community has been excited about large-scale applications of distributed consensus. Two deployment scenarios are of interest: 1) the *permissionless* setting where anyone can join freely (e.g., decentralized cryptocurrencies); and 2) the *permissioned* setting where only approved participants may join (e.g., a consortium blockchain where multiple banks collaborate to build a distributed ledger). Regardless of which setting, the typical deployment would involve a large number of nodes (e.g., thousands or more) controlled by mutually distrustful individuals and organizations.

Roughly speaking, two broad classes of protocols have been considered for the large-scale setting, each with their own set of deficiencies:

- First, *classical-style protocols* such as PBFT [13] and Byzantine-Paxos [34] confirm transactions quickly in the normal case; but these protocols are notoriously complicated, making implementation, reconfiguration, and maintenance relatively difficult especially in a large-scale setting. Further, these protocols achieve "fast confirmation" by adopting the asynchronous (or partially synchronous) model, and thus inherently they can tolerate at most $\frac{1}{3}$ corruptions [19, 41].

- Second, *blockchain-style* protocols, represented by Nakamoto's original blockchain [23, 38, 39], are a new breakthrough in distributed consensus: these protocols are conceptually simple and tolerate *minority corruptions*. Moreover, it is has been shown how to remove the expensive proof-of-work from blockchain-style consensus [15, 30, 43] thus solving the energy waste problem. Further, not only has blockchains' robustness been empirically proven, earlier works [15, 43] have

[1] Although the term "consensus" has been used in the distributed systems literature to mean other related abstractions such as single-shot consensus; in this paper, we use "consensus" to specifically refer to "state machine replication".

also shown mathematically that blockchain-style consensus indeed achieves certain robustness properties in the presence of sporadic participation and node churn that none of the classical-style protocols can attain! Unfortunately known blockchain-style protocols suffer from slow transaction confirmation, e.g., Bitcoin's Nakamoto consensus has a 10-minute block interval and it takes several blocks to confirm a transaction with sufficient confidence. Earlier works that mathematically analyze blockchain-style consensus [39, 43] have pointed out that such slowness is inherent for blockchain-style protocols since the expected block interval must be set to be sufficiently large for the protocol to retain security.

A natural question that arises is whether there is some way to *simultaneously* reap the benefit of both of these "worlds". Unfortunately, a negative answer was presented by earlier works [41–43] which showed that a natural notion of fast transaction confirmation called "responsiveness" is unattainable against $\frac{1}{3}$ (even static) corruptions in classical or permissionless models. In this paper we consider a new notion called *optimistic responsiveness* that allows us "circumvent" this lower bound such that we can achieve responsiveness most of the time in practice and yet tolerate up to minority corruptions in the worst-case. In our approach, in the *optimistic case* (when e.g., a super majority is honest), we enjoy the fast nature of asynchronous protocols; and yet we retain the resilience of synchronous (e.g., blockchain) protocols as well as their robustness properties (e.g., support for sporadic participation). More precisely, we show how to combine a fast and simple "asynchronous path"—which guarantees consistency but not liveness—with a (slow) synchronous "fall-back" path which only gets executed if something goes wrong.

1.1 The Thunderella Paradigm

To characterize what we mean by "fast" or "instant confirmation", we adopt the same notion of *responsiveness* as proposed in the work by Attiya, Dwork, Lynch, and Stockmyer [2] and later adopted by others [27, 41]. A consensus protocol is said to be responsive iff any transaction input to an honest node is confirmed in time that depends only on the *actual network delay*, but not on any a-priori known *upper bound on the network delay*. Henceforth in this paper, we use δ to denote the actual network delay and use Δ to denote an a-priori known upper bound of the network's delay where Δ is possibly provided as input to the protocol.

As shown in [41], achieving responsiveness requires us to assume that 2/3 of the players are honest. (Additionally, all known protocols that are responsive are very complicated, and thus hard to implement.)

Towards overcoming this issue, we here instead consider a notion of **optimistic responsiveness**—where responsiveness is only required to hold whenever some "goodness conditions" are satisfies. More precisely, we consider two sets of conditions:

- worst-case conditions (denoted W) under which the protocol provides worst-case guarantees including consistency and "slow" confirmation (e.g., W = majority honest).
- optimistic-case conditions (denoted $O \subseteq W$) under which the protocol additionally provides responsive confirmation (e.g., O ="more than $\frac{3}{4}$ are honest and online, and some designated player (the "leader") is honest").

Our main result is a paradigm for taking any blockchain protocol (permissioned or permissionless) that satisfies consistency and liveness under conditions W, and transform it into a new protocol that satisfies consistency and liveness under "essentially" the same conditions W (and in many cases, actually the same conditions W), and additionally satisfies optimistic responsiveness under condition O.

The idea in a nutshell To explain our approach, consider first the following simple protocol:

- We have a designated entity: the leader, or "accelerator".
- Transactions are sent to the leader; the leader signs the transaction (with an increasing sequence number), and sends out the signed transaction to a "committee" of players.
- The committee members "ack" all leader-signed transactions, but at most one per sequence number.
- If a transaction has received more than 3/4 of the committees signatures—we refer to such a transaction as being notarized. Participants, can *directly output* their longest sequence of consecutive (in terms of their sequence numbers) notarized transactions—all those transactions are confirmed.

It is not hard to see that this protocol is consistent under condition W' = "1/2 the committee is honest"); additionally, it satisfies liveness with optimistic responsiveness under condition O = "leader is honest, and 3/4 of the committee is honest". In fact, under these optimistic condition, we only need 2 communication rounds to confirm a transaction! This approach is extremely practical and indeed this protocol is often used in practice—for instance `chain.com` use something very similar as their permissioned blockchain (and manage to handle a very high volume of transactions with fast confirmations).

The problem with this approach, however, is that the protocol does not satisfy liveness (even "slow" liveness) under condition W'. If the leader is cheating (or is simply taken down from the network), the protocol halts. (Indeed, in this case `chain.com` resorts to manually fixing the issue.)

To overcome this problem, we leverage the underlying (slow) blockchain protocol, which satisfies both consistency and liveness under W = "honest majority of players". Roughly speaking, if players notice that transactions are not getting confirmed by the leader/committee, some "evidence" of this is sent to the underlying blockchain. We then enter a "cool-down" period, where committee members stop signing messages from the leader, yet we allow players to broadcast any notarized transactions they have seen so far. The length of the cool-down period is counted in blocks on the underlying blockchain (say κ blocks where κ is a security parameter). Finally, after the cool-down period ends, we can safely enter a "slow period" where transactions only get confirmed in the underlying blockchain. We can next use the blockchain to switch out the leader (if needed) and begin a new epoch of the optimistic protocol.

Let us point out the reason for having a cool-down period: without it, players may disagree on the set of transactions that have been confirmed before entering the "slow mode", and thus may end up with inconsistent views. The cool-down period enables honest players to post all notarized transactions they have seen to the (slow) underlying blockchain, and thus (slowly) reach consistency of this set of transactions; once we have reached this consistent view (at the end of the cool-down), we can finally fully switch over to confirming new transactions on the blockchain.

Collecting evidence of cheating It only remains to explain how to collect evidence that the leader (and/or committee) is cheating or is simply "unavailable". This turns out to also be simple: if a player notices that his transaction is not getting confirmed by the leader or committee, he can send the transaction to the underlying blockchain. The leader is additionally instructed to confirm all transactions it sees on the blockchain.

Now, if players see some transaction on the blockchain, that has not gotten notarized within a sufficiently long amount of time—counted in blocks in the underlying blockchains (say within n

blocks)—they know that the leader/committee must be cheating/unavailable, and thus should enter the cool-down period. (Note that as long as the leader can confirm transactions before n blocks are created on the underlying blockchain, he cannot be "falsely accused"; and, by the security of the underlying blockchains those blocks cannot be created too fast).

Selecting the committee So far we have constructed a protocol that satisfies consistency and liveness under conditions $W \cap W'$ (i.e., assuming an honest majority of players, and an honest majority in the committee), and additionally satisfies liveness with optimistic responsiveness under condition O. The question now is how to select the committee. We consider two different approaches:

- **Using all players as the committee:** In a permissioned setting, the simplest approach is to simply use all players as the committee. In this case, $W' = W$ and thus, we trivially have resilience under W. A variant of this approach with improved communication complexity is to subsample a committee among the set of players (for instance, using the approach in [15] which additionally requires a random oracle), and change committees on a regular basis (to ensure adaptive security)—the resulting protocol, however, will only be secure if corruptions are "slow" (to ensure the attacker does not have time to corrupt the whole committee before it gets switched out.) If sub-sampling is instead done "secretly" using a VRF and a random oracle (as in [36]), we can also ensure that the resulting protocol is adaptively secure *in a model with erasures*, even with "instantaneous corruption".

  We mention that these approaches may also be used in the permissionless setting if Thunderella is used to construct a crypto currency: then we can use (potentially a sub-sample of) recent "stakeholders" to form a committee.

- **Using "recent miners" as the committee:** A different approach that works in both the permissioned and permissionless setting is to select the committee as the miners of recent blocks (as was done in [41]). We note, however, that to rely on this approach, we need to ensure that the underlying is blockchain is "fair" [40] in the sense that the fraction of honestly mined blocks is close to the fraction of honest players. This is not the case for Nakamoto's original blockchain (see e.g., [21]), but as shown in [40], any blockchain can be turned into a fair one. If we use this approach, the resulting protocol will now be consistent and live under simply the condition W (i.e., honest majority), yet also satisfy optimistic liveness under condition O. (Again, this only gives security under adaptive corruption where corruption is "slow", so the set of recent miners changes sufficiently fast before they can all be corrupted.)

Permissionless Thunderella For instance, if we apply the second approach (of selecting the committee as the recent miners) to Nakamoto's proof-of-work based blockchain, we get the following theorem:

Theorem 1 (Thunderella for permissionless environments, informal)**.** *Assume a proof-of-work random oracle. Then, there exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a permissionless environment as long as the adversary wields no more than $\frac{1}{2} - \epsilon$ the total online computation power in every round where ϵ is an arbitrarily small constant, and moreover it takes a short while for the adversary to adaptively corrupt nodes. Moreover, if more than $\frac{3}{4}$ of the online computation power is honest and online, then the protocol achieves responsiveness (after a short non-responsive warmup period) in any "epoch" in which the leader is honest and online.*

Permissioned Thunderella Similar theorems can be shown for permissioned environments (in e.g., the "sleepy model" of [43], or even just in the "classic" model of Dolev-Strong [18].)

The classical mode is essentially the standard synchronous model adopted by the existing distributed systems and cryptography literature. In this model, all nodes are spawned upfront, and their identities and public keys are provided to the protocol as input; further, crashed nodes are treated as faulty and count towards the corruption budget. In a classical, synchronous network, we show that the classical Dolev-Strong byzantine agreement protocol [18] can be extended to implement Thunderella's underlying "blockchain". In this case, our Thunderella paradigm (where use the first approach to instantiate the committee) gives rise to the following informal theorem:

Theorem 2 (Thunderella for permissioned, classical environments (informal))**.** *Assume the existence of a PKI and one-way functions. There exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a classical environment under any $f < n$ number of fully adaptive, byzantine corruptions where n denotes the total number of nodes; moreover, the protocol achieves responsiveness as long as the leader is honest and moreover $\lfloor \frac{n+f}{2} + 1 \rfloor$ nodes are honest.*

The "sleepy" model was recently proposed by Pass and Shi [43] to capture the requirements arising from "sporadic participation" in large-scale, permissioned consensus. Specifically, the sleepy model was in fact inspired by permissionless decentralized cryptocurrencies such as Bitcoin, where nodes may come and go frequently during the protocol, and the protocol should nonetheless guarantee consistency and liveness even for players that join late, and for players who might have had a short outage and woke up later to rejoin the protocol.

The sleepy model is "permissioned" in nature in that the set of approved protocol participants and their public keys are a-priori known and provided to the protocol as input. However, unlike the classical setting, 1) nodes are allowed to be non-participating (i.e., *sleeping*); 2) sleeping nodes are not treated as faulty; and 3) the protocol may not know in advance how many players will actually show up. In comparison, in a classical setting, non-participating nodes are regarded as having crashed and count towards the corruption budget; and moreover a classical protocol need not guarantee consistency and liveness for nodes that have crashed but wake up later to rejoin.

In such a sleepy model, Pass and Shi [43] show that roughly speaking, we can achieve consensus when the majority of *online* (i.e., non-sleeping) nodes are honest (interestingly, unlike the classical synchronous model, [43] also prove that no state machine replication protocol can tolerate more than $\frac{1}{2}$ corruption (among online nodes).

Our Thunderella paradigm (again using the first approach for selecting the committee) can be instantiated in the sleepy model using the sleepy consensus protocol as the underlying blockchain. This gives rise to the following informal theorem in a sleepy environment (where we assume that the adversary can adaptively put honest nodes to sleep).

Theorem 3 (Thunderella for permissioned, sleepy environments (informal))**.** *Assume the existence of a PKI, enhanced trapdoor permutations, and a common reference string (CRS). There exists a state machine replication protocol that achieves consistency and (non-responsive) liveness in a sleepy environment with static corruptions, as long as $\frac{1}{2} - \epsilon$ of the online nodes are honest in every round for any arbitrarily small constant ϵ; moreover, if more than $\frac{3}{4}$ fraction of nodes are honest and online, the protocol achieves responsiveness (after a short non-responsive warmup period) in any epoch in which leader is honest and online.*

In fact, the above theorem also extends to adaptive corruptions *with erasures* using the adap-

tively secure version of sleepy consensus [43][2] as Thunderella's underlying blockchain, assuming the existence of a VRF and a random oracle (using the approach from [36]).

Lower bounds on the optimistic honest threshold We additionally prove that our optimistic bound of 3/4 is tight: no protocol that is (worst-case) resilient for simply an honest majority, can be optimistically responsive when more than 1/4 of the player can be corrupted.

Practical Considerations: Instant Confirmation and Scalability The *low latency* and *poor scalability* of Nakamoto's blockchain protocol are typically viewed as the main bottlenecks for Bitcoin as well as other cryptocurrencies.

Our paradigm provides a very practical and simple approach for overcoming these issue. The Thunderella paradigm shows how to build on top of currently running blockchains, to enable "optimistic instant confirmation" of transactions. Additionally, note that in our protocol, players only need to send transactions to the leader, who in turn lead the committee to confirm the transaction. Most notably, the underlying blockchain is essentially only used when something goes wrong, and blocks need not be distributed to the whole network before getting confirmed; thus, Thunderella also solves the scalability issue with Nakamoto's blockchain protocol. Of course, both of these guarantees are only "optimistic"— but arguably, under normal circumstances one would expect 3/4 of the players to act honestly, and the leader could be incentivized (paid) to perform its job (and if it doesn't, will be kicked out). Thus, we believe our approach is a practically viable approach for circumventing the main bottlenecks of today's cryptocurrencies.

Comparison. At the surface, our idea is reminiscient of classical-style protocols such as PBFT and Byzantine-Paxos. In particular, protocols like PBFT also have a very simple normal path that consists of $O(1)$ rounds of voting. However, when the normal path gets stuck, PBFT-style protocols fall back to a "view change" mechanism that is also responsive—and thus these protocols tolerate only $\frac{1}{3}$ corruptions in the worst-case, and are invariably complex due to the need to handle asynchrony. (Furthermore, this approach is not amenable for protocols in the permissionless setting). Our key insight is to instead fall back to a synchronous path in the worst case, thus allowing us to circumvent the $\frac{1}{3}$ lower bound for partial synchrony and yet still be responsive in practice most of the time. Moreover, since our protocol is fundamentally synchronous, we benefit from the simplicity and robustness enjoyed by synchronous protocols (e.g., blockchains).

Interestingly, Thunderella is also a constant factor faster in the fast path than most PBFT- or Paxos-style protocols. PBFT-style protocols typically require multiple rounds of voting even in the normal path (*c.f.* Thunderella has exactly one) — and the latter rounds are necessary to prepare for the possibility of a view change. Although it is possible to compress the normal path to a single round of voting, this is typically achieved either by sacrificing resilience (e.g., tolerating only $\frac{1}{5}$ corruptions) [45] or by adding yet another optimistic layer on top of the normal path — thus further complicating the already complex protocol [33].

Roadmap In this extended abstract, we simply provide a description and proof of the general Thunderella paradigm (informally described above) assuming the existence of a fixed committee, a majority of which is honest. We defer the formal treatment of how to select the committee to the supplemental material (although we described it informally above).

[2]The paper has multiple adaptively secure versions; here we rely on the one that achieves adaptive security with erasures in the random oracle model (as this protocol has better parameters than the one which satisfies adaptive security without a random oracle).

1.2 Related Work

State machine replication: classical and blockchain-style approaches. State machine replication or atomic broadcast (referred to as consensus for short in this paper) is a central abstraction of distributed systems, and has been extensively investigated and widely adopted in real-world systems. Roughly speaking, there are two, technically speaking, fundamentally different approaches towards realizing state machine replication, classical-style consensus [13,18,19,29,34,35], and blockchain-style consensus [15, 23, 30, 38, 39, 43]. For a while, it has been vaguely understood by the community that blockchain-style protocols and classical ones achieve different properties — but the community has only recently begun to formally understand and articulate these differences.

The recent work by Pass and Shi [43] point out one fundamental difference between classical style and blockchain-style consensus. Most classical protocols [13,18,19,29,34,35], synchronous and asynchronous ones alike, rely on nodes having collected sufficiently many votes to make progress; thus these protocols would fail in a model where *participation is sporadic* and the exact number of players that do show up cannot be predicted upfront. More specifically, classical models of consensus would pessimistically treat nodes that do not show up as faulty (also referred to as crash fault); and if too many nodes do not show up, the protocol fails to make progress. In comparison, blockchain-style protocols can make progress regardless of how many players actually show up. Moreover, blockchain-style consensus has also been shown to be secure in a setting where the number of players can vary over time [22].

Classical deployments of consensus protocols are typically in a relatively small-scale and permissioned setting. Consensus in the permissionless setting was first empirically demonstrated to be possible due to Bitcoin's ingenious Nakamoto blockchain [38]. While the original Nakamoto blockchain relies on proofs-of-work to solve the Sybil attack in the permissionless setting, other proposals have been suggested since then for securely establishing identities in a permissionless setting — for example, proof-of-stake [3,4,8,14,15,30,31,36,44] is a most-oft cited approach where the stake-holders of a cryptocurrency system are responsible for voting on transactions. Recent works [36] have also explored adopting classical style consensus in a permissionless setting where approaches such as proof-of-stake can be used to establish identities.

Other closely related works. Our work is also reminisient of recent works that combine classical consensus and blockchains [16, 32, 41] although these works are of a different nature as we explain below. Among these works, Hybrid Consensus [41] is the only known formally correct approach, and moreover the only known approach that achieves *responsiveness*. From a theoretical perspective, our results are incomparable to Hybrid Consensus: we tolerate up to $\frac{1}{2}$ corruption in the worst-case and offer responsiveness only in the optimistic case but not in the worst case; in comparison, Hybrid Consensus achieves responsiveness even in the worst case — but in exchange, their protocol can only tolerate up to $\frac{1}{3}$ corruption, and this turns out to be inherent for any worst-case responsive protocol even when assuming proof-of-work [19, 41]. From a practical perspective, Thunderella is more likely to be the protocol of choice in a real-world implementation partly due to its simplicity — in comparison, Hybrid Consensus requires a full-fledged classical protocol such as PBFT and Byzantine Paxos as a subroutine, and thus inherits the complexity of these protocols.

A line of research [12,17,20,25,26,37] has investigated Byzantine agreement protocols capable of early-stopping when conditions are more benign than the worst-case faulty pattern: e.g., the actual number of faulty nodes turns out to be smaller than the worst-case resilience bound. However, these works are of a different nature than ours as we explain below. First, these earlier works focus on stopping in a fewer number of synchronous rounds, and it is not part of their goal to achieve *responsiveness*. Second, although some known lower bounds [17] show that the number of actual

rounds must be proportional to the actual number of faulty processors — note that these lower bounds work only for deterministic protocols, and thus they are not applicable in our setting.

Finally, the idea of combining asynchrony and synchrony was described in earlier works [5]; other works have also proposed frameworks for composing multiple BFT protocols [24]. However, to the best of our knowledge, none of the earlier works combined a synchronous fallback path and an asynchronous optimistic path in the manner that we do, allowing us to tolerate more than $\frac{1}{3}$ corruptions in the worst-case while still be responsive most of the time in practice.

Relevance to Ethereum's current efforts. Ethereum [47], the second largest decentralized cryptocurrency immediately following Bitcoin [38], has investigated significant resources in developing the next-generation proof-of-stake consensus protocol [8]. Instead of immediately dispensing with proof-of-work, Ethereum seems to have devised a two-stage agenda, where, in an initial transitional stage, voting by stake-holders and proof-of-work will co-exist — see the recent online blog-post by Vitalik Buterin [7]. Specifically, stake-holders will vote on transactions or blocks atop their existing proof-of-work blockchain. Vitalik Buterin's blog-post then proposes a set of "minimal slashing conditions" [7] to penalize voters that behave maliciously (e.g., those who cast equivocating votes). However, it is not Ethereum's goal to achieve optimistic instant confirmation [1].

Others have also tried to achieve instant finality in blockchains [46] — however, to date there is no provably secure approach, and some folklore approaches even appear flawed.

Chapter 2

Definitions and Preliminaries

Since we would like to capture what the Thunderella paradigm can achieve in various settings including permissionless and permissioned, we adopt the following modeling approach. We first describe a general execution model that is capable of expressing an unconstrained permissionless setting. We then express permissioned settings including classical and sleepy environments as additional constraints that the adversary must respect.

2.1 Execution Model

2.1.1 Modeling Protocol Execution

Interactive Turing Machines. We adopt the standard Interactive Turing Machines (ITM) approach to model protocol execution [9–11]. A protocol refers to an algorithm for a set of interactive Turing Machines (also called nodes) to interact with each other. The execution of a protocol Π that is directed by an environment $\mathcal{Z}(1^\kappa)$ (where κ is a security parameter), which activates a number of nodes as either *honest* or *corrupt* nodes. Honest nodes would faithfully follow the protocol's prescription, whereas corrupt nodes are controlled by an adversary $\mathcal{A}$ which reads all their inputs/message and sets their outputs/messages to be sent.

The environment $\mathcal{Z}$ is a term often used in protocol composition in the cryptography literature [9–11] — one can regard the environment $\mathcal{Z}$ a catch-all term that encompasses everything that lives outside the "box" defined by the protocol. For example, as mentioned later, part of the environment $\mathcal{Z}$'s job is to provide inputs to honest nodes and receive outputs from them. This models the fact that the inputs to the protocol may originate from external applications and the protocol's outputs can be consumed by external applications where any external application or other protocols running in the system are viewed as part of $\mathcal{Z}$.

Round-based execution. A protocol's execution proceeds in *rounds* that model atomic time steps. Henceforth, we use the terms *round* and *time* interchangeably. At the beginning of every round, honest and online nodes receive inputs from an environment $\mathcal{Z}$; at the end of every round, honest and online nodes send outputs to the environment $\mathcal{Z}$.

Corruption model. In the standard distributed systems or cryptography literature, crashed nodes are often treated as faulty and count towards the corruption budget. In this paper, we we distinguish *crashed* nodes (also referred to as *sleeping* nodes) and *corrupt* nodes. An honest node may have a short-term or long-term outage during which it is not able to participate in the

protocol. However, such a crashed node is not in the control of the adversary — in this case we do not attribute this node as corrupt. Informally, we often refer to the set of honest nodes that have not crashed as being *online*. We also consider all corrupt nodes as being online (since this gives the adversary more advantage).

We stress that the motivation for not treating crashed nodes as corrupt is to allow us to prove a more powerful theorem: our Thunderella paradigm ensures consistency and worst-case liveness when α fraction of the committee are honest but not necessarily online (and assuming that the underlying blockchain is secure). In particular, as we noted, α can be as small as a single member of the committee — but in this case the conditions necessary for instant confirmation are somewhat more stringent (i.e., all committee members must be honest and online for instant confirmation). In a more traditional model where crash is treated as corrupt, all of our theorems still apply — except that "honest" would equate to "honest and online".

More formally, in our model, the environment $\mathcal{Z}$ controls when nodes are spawned, corrupted, put to sleep, or waken up:

- At any time during the protocol execution, the environment $\mathcal{Z}$ can spawn new nodes, and newly spawned nodes can either be *honest* or *corrupt*. The adversary $\mathcal{A}$ has full control of all corrupt nodes.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `corrupt` instruction to an honest (and possibly sleeping) node. When this happens, its internal states are exposed to $\mathcal{A}$ and $\mathcal{A}$ henceforth controls the node.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `sleep` instruction to an honest node. When this happens, the node immediately becomes asleep (or sleeping), and it stops sending and receiving protocol messages and performing any computation. Sleeping is similar to the notion of a crash fault in the classical distributed systems terminology. In our paper, though, we treat sleeping nodes as being honest rather than attributing them towards the faulty budget.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `wake` instruction to an honest, sleeping node. At this point, this node immediately wakes up and continues to participate in the protocol. When an honest, sleeping node wakes up, pending messages that the node should have received while sleeping and additionally some adversarialy inserted messages may be delivered to the waking node.
- At any time during the protocol execution, $\mathcal{Z}$ can issue a `kill` instruction to a corrupt node. At this point, the corrupt node is removed from the protocol execution and is no longer considered as an online node — but note that the adversary $\mathcal{A}$ still knows the internal states of a killed node prior to its being killed.

Formally, we use the terminology *online* nodes to refer to the set of nodes that are *i)* either honest and not sleeping; or *ii)* corrupt but not having been killed.

Communication model. We assume that honest and online nodes can send messages to all other honest and online nodes. The adversary $\mathcal{A}$ is in charge of scheduling message delivery. $\mathcal{A}$ cannot modify the contents of messages broadcast by honest nodes, but it can reorder and delay messages sent by honest and online nodes, possibly subject to constraints on the maximum delays to be defined later. The adversary $\mathcal{A}$ is allowed to send messages to a subset of honest and online nodes but not all of them. The identity of the sender is not known to the recipient[1].

[1]Later in the paper, for instantiations in the permissioned model under a PKI, authenticated channels are implied by the PKI.

Formally, we say that $(\mathcal{A}, \mathcal{Z})$ *respects* Δ*-bounded delay* iff $\mathcal{Z}$ inputs Δ to all honest nodes when they are spawned, and moreover the following holds:

> Δ*-bounded delay.* Suppose an honest (and online) node sends a message at time t, then in any round $r \geq t + \Delta$, any honest node that is online in round r will have received the message, including nodes that may possibly have been sleeping but just woke up in round r, as well as nodes which may have just been spawned at the beginning of round r.

Throughout this paper, we assume that $\mathcal{Z}$ inputs the maximum delay parameter Δ to all honest nodes upon spawning (as noted in the above definition of Δ-bounded delay) — this means that the protocol has a-priori knowledge of an upper bound on the network's maximum delay. This is akin to the synchronous communication model in the classical distributed systems literature.

2.1.2 Constrained Execution Environments

Our execution model defined thus far is meant to be general. Later in the paper, sometimes we would like to say that a protocol retains security as long as $(\mathcal{A}, \mathcal{Z})$ respects certain constraints. We will model various assumptions as constraints on $(\mathcal{A}, \mathcal{Z})$.

Definition 1 ((n, ρ, Δ)-permissionless environments)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects (n, ρ, Δ)-permissionless execution w.r.t. some protocol Π iff for every $\kappa \in \mathbb{N}$ and every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold:

- there are exactly n online nodes in each round among which at most ρn are corrupt;
- $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded delay; and
- $\mathcal{Z}$ informs all honest nodes of the parameters (n, ρ, Δ) upon spawning.

Whenever the context is clear, we often omit writing "w.r.t. which protocol" without risk of ambiguity.

Corruption constraints. Our general model by default allows adaptive corruptions; however, for protocols that retain security only under static corruption, we will model static corruption as constraints on $(\mathcal{A}, \mathcal{Z})$. We say that $(\mathcal{A}, \mathcal{Z})$ respects static corruption w.r.t. to some protocol Π iff for every κ, for every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, $\mathcal{Z}$ never issues `corrupt` to an honest node after spawning (although $\mathcal{Z}$ is allowed to spawn corrupt nodes directly).

Permissioned model. Similarly, our general execution model can capture permissionless environments where nodes can join and leave the protocol at any time, and there is no a-priori knowledge of the number of nodes. However, whenever we are concerned about the permissioned setting, we can express "permissioned" execution environments also as additional constraints on $(\mathcal{A}, \mathcal{Z})$. Specifically, in a permissioned setting, we require that $\mathcal{Z}$ spawn all nodes upfront and inform all honest nodes the identities of all nodes spawned; further, $\mathcal{Z}$ is not allowed to issue `kill` instructions. Henceforth without loss of generality, we may assume that the spawned nodes have identities $0, 1, \ldots, n-1$ respectively. We will distinguish two types of permissioned execution environments:

1. In the permissioned-sleepy environment, $\mathcal{Z}$ can make nodes sleep — this way, we can treat crashed nodes not under adversarial control differently from corrupt nodes that are under adversarial control. Such a "sleepy" model was first considered in a recent work by Pass and Shi [43].

2. In the permissioned-classical environment, $\mathcal{Z}$ is not allowed to issue `sleep` or `wake` instructions — and thus the model here is akin to that considered in the classical distributed computing literature (in this case "honest" and "honest and online" are the same).

We now defined permissioned-sleepy and permissioned-classical environments more formally.

Definition 2 (Permissioned, sleepy environments)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects (n, ρ, Δ)-sleepy execution w.r.t. some protocol Π iff for every $\kappa \in \mathbb{N}$ and every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold: 1) $\mathcal{Z}$ spawns a set of numbered $0, 1, 2, \ldots, n-1$ (for some n) upfront prior to the start of protocol execution, and never spawns additional nodes later; 2) in every round, at most ρ fraction of the online nodes are honest; 3) $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded message delay; 4) $\mathcal{Z}$ does not issue `kill` instructions; and 5) $\mathcal{Z}$ informs all honest nodes of the parameters (n, ρ, Δ) upon spawning.

Henceforth when the context is clear, we often say that $(\mathcal{A}, \mathcal{Z})$ respects (n, ρ, Δ)-sleepy execution omitting which protocol Π we refer to.

Definition 3 (Permissioned, classical environments)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects (n, f, Δ)-classical execution w.r.t. some protocol Π iff for every $\kappa \in \mathbb{N}$ and every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold: 1) $\mathcal{Z}$ spawns a set of numbered $0, 1, 2, \ldots, n-1$ (for some n) upfront prior to the start of protocol execution, and never spawns additional nodes later; 2) $\mathcal{Z}$ never issues `sleep`, `wake`, or `kill` instructions; 3) $\mathcal{Z}$ corrupts at most f nodes; 4) $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded message delay; and 5) $\mathcal{Z}$ informs all honest nodes of the parameters (n, f, Δ) upon spawning.

Henceforth when the context is clear, we often say that $(\mathcal{A}, \mathcal{Z})$ respects (n, f, Δ)-classical execution omitting which protocol Π we refer to. Note that in a permissioned-classical environment, all nodes are online since $\mathcal{Z}$ cannot issue `sleep` instructions; and thus "honest" and "honest and online" are equivalent.

Besides these typical constraints on $(\mathcal{A}, \mathcal{Z})$, later we can also define protocol-specific constraints on $(\mathcal{A}, \mathcal{Z})$.

2.1.3 Notations

Notations for randomized execution. A protocol's execution is randomized, where the randomness comes from honest nodes, the adversary denoted $\mathcal{A}$ that controls all corrupt nodes, the environment $\mathcal{Z}$ that sends inputs to honest nodes during the protocol execution, and possibly the random oracle if the protocol adopts one.

We use the notation $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ to denote a randomly sampled execution trace, and $|\mathsf{view}|$ denotes the number of rounds in the execution trace view. More specifically, view is a random variable denoting the joint view of all nodes (i.e., all their inputs, random coins and messages received, including those from the random oracle) in the above execution; note that this joint view fully determines the execution.

Additional notational conventions. We use the notation κ to denote the security parameter. Unless otherwise stated, all variables appearing in the paper are implicitly polynomially bounded (or inverse polynomially bounded) functions of κ (and possibly of other variables). If some variable is not a function of κ, we will explicitly declare it as a *constant* for clarity.

2.2 State Machine Replication

State machine replication has been a central abstraction in the 30 years of distributed systems literature. In a state machine replication protocol, a set of nodes seek to agree on an ever-growing log over time. We require two critical security properties: 1) *consistency*, i.e., all honest nodes' logs agree with each other although some nodes may progress faster than others; 2) *liveness*, i.e., transactions received by honest nodes as input get confirmed in all honest nodes' logs quickly. We now define what it formally means for a protocol to realize a "state machine replication" abstraction.

Syntax. In a state machine replication protocol, in every round, an honest and online node receives as input a set of transactions txs from $\mathcal{Z}$ at the beginning of the round, and outputs a LOG collected thus far to $\mathcal{Z}$ at the end of the round.

Security definitions. Let $T_{\text{confirm}}(\kappa, n, \rho, \Delta, \delta)$ and $T_{\text{warmup}}(\kappa, n, \rho, \Delta, \delta)$ be polynomial functions in the security parameter κ and possibly other parameters of the view such as the number of nodes n, the corrupt fraction ρ, the actual maximum network delay δ, the network delay upper bound Δ that is provided by $\mathcal{Z}$ to the protocol as input, etc.

Definition 4 (Security of a state machine replication protocol)**.** We say that a state machine replication protocol Π satisfies consistency (or $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness resp.) w.r.t. some $(\mathcal{A}, \mathcal{Z})$, iff there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, consistency (or $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness resp.) is satisfied:

- *Consistency*: A view satisfies consistency iff the following holds:
  - *Common prefix.* Suppose that in view, an honest node i outputs LOG to $\mathcal{Z}$ at time t, and an honest node j outputs LOG' to $\mathcal{Z}$ at time t' (i and j may be the same or different), it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that in view, a node i is honest and online at time t and $t' \geq t$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.
- *Liveness*: A view satisfies $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness iff the following holds: if in some round $T_{\text{warmup}} < t \leq |\mathsf{view}| - T_{\text{confirm}}$, some node honest and online in round t either received from $\mathcal{Z}$ an input set txs that contains some transaction m or has m in its output log to $\mathcal{Z}$ in round t, then, for any node i honest and online at any time $t' \geq t + T_{\text{confirm}}$, let LOG be the output of node i at time t', it holds that $\mathsf{m} \in \mathsf{LOG}$.

  Intuitively, liveness says that transactions input to an honest node get included in honest nodes' LOGs within T_{confirm} time; and further, if a transaction appears in some honest node's LOG, it will appear in every honest node's LOG within T_{confirm} time.

2.3 Abstract Blockchain Protocols

A blockchain protocol can be regarded as a way to realize state machine replication. We now formally define what it means for a protocol to realize to a blockchain abstraction. In our paper, our end goal is to realize state machine replication and we leverage an abstract blockchain as an underlying building block. We note that while the blockchain abstraction may superficially resemble

that of state machine replication, the blockchain abstraction in fact allows us to additionally express 1) a rough notion of time through chain growth; and 2) fairness properties [40] through chain quality.

2.3.1 Syntax and Security Definitions

Syntax. An abstract blockchain protocol satisfies the following syntax. In each round, every node that is honest and online in this round receives from $\mathcal{Z}$ a set of transactions txs at the beginning of the round; and outputs to $\mathcal{Z}$ an abstract blockchain chain at the end of the round. An abstract blockchain denoted chain is an ordered sequence of blocks of the following format:

$$\mathsf{chain} := \{\mathsf{txs}_i\}_{i \in [|\mathsf{chain}|]}$$

where each txs_i is an application-specific payload such as a set of transactions.

Blockchain notations. We use the notation chain to denote an abstract blockchain. The notation $\mathsf{chain}[:-\ell]$ denotes the entire chain except the trailing ℓ blocks; $\mathsf{chain}[:\ell]$ denotes the entire chain upto the block at length ℓ; $\mathsf{chain}[-\ell:]$ denotes the trailing ℓ blocks of chain; and $\mathsf{chain}[\ell:]$ denotes all blocks at length ℓ or greater.

Henceforth we say that chain is "*an honest chain in* view", iff chain is some honest (and online) node's output to the environment $\mathcal{Z}$ in some round in view. We use the notation $\mathsf{chain}_i^t(\mathsf{view})$ to denote node i's chain in round t in view — since the context is clear, we often omit writing the view explicitly in the above notation.

Security definitions. A blockchain protocol should satisfy chain growth, chain quality, and consistency. Intuitively, chain growth requires that honest nodes' blockchains grow steadily, neither too fast nor too slow. Chain quality requires that in any honest node's chain, any sufficiently long window of consecutive blocks contains a certain fraction of blocks that are mined by honest nodes. Consistency requires that all honest nodes' chains agree with each other except for the trailing few blocks. We will formally define these security properties below.

Definition 5 (Security of an abstract blockchain protocol)**.** We say that a blockchain protocol $\Pi_{\text{blockchain}}$ satisfies (T, g_0, g_1)-chain growth, (T, μ)-chain quality, and T-consistency w.r.t. some $(\mathcal{A}, \mathcal{Z})$, iff there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{blockchain}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold for view:

- (T, g_0, g_1)-*chain growth.* A view satisfies (T, g_0, g_1)-chain growth iff the following hold:
  - *Consistent length:* If in round r some honest chain is of length ℓ, then in round $r + \Delta$, all honest chains must be of length at least ℓ.
  - *Growth lower bound:* For any r and t such that $g_0(t - r) \geq T$, let chain^r and chain^t denote two honest chains in round r and t respectively, it holds that
  $$|\mathsf{chain}^t| - |\mathsf{chain}^r| \geq \lfloor g_0(t - r) \rfloor$$
  - *Growth upper bound:* For any r and t such that $g_1(t - r) \geq T$, let chain^r and chain^t denote two honest chains in round r and t respectively, it holds that
  $$|\mathsf{chain}^t| - |\mathsf{chain}^r| \leq \lceil g_1(t - r) \rceil$$

- (T, L, μ)-*chain quality.* A view satisfies (T, L, μ)-chain quality iff the following holds: for any honest chain denoted chain in view, for any T consecutive blocks $\mathsf{chain}[j+1..j+T]$, more than μ fraction of the blocks in $\mathsf{chain}[j+1..j+T]$ are mined by honest nodes at most L blocks ago — here we say that a block $\mathsf{chain}[i]$ is "mined by an honest node at most L blocks ago" iff there is a set txs such that $\mathsf{txs} \subseteq \mathsf{chain}[i]$ and moreover $\mathcal{Z}$ input txs to some honest node when its last output to $\mathcal{Z}$ contains the prefix $\mathsf{chain}[:i-L]$ (here if $i - L < 0$, we round it to 0).

- T-*consistency.* A view satisfies T-consistency iff the following hold: for any two honest chains chain^r and chain^t in round r and $t \geq r$ respectively, it holds that

$$\mathsf{chain}^r[:-T] \prec \mathsf{chain}^t$$

  We stress that since chain^r and chain^t can possibly belong to the same node, the above definition also implies "future self consistency" except for the trailing T blocks.

Liveness as a derived property. Intuitively, liveness requires that if honest nodes receive a transaction m as input, then m appear in honest chains very soon. More formally, we say that a blockchain protocol $\Pi_{\mathrm{blockchain}}$ satisfies (K, T)-liveness w.r.t. some $(\mathcal{A}, \mathcal{Z})$ iff there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\mathrm{blockchain}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds:

- Suppose that in any round $r \geq t$, $\mathcal{Z}$ always inputs a set that contains some m to every honest and online node i unless $\mathsf{m} \in \mathsf{chain}_i^r[:-T]$. Then, for any honest chain denoted chain in view whose length is at least $\ell + K + T$, it holds that $\mathsf{chain}[:\ell + K]$ contains m where ℓ denotes the shortest honest chain length at time t.

The liveness of a blockchain protocol is directly implied by chain growth and chain quality as stated in the following lemma.

Lemma 1 (Liveness)**.** *Suppose that a blockchain protocol* $\Pi_{\mathrm{blockchain}}$ *satisfies* (K, g_0, g_1)*-chain growth,* (K', L, μ) *chain quality and* T*-consistency w.r.t. some* $(\mathcal{A}, \mathcal{Z})$ *for positive parameters* K, g_0, g_1, K', L, μ *and* T*, then it holds that* $\Pi_{\mathrm{blockchain}}$ *satisfies* $(2K + 2g_1 + K' + L, T)$*-liveness w.r.t.* $(\mathcal{A}, \mathcal{Z})$*.*

Proof. We ignore the negligible fraction of views where relevant bad events take place. Let r' be the earliest round in which some honest chain reaches length at least $\ell + K + g_1 + K' + L + T$, and let chain^* be an honest chain in round r' of length at least $\ell + K + g_1 + K' + L + T$. By chain quality, in the window $\mathsf{chain}^*[\ell + K + g_1 + L + 1 : \ell + K + g_1 + K' + L]$, there must be an honest block denoted B such that $\mathcal{Z}$ input (a subset of) the contents of B to some honest node i in round $r \leq r'$ when its chain contains the prefix $\mathsf{chain}^*[:\ell + K + g_1 + 1]$. By chain growth upper bound, the longest honest chain in round t must be of length at most $\ell + K + g_1$, and thus B must be input to some honest and online node i by $\mathcal{Z}$ in some round r where $t \leq r \leq r'$. By assumption, B must contain m unless $\mathsf{chain}_i^r[:-T]$ already contains m. By consistency, it must be that chain^* and chain_i^r are no longer than $\ell + 2(K + g_1) + K' + L + T$. By consistency, for any honest chain ch in view of length at least $\ell + 2(K + g_1) + K' + L + T$, it must be that $\mathsf{chain}_i^r[:-T] \prec \mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$ and $\mathsf{chain}^*[:-T] \prec \mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$, and thus $\mathsf{ch}[:\ell + 2(K + g_1) + K' + L]$ must contain m. □

2.3.2 Blockchain Implies State Machine Replication

Given any blockchain protocol $\Pi_{\text{blockchain}}$, it is easy to construct a state machine replication protocol where 1) nodes run an instance of $\Pi_{\text{blockchain}}$; 2) an honest node broadcasts all newly seen transactions to each other; and 3) in every round, nodes remove the trailing T blocks from the present chain (where T is the consistency parameter) and output the truncated chain to the environment $\mathcal{Z}$ [41, 43]. It is not difficult to see that consistency (of the resulting state machine replication protocol) follows directly from consistency of the blockchain; and liveness follows from chain quality and chain growth. The above intuition has been formalized in earlier works [41, 43].

2.4 Warmup: Blockchain Instantiations

The most representative blockchain protocol is obviously Nakamoto's orginal blockchain [38], i.e., the core consensus protocol underlying Bitcoin. In this section, however, we give several instantiations of an "abstract blockchain protocol" in both the permissioned and permissionless models to demonstrate the generality of this abstraction.

2.4.1 Permissionless Environment

Modeling a proof-of-work oracle. Our technique can be instantiated in a blackbox manner from any blockchain protocol, including proof-of-work and non-proof-of-work blockchains. A proof-of-work blockchain as represented by Nakamoto's original proposal [23, 38, 39] relies on a proof-of-work random oracle. Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ denote a random function. Nodes are allowed to query two functions H and $\mathsf{H.ver}$. $\mathsf{H}(x)$ simply outputs the outcome of the random function $H(x)$, and $\mathsf{H.ver}(x, y)$ outputs 1 iff $H(x) = y$, else it outputs 0. In any round, any node is allowed to make an arbitrary number of queries to $\mathsf{H.ver}$ but *at most one query* to H. If the adversary $\mathcal{A}$ controls q corrupt nodes, we allow $\mathcal{A}$ to make q sequential queries to H. We emphasize that the environment $\mathcal{Z}$ cannot access the random oracle.

Nakamoto's proof-of-work blockchain. Nakamoto's original proposal [38] relies on proof-of-work to defend against Sybil attacks and achieve consensus in a permissionless setting. Garay et al. [23] and Pass et al. [39] formally show that Nakamoto's blockchain is secure in a fully adaptive corruption model as long as the adversary controls only a minority of the computation power.

In the following theorem, we say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. Π^η_{nak} (parametrized by a positive constant η) iff there exist n, Δ and some positive constant $\rho < \frac{1}{2}$ such that $(\mathcal{A}, \mathcal{Z})$ respects (n, ρ, Δ)-permissionless execution.

Theorem 4 (Nakamoto's blockchain [23, 38, 39, 42])**.** *Assume the existence of a proof-of-work random oracle. Then, for any positive constant η, there exist a blockchain protocol Π^η_{nak} and positive constants c_0, c_1, such that for any p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair[2] that is compliant w.r.t. Π^η_{nak}, Π^η_{nak} satisfies the following w.r.t. $(\mathcal{A}, \mathcal{Z})$ for any positive constant ϵ:*

- $(\epsilon\kappa, \frac{1}{c_0\Delta}, \frac{1}{c_1\Delta})$*-chain growth;*
- $(\epsilon\kappa, 1, 1 - \frac{(1+\eta)\rho}{1-\rho})$*-chain quality;*
- $\epsilon\kappa$*-consistency; and*

[2]Throughout this paper, whenever we refer to a p.p.t.$(\mathcal{A}, \mathcal{Z})$ pair, we mean that $\mathcal{A}$ and $\mathcal{Z}$ are non-uniform probabilitic polynomial-time algorithms.

- $(\epsilon\kappa, \epsilon\kappa)$-*liveness.*

In the above theorem, Nakamoto's protocol Π^{η}_{nak} will automatically pick a suitable difficulty parameter based on the parameters (n, ρ, Δ) honest nodes receive from $\mathcal{Z}$ as well as η. We refer the readers to earlier works [39, 42] regarding how to concretely choose an appropriate difficulty parameter.

We stress that since we rely on the underlying blockchain as a blackbox, Thunderella can also be instantiated with a varying difficulty blockchain [22, 38] such that we can allow the number of players to vary over time.

2.4.2 Permissioned, Classical Environment

As a warmup, we will additionally show that in a permissioned, classical environment and assuming the existence of a public-key infrastructure (PKI), we can realize an abstract blockchain by modifying the classical Dolev-Strong protocol [18] that tolerates arbitrary number of corruptions.

Recall that the permissioned, classical setting can be formally modeled as further constraints on $(\mathcal{A}, \mathcal{Z})$. Specifically, recall that we say that $\mathcal{Z}$ respects permissioned-classical execution (or $(\mathcal{A}, \mathcal{Z})$ respects permissioned-classical execution) iff the following holds: 1) $\mathcal{Z}$ must spawn all nodes upfront prior to protocol execution; 2) $\mathcal{Z}$ must input the identities of all spawned nodes to every honest node as soon as it is spawned; and 3) $\mathcal{Z}$ is not allowed to issue `sleep` or `wake` instructions. Henceforth in this subsection, we will implicitly assume such a permissioned, classical execution environment; and in particular, "honest" now equates to "honest and online".

Multi-valued broadcast: definitions. First, we describe a multi-valued variant of the Dolev-Strong protocol which realizes a multi-valued broadcast abstraction. In a multi-valued broadcast protocol, there is a publicly known sender that will try to broadcast one or more values to all other nodes. At the beginning of the protocol, all nodes receive from $\mathcal{Z}$ the identity of the sender, as well as the maximum number of corrupt nodes f and the maximum network delay Δ. Moreover, an honest sender would receive an input set U from $\mathcal{Z}$. At the end of the protocol, all honest nodes output a set of values V.

Definition 6 (Security of multi-valued broadcast)**.** We say that a multi-valued broadcast protocol Π_{bcast} satisfies consistency and validity w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{bcast}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following properties hold for view:

- *Consistency.* If two honest nodes output the sets V and V' respectively in view, then $V = V'$.
- *Validity.* If the sender remains honest in view, then all honest nodes output U where U is the input the sender receives from $\mathcal{Z}$ at the beginning of view.

Multi-valued broadcast from Dolev-Strong. One can easily modify the Dolev-Strong protocol to realize multi-valued broadcast. The protocol Π_{bcast} is as described below — without loss of generality, imagine that there are n nodes numbered $0, 1, \ldots, n-1$, and node 0 is the sender. Henceforth we use the notation pk_i to denote node i's public key registered with the PKI.

Henceforth in our protocol, a *message-round* is defined to be Δ small rounds where Δ denotes the maximum network delay. In other words, nodes always wait sufficiently long to receive messages sent by honest nodes in the previous message round before taking any new action.

Remark 1 (Regarding composition)**.** *Our state machine replication protocol will make use of multiple instances of multi-valued broadcast. we adopt the following conventions necessary for secure protocol composition:*

1. All instances of multi-valued broadcast are spawned with a unique session identifier denoted sid.
2. We describe our protocols assuming a global signing functionality $\mathcal{G}_{\text{sign}}^{\Sigma}$ (parametrized by a signature scheme Σ) shared by all instances of multi-valued broadcast — see Appendix 5.4 for a formal specification and regarding how $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$. *Such a global signing functionality can be realized in the most obvious manner from a global bare public-key infrastructure* shared by all multi-valued broadcast instances.
3. Henceforth, when we describe our multi-valued broadcast protocol, when honest nodes need to sign messages, although not explicitly noted we assume that they always call $\mathcal{G}_{\text{sign}}^{\Sigma}$ to obtain a signature.

We now describe our multi-valued broadcast protocol below in the $\mathcal{G}_{\text{sign}}^{\Sigma}$-hybrid world.

- *Initialize:* Initialize $V := \emptyset$.
- *Message-round 0:* For every $v \in U$, sender broadcasts a tuple (v, σ_0) where σ_0 is a signature on the value v. All other nodes do nothing.
- *Each message-round r where $1 \leq r \leq f+1$:* The following is performed by all nodes. For every message $(v, \{\sigma_i\}_{i \in S})$ received from the network where S is a set of size r and σ_i is a signature on v valid under pk_i: if $v \notin V$, then
  a) add v to V;
  b) let $S' := S \cup \{\sigma^*\}$ where σ^* is the current node's signature on the value v; and
  c) broadcast (v, S').
- *Output:* Finally, at the end of message-round $f+1$, output V.

Henceforth we say that a protocol Π satisfies consistency (or validity resp.) in (n, f, Δ)-classical environments iff for every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects (n, f, Δ)-classical execution, Π satisfies consistency (or validity resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$.

Proposition 1 (Security of Π_{bcast})**.** *Assume that the signature scheme Σ is secure. For every n, Δ and every $f < n$, Π_{bcast} satisfies consistency and validity in (n, f, Δ)-classical environments.*

Proof. The proof is a straightforward modification of the classical proof due to Dolev and Strong [18]. □

Blockchain from multi-valued broadcast. We can construct the following blockchain protocol Π_{DS} in a permissioned, classical environment from multi-valued broadcast. We now describe Π_{DS} in the $\mathcal{G}_{\text{sign}}$-hybrid world.

- *Broadcast*: At the beginning of each message-round r, for each sender $i \in \{0, 1, \ldots, n-1\}$:
  a) fork a new instance of multi-valued broadcast $\Pi_{\text{bcast}}[sid = (r, i)]$ where the sender i is input as the sender.

b) let chain be the sender i's last output to $\mathcal{Z}$, i inputs to $\Pi_{\text{bcast}}[sid = (r, i)]$ a set containing every transaction that has been received from $\mathcal{Z}$ but is not in chain.

- *Output:* At the end of every message round r:

a) if $r < f + 1$, let $\mathsf{B} := \emptyset$; else for any i, create a new block B by concatenating the outputs of instances $\Pi_{\text{bcast}}[(r - (f + 1), i)]$ for $i = 0, 1, \ldots, n - 1$.

b) let chain be the last output to $\mathcal{Z}$ and $\mathsf{chain} =$ if nothing has been output.

c) output $\mathsf{chain}||\mathsf{B}$.

We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. Π_{DS} iff there exists some n, Δ and some $f < n$ such that $(\mathcal{A}, \mathcal{Z})$ respects (n, f, Δ)-classical execution.

Theorem 5 (Blockchain tolerating arbitrary corruptions in a permissioned, classical environment)**.** *Suppose that the signature scheme Σ is secure. Then, for any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. Π_{DS}, Π_{DS} satisfies the following security properties w.r.t. $(\mathcal{A}, \mathcal{Z})$:*

- $(0, \frac{1}{\Delta}, \frac{1}{\Delta})$-chain growth;
- $(1, 2, 1)$-chain quality;
- 0-consistency; and
- $(2, 0)$-liveness.

Proof. Follows from the security of Π_{bcast}, proof is straightforward and left as an exercise for the reader. □

2.4.3 Permissioned, Sleepy Environment

Classical modeling techniques in the distributed computing and cryptography literature typically treat crashed nodes as corrupt, and thus crashes would count towards the corruption budget. We next consider a permissioned model in which crashed nodes that are not under adversarial control are treated not as corrupt, but rather as honest but "sleepy" [43]. Such a "sleepy model" was first proposed in the recent work [43]. In the sleepy model, the set of online and honest nodes in adjacent rounds may be completely disjoint; and nodes can go to sleep and then wake up later. Recent work [43] showed that the following interesting facts:

- State machine replication is possible in such a permissioned, sleepy model, as long as in every round, the majority of *online* nodes are honest (Theorem 6).
- Interestingly, no known classical consensus protocol can achieve state machine replication in the sleepy model, even when we are guaranteed that 99% of online nodes must be honest in every round.
- Moreover, the honest majority assumption (among online nodes) turns out to be necessary to realize state machine replication in the sleepy model [43].

More concretely, Pass and Shi construct a blockchain protocol in the permissioned, sleepy model called "sleepy consensus" [43], assuming the existence of a PKI and a common reference string. Their protocol is inspired by Nakamoto's proof-of-work blockchain, but they show how to remove

the proof-of-work securely in a permissioned, sleepy setting. At a high level, the sleepy consensus protocol is very similar in nature to the original Nakamoto blockchain — while Nakamoto's blockchain uses proof-of-work to elect random leaders, the sleepy consensus protocol emulates the random leader election mechanism without relying on proof-of-work. Based on this high-level idea, a few extra tricks are necessary to ensure the security of the protocol [43].

Although Pass and Shi [43] show positive results under both adaptive and static corruption environments, below we state the theorem for the static corruption version where the parameters are tighter.

In the following theorem, we say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi^{\eta}_{\text{sleepy}}$ iff there exist some n, Δ and some positive constant $\rho < \frac{1}{2}$ such that $(\mathcal{A}, \mathcal{Z})$ respects (n, ρ, Δ)-sleepy execution, and moreover $(\mathcal{A}, \mathcal{Z})$ respects static corruption.

Theorem 6 (Blockchains in permissioned, sleepy environments [43])**.** *Assume the existence of a common reference string, a bare public-key infrastructure, and enhanced trapdoor permutations. For any positive constant η, there exists a blockchain protocol $\Pi^{\eta}_{\text{sleepy}}$ and positive constants c_0 and c_1, such that for any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi^{\eta}_{\text{sleepy}}$, for any positive constant ϵ, $\Pi^{\eta}_{\text{sleepy}}$ satisfies the following w.r.t. $(\mathcal{A}, \mathcal{Z})$:*

- $(\epsilon\kappa, \frac{\chi}{c_0\Delta}, \frac{1}{c_1\Delta})$*-chain growth;*
- $(\epsilon\kappa, 1, 1 - \frac{(1+\eta)\rho}{1-\rho})$*-chain quality;*
- $\epsilon\kappa$*-consistency; and*
- $(\epsilon\kappa, \epsilon\kappa)$*-liveness.*

where $\chi(\mathsf{view}) := \widetilde{n}(\mathsf{view})/n$ is the ratio of minimum number of online nodes in any round in view over n.

In the above theorem, we assume that $\Pi^{\eta}_{\text{sleepy}}$ will automatically choose an appropriate difficulty parameter based on the parameters n, ρ and Δ from $\mathcal{Z}$ as well as η. We refer the readers to the earlier work [43] for concrete choice of parameters.

We stress that our Thunderella paradigm can also be extended to the *adaptively* secure version of the sleepy consensus protocol [43] although protocol parameters must be re-selected.

2.5 Preliminaries: Responsiveness and Optimistic Responsiveness

2.5.1 Responsiveness

Recall that throughout this paper we always assume that $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded delay for some Δ, i.e., $\mathcal{Z}$ informs the protocol of a delay upper bound Δ upfront and all honest messages are then delivered within Δ number of rounds. A state machine replication protocol is said to be *responsive* if the transaction confirmation time is independent of the a-priori known upper bound Δ of the network's delay, but depends only on the actual maximum network delay. To put our results in perspective, we formally define the notion of responsiveness below and state a known lower bound result suggesting the impossibility of responsiveness against $\frac{1}{3}$ fraction of corruption. In the remainder of the paper, we will show that if one *optimistically* hopes for responsiveness only in lucky situations, then we can have protocols that retains consistency and liveness even under more than $\frac{1}{3}$ corruption. In practice, this means that we can have protocols that are responsive most of the time, and even when more than $\frac{1}{3}$ nodes are corrupt, the protocol can still guarantee consistency and liveness although performance would degrade when under attaack.

Responsiveness. We define a technical notion called responsiveness for a state machine replication protocol. Intuitively, responsiveness requires that except for a (possibly non-responsive) warmup period in the beginning, all transactions input afterwards will perceive transaction confirmation delay that is independent of the a-priori set upper bound Δ on the network's delay. As shown in earlier works [19, 41], responsive state machine replication is impossible if $\frac{1}{3}$ or more fraction of the nodes are corrupt (even in a permissioned, classical environment with static corruptions, and even assuming that a proof-of-work oracle exists).

Definition 7 (Responsive state machine replication [41])**.** Suppose that $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded delay for some Δ. We say that a state machine replication protocol Π satisfies $(T_{\text{confirm}}, T_{\text{warmup}}$-responsiveness w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff Π satisfies $(T_{\text{confirm}}, T_{\text{warmup}})$-liveness w.r.t. $(\mathcal{A}, \mathcal{Z})$, and moreover the function T_{confirm} does not depend on the a-prior delay upper bound Δ.

We say that a protocol Π satisfies consistency (or responsiveness resp.) in (n, f, Δ)-classical, static environments iff for every p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair that respects (n, f, Δ)-classical execution and static corruption, Π satisfies consistency (or responsiveness resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$. We can similarly define (n, ρ, Δ)-sleepy, static environments and (n, ρ, Δ)-permissionless, static environments.

Theorem 7 (Impossibility of responsiveness against $\frac{1}{3}$ corruption [41])**.** *For any n and f such that $n \leq 3f$ and for any polynomial T_{confirm} in κ and δ, and T_{warmup} in κ, Δ, and δ, there exists some polynomial function Δ in κ such that no state machine replication protocol no state machine replication protocol can can simultaneously achieve consistency and $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness in (n, f, Δ)-classical, static environments even assuming the existence of a proof-of-work oracle.*

The proof of the above theorem was presented by Pass and Shi in a recent work [41] where they modified the classical lower bound proof by Dwork, Lynch, and Stockmeyer [19] and made it work even in the proof-of-work setting.

Recall that permissioned-classical is expressed as constraints on $(\mathcal{A}, \mathcal{Z})$ in our formal framework. This means that a lower bound for $n \leq 3f$ in the classical setting immediately implies a lower bound in more permissive settings where $(\mathcal{A}, \mathcal{Z})$ need not respect the permissioned-classical constraints as long as $n \leq 3f$ (or the equivalent holds). Thus the following corollaries are immediate from Theorem 7.

Corollary 1. *For any n and any $\rho \geq \frac{1}{3}$, for any polynomial T_{confirm} in κ, and δ and polynomial T_{warmup} in κ, Δ, and δ, there exists some polynomial function Δ in κ such that no state machine replication protocol no state machine replication protocol can can simultaneously achieve consistency and $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness in (n, ρ, Δ)-sleepy, static environments even assuming the existence of a proof-of-work oracle.*

Corollary 2. *For any n and any $\rho \geq \frac{1}{3}$, for any polynomial T_{confirm} in κ, and δ and polynomial T_{warmup} in κ, Δ, and δ, there exists some polynomial function Δ in κ such that no state machine replication protocol no state machine replication protocol can can simultaneously achieve consistency and $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness in (n, ρ, Δ)-permissionless, static environments even assuming the existence of a proof-of-work oracle.*

Interestingly, how to achieve responsive state machine replication against fewer than $\frac{1}{3}$ fraction of corruption is also known in the in the permissioned setting assuming the existence of a PKI [13], as well as in the permissionless setting assuming proof-of-work [41] (and under additional technical assumptions).

2.5.2 "Bare Minimum" Optimistic Responsiveness

Since full responsiveness is subject to a $\frac{1}{3}$ corruption lower bound as stated in Theorem 7, we consider how to relax the notion of responsiveness such that we can in practical scenarios hope for responsiveness, say, most of the time, but not be subject to the $\frac{1}{3}$ corruption lower bound for worst-case security guarantees.

Although our protocols achieve much stronger guarantees regarding optimistic responsiveness, we start with a bare-minimum goal one can hope for: informally speaking, *can we have have a protocol (for permissionless or permissioned settings) that resists $\rho \geq \frac{1}{3}$ fraction of corruption in the worst case, but is at least "occasionally responsive" in honest executions where all nodes remain honest and online throughout?* Interestingly, to the best of our knowledge, no known state machine replication protocol can satisfy even the above "bare minimum" goal.

Let T_{opt} be a polynomial function in κ, n, ρ, and δ where n, ρ, δ are functions of a view and δ represents the maximum actual network delay. Henceforth in this paper, without loss of generality, we assume that $\mathcal{Z}$ *always inputs a fresh transaction to some honest node in every round.*

Definition 8. Given a view, we say that $[t_0, t_1]$ is a T_{opt}-responsive period in view iff for every transaction m input to some honest node in some round $t \in [t_0, t_1]$, it holds that in any round $r \geq t + T_{\text{opt}}(\kappa, n, \rho, \delta)$, any honest node's output log to $\mathcal{Z}$ contains m.

Definition 9. We say that Π is T_{opt}-*occasionally responsive* w.r.t. some $(\mathcal{A}, \mathcal{Z})$ iff there exists a negligible function $\mathsf{negl}(\cdot)$, such that for every κ, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, view contains a T_{opt}-responsive period of non-zero length.

Note that the above definition is only interesting if $(\mathcal{A}, \mathcal{Z})$ inputs a larger upper bound Δ than the actual maximum message delay.

Unfortunately, almost all known state machine replication protocols in the synchronous model (including blockchain-style protocols) are not occasionally responsive even when all nodes are honest. In particular, recall that earlier we described three blockchain instantiations Π_{nak}, Π_{DS}, and Π_{sleepy} in the permissionless, permissioned-classical, and permissioned-sleepy settings respctively. As noted in Section 2.3.2, we can easily construct state machine replication protocols from blockchains. The following facts state that the resulting state machine replication protocols are never responsive even under honest executions. Below we say that a protocol Π is T_{opt}-occasionally responsive in (n, ρ, Δ)-permissionless environments iff Π is T_{opt}-occasionally responsive w.r.t. to any $(\mathcal{A}, \mathcal{Z})$ that respects (n, ρ, Δ)-permissionless execution. The notions of (n, f, Δ)-classical environments and (n, ρ, Δ)-sleepy environments are similarly defined.

Fact 1. For any polynomial function T_{opt} and any n, there exists a polynomial function Δ such that the state machine replication protocol implied by Π_{nak} is not T_{opt}-occasionally responsive in $(n, \rho = 0, \Delta)$-permissionless environments.

Fact 2. For any polynomial function T_{opt} and any n, there exists a polynomial function Δ such that the state machine replication protocol implied by Π_{DS} is not T_{opt}-occasionally responsive in $(n, f = 0, \Delta)$-classical environments.

Fact 3. For any polynomial function T_{opt} and any n, there exists a polynomial function Δ such that the state machine replication protocol implied by Π_{sleepy} is not T_{opt}-occasionally responsive in $(n, \rho = 0, \Delta)$-*classical* environments.

Note that in the above theorem, we say that the sleepy consensus protocol Π_{sleepy} is not even occasionally responsive in honest *classical* environments where all nodes are honest and online throughout (in comparison, in sleepy environments, honest nodes can fall asleep).

Indeed, asking for occasional responsiveness in entirely honest executions is the bare minimum to hope for — despite this, no known state machine replication protocol can achieve it while tolerating $\frac{1}{3}$ corruptions in the worst case. Later in our paper, we will construct protocols that tolerate minority or arbitrarily many corruptions in the worst case, and achieve not just occasional responsiveness, but full responsiveness in honest executions. In fact, our protocols will achieve a much stronger form of optimistic responsiveness (not just in honest executions). Roughly speaking, our protocols are epoch based. We will guarantee optimistic responsiveness of the following nature: whenever we encounter a lucky epoch during which $(\mathcal{A}, \mathcal{Z})$ behaves "somewhat nicely", then during this epoch transactions will confirm responsively. More specifically, our protocols are leader-based such that during each epoch, a designated leader plays the role of a coordinator. Informally, suppose that we ask for worst-case guarantees under minority corruption — then, if during any epoch the leader remains honest and online and moreover more than $\frac{3}{4}$ fraction of nodes are honest and online, then this epoch is a *lucky epoch*. As a special case, if more than $\frac{3}{4}$ fraction of the nodes remain honest and online throughout the execution and the initial leader also remains honest and online, then the entire execution will be responsive.

Chapter 3

Basic Thunderella Protocol with a Static Committee

We first describe the basic Thunderella protocol assuming a static committee that is known a-priori to all nodes. We will discuss how to perform committee reconfiguration later in the paper. For conceptual simplicity, we describe a version of the protocol where the blockchain is also collecting transactions constantly in the background — in practical implementations, it will not be too difficult to optimize our theoretical construction further such that the blockchain need not store an additional copy of all transactions under optimistic conditions.

As mentioned, in general, the Thunderella paradigm can be instantiated with any suitable asynchronous protocol to serve as the optimistic path and any suitable synchronous protocol to serve as the fallback path. However, we believe that a particular attractive instantiation is to use a simple voting-based protocol for the optimistic path and a blockchain as the fallback. Thus for concreteness, we will describe Thunderella for this specific instantiation.

Terminology. Our basic approach assumes three logical entities:

- *miners* of the underlying blockchain $\Pi_{\text{blockchain}}$;
- a *leader*; and
- a *committee* denoted committee.

To retain consistency and worst-case liveness (i.e., confirmation in the speed of the underlying $\Pi_{\text{blockchain}}$), we need to assume that 1) the underlying blockchain $\Pi_{\text{blockchain}}$ retains security (and this would translate to different compliance rules depending on how we instantiate the underlying blockchain); 2) α fraction of the committee are assumed to remain honest (but not necessarily online) where α is a knob that effectively allows us to trade-off security and performance as is explained later. Notably, the leader need not be trusted for consistency and worst-case liveness.

For concreteness, in our description we will often assume that $\alpha = \frac{1}{2}$, but in fact our approach generalizes to any choice where $0 < \alpha < 1$; and whenever appropriate, we will remark how to generalize the scheme's parameters for arbitrary α.

For simplicity, in this section we start out by assuming a static committee. In a permissioned setting, this committee can be the set of all nodes. In a permissionless setting where the set of players are not known in advance, we can elect the committee dynamically from the underlying blockchain using known techniques [40, 41] or have stake-holders act as the committee [7, 15, 36] — however we defer the discussion of committee election and reconfiguration to later sections. Although we

assume a static committee in this section, our basic protocol supports leader reconfiguration. In our presentation below we focus on describing the mechanism that enables leader reconfiguration without specifying concretely what leader re-election policy to adopt — exactly what policy to adopt depends on the application context and we thus defer the discussion of policies to later sections.

3.1 Our Basic Protocol in a Nutshell

We first describe the intuition behind our basic protocol. For simplicity, we focus our description on what happens within a single epoch in which the identity of the leader is common knowledge.

3.1.1 Optimistic Fast Path

The optimistic fast path consists of a single round of voting to confirm each transaction (or batch). The leader serves as a coordinator and sequences transactions in the optimistic path. It tags each freshly seen transaction (or a batch) with a sequence number that increments over time, and the resulting tuple $(\mathsf{seq}, \mathsf{tx})$ is referred to as a notarization request. Whenever the committee members hear a notarization request $(\mathsf{seq}, \mathsf{tx})$ from the leader, it will sign the tuple $(\mathsf{seq}, \mathsf{tx})$ as long as it has not signed any tuple for seq before. For consistency, it is important that an honest committee member signs only one unique tuple $(\mathsf{seq}, \mathsf{tx})$ for every sequence number seq.

Whenever an honest node observes that a notarization request $(\mathsf{seq}, \mathsf{tx})$ has collected votes from more than $\frac{3}{4}$ of the committee, $(\mathsf{seq}, \mathsf{tx})$ is considered notarized. Although any notarized transaction is ready to be confirmed, an honest node is only allowed to output a notarized $(\mathsf{seq}, \mathsf{tx})$ tuple iff for every $s < \mathsf{seq}$, a tuple $(s, \_)$ has already been output. In other words, the output sequence is not allowed to skip any sequence numbers (since transactions must be processed in a linearized order). Henceforth, we referred to a sequence of notarized transactions with contiguous, non-skipping sequence numbers as a *lucky sequence*. In other words, honest nodes always output the maximal lucky sequence they have observed.

It is not hard to see that the optimistic, fast path satisfies the following properties as long as *the majority of the online committee members are honest* (below, we focus our discussion for the specific case $\alpha = \frac{1}{2}$, although the argument can easily be generalized to arbitrary choices of α):

- The following *agreement* property is satisfied even when the leader is corrupt and the committee may not be online: if any two honest nodes have output $(\mathsf{seq}, \mathsf{tx})$ and $(\mathsf{seq}, \mathsf{tx}')$ respectively, it must be that $\mathsf{tx} = \mathsf{tx}'$ (except with negligible probability over the choice of view).

- The following *liveness* property is satisfied only when the leader is honest and online and moreover more than $\frac{3}{4}$ of the committee are honest and online (i.e., when the optimistic conditions hold): every transaction input to an honest node will appear in all nodes' output logs in $O(1)$ actual roundtrips — in other words, when optimistic conditions hold, not only do we achieve liveness but we in fact also achieve *responsiveness*.

Note that when the optimistic conditions do not hold, liveness is not guaranteed for the optimistic path. For example, a corrupt leader can propose different transactions to different nodes for the same sequence number, and thus no transaction will collect enough votes to become notarized. Further, progress can also be hampered if not enough committee members are honest and online to vote.

Summarizing the above, if the leader is honest and online and moreover more than $\frac{3}{4}$ fraction of the committee are honest and online, all nodes will confirm transactions responsively in the

optimistic path. However, to make our protocol complete, we need to deal with the case when either the leader is corrupt (or not online), or the committee is not more than $\frac{3}{4}$ honest and online — in the latter case, we wish to fall back to the worst-case guarantees offered by the underlying blockchain. Below we describe how such fallback can be achieved.

3.1.2 Falling Back to the Blockchain

In the fallback slow path, nodes will confirm transactions using the slow blockchain. The most non-trivial part of our protocol is how to switch between the optimistic path and the fallback path. To this end, we must answer the following two questions.

1. How do nodes decide when to fall back to the slow path?
2. Once the above decision is made, what is the mechanism for achieving this fallback?

When to fall back. The idea is to use the underlying blockchain to collect evidence of the optimistic path not working (e.g., either due to corrupt or crashed leader or due to not sufficiently many committee members being honest and online). Such evidence must be robust such that the adversary cannot raise false alarms when the optimistic path is actually working.

For conceptual simplicity, we can imagine the following: 1) whenever honest nodes mine a block, they incorporate into the block their entire view so far, including all unnotarized transactions and notarized transactions they have seen — in the actual protocol, the transactions stored in the blockchain can be easily deduplicated and compressed; 2) honest nodes always gossip their views to each other, such that if one honest node sees some (notarized or unnotarized) transaction by round r, then all honest nodes will have seen it by round $r + \Delta$. Thus by the liveness property of the underlying blockchain, if any (notarized or unnotarized) transaction is observed by any honest node in round r, then in roughly $\epsilon\kappa$ blocks of time, the transaction will appear in the blockchain.

Now, we may use the following criterion to detect when the optimistic path is not working:

> *Fallback condition:* Assume that chain is the stabilized prefix of some honest node's blockchain. If some unnotarized transaction tx appears in the block $\mathsf{chain}[\ell]$ but tx still has not become part of a lucky sequence contained in $\mathsf{chain}[: \ell + \kappa]$ where κ is a security parameter[1], then we conclude that the optimistic path has failed, and that a fallback is necessary.

Note that if the optimistic conditions hold, then the leader would have observed the unnotarized tx when its blockchain is roughly ℓ in length, and the committee would have notarized tx quickly; thus tx will soon become part of a lucky sequence contained in every node's blockchain. If this has not happened within κ blocks of time, then the optimistic conditions must no longer hold.

We also note that using the above mechanism, all honest nodes will decide to fall back within Δ rounds from each other. We now reach our next question: what mechanism do we rely on for the falling back?

How to fall back. The challenge is that when honest nodes decide to fall back (within Δ rounds from each other), although their optimistic logs are prefixes of each other, the logs could be of different lengths. One decision to make during the fallback is where (i.e., at which sequence number) to end the optimistic log before switching to blockchain mode — importantly, for consistency, honest nodes must agree on this decision. We point out that agreeing on this decision actually requires a

[1] Transactions of a lucky sequence are allowed to appear out of order in the blockchain.

full agreement instance — unlike the optimistic path where we punted on liveness, here this decision must be made with both consistency and liveness.

Thus the most natural idea is to rely on the underlying blockchain to reach agreement regarding this decision. To this end, we introduce the notion of a grace period that serves as a cool-down period before we eventually fall back into slow mode. The grace period consists of κ number of consecutive blocks where κ is a security parameter. Let chain denote the stabilized part of an honest node's blockchain and suppose that ℓ^* is the first block such that $\mathsf{chain}[:\ell^*]$ triggers the "fallback condition" as described above. Then, the grace period will consist of the blocks $\mathsf{chain}[\ell^*+1:\ell+\kappa]$. Informally speaking, the grace period is utilized in the following manner:

- Let LOG^* be an honest node's output log at the moment that the grace period starts (thus LOG^* must be a lucky sequence);
- Let chain be the stabilized prefix of this honest node's chain:
  - If the grace period has not ended in chain, then the node outputs the longer of 1) LOG^*; and 2) the maximal lucky sequence contained in chain. Note that in this case, the node does not output any additional transactions that are not part of the lucky sequence.
  - Else if the grace period has ended in chain, then the node first outputs the maximal lucky sequence contained in chain; then it outputs every other transaction (notarized or unnotarized) contained in chain (in the order that they are included in chain). In other words, after the grace period is over, the nodes start confirming transactions based on the blockchain.

Let $\mathsf{LOG}_{\max}$ denote the maximal lucky sequence contained in an honest node's blockchain by the end of the grace period. Effectively, in the above mechanism, nodes agree on $\mathsf{LOG}_{\max}$ before falling to blockchain mode. Importantly, the following informal claim must hold:

Claim 1 (Informal)**.** Except with negligible probability, $\mathsf{LOG}_{\max}$ must be at least as long as any honest node's output log when the node detects the start of the grace period.

To see why, recall that as mentioned earlier, all honest nodes gossip always their protocol views to each other; and honest nodes always embed their entire protocol view into any block they mine (in the actual protocol, the messages can be compressed). Thus, by liveness, any honest node's output log when the grace period starts will be in the blockchain κ blocks later.

3.1.3 Initiating a New Optimistic Epoch

So far, we have described our protocol from the perspective of a single epoch in which the leader is common knowledge. Whenever the protocol is in a slow path, however, we would like to allow the nodes to try to reinitiate an optimistic epoch and try to be fast again. This is easy to achieve since our underlying blockchain is always up and live! Thus one can simply rely on the underlying blockchain to implement any policy-based decision to reinitiate a new epoch. For example, the blockchain can be used to agree on 1) at which block length to reinitiate a new epoch; and 2) who will act as the leader in the new epoch. Our Thunderella framework leaves it to the application layer to specify such policy decisions (e.g., such policies can be implemented through generic smart contracts running atop the underlying blockchain).

Our detailed description in the remainder of this section is aware of the existence of multiple epochs, and thus transactions' sequence numbers are tagged with the epoch number to avoid namespace collision.

3.2 Detailed Protocol Description

We now formally describe our basic Thunderella protocol with a static committee. Our description and proofs are modularized. Specifically, we first describe the minimal set of protocol instructions necessary for guaranteeing consistency (Section 3.2.2) — in an actual implementation, security audit should be prioritized for this part of the protocol. We then describe other surrounding mechanisms (e.g., how to concretely instantiate the chain state function and how the leader proposes transactions) that allow us to additionally achieve worst-case liveness (Section 3.2.3) and optimistic responsiveness (Section 3.2.4).

Concrete blockchain parameters. For concreteness, henceforth in this section we assume a blockchain protocol denoted $\Pi_{\text{blockchain}}$ that achieves $(0.05\kappa, g_0, g_1 = \frac{1}{c\Delta})$-chain growth for some positive g_0 and some positive constant c, $(0.05\kappa, 1, \mu)$-chain quality where μ is positive, 0.05κ-consistency, and $(0.05\kappa, 0.05\kappa)$-liveness w.r.t. to any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{blockchain}}$. All the concrete blockchain instantiations mentioned earlier in Section 2.4 satisfy these parameters (Theorems 4, 5, and 6). Although we assume these concrete parameters, our Thunderella framework can easily be generalized to other parameters.

3.2.1 Useful Definitions

Henceforth, let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ denote a digital signature scheme.

Notarized transactions. We say that a tuple (e, s, m, V) is a *notarized* transaction for epoch e and sequence number s w.r.t. committee iff

- For each $(\mathsf{pk}, \sigma) \in V$, $\mathsf{pk} \in \mathsf{committee}$ and moreover σ is a valid signature for (e, s, m) under pk — in this case, we also say that (pk, σ) is a *valid vote* for (e, s, m).
- There are more than $\frac{3}{4} \cdot \left|\mathsf{committee}\right|$ votes in V with distinct pks.

If (e, s, m, V) is a notarized transaction, we also say that V is a *valid notarization* for (e, s, m).

Remark 2. *Note that the above definition works for $\alpha = \frac{1}{2}$. For a general $\alpha \in (0, 1]$, we can simply replace the constant $\frac{3}{4}$ with $1 - \frac{\alpha}{2}$.*

Blockchain states. We assume that there is a deterministic and efficiently computable function Γ such that given an abstract blockchain chain, the function Γ divides chain into multiple *epochs* interspersed with *interims*. Each epoch is a sequence of consecutive blocks in chain, and f also outputs a unique epoch number e for each epoch. A sequence of consecutive blocks that do not belong to any epoch are called interim blocks. Each epoch always contains two sub-phases, an *optimistic period* followed by a *grace period*, each of which contains at least κ consecutive blocks and thus each epoch contains at least 2κ consecutive blocks (unless end of chain is reached).

Formally, we say that $\Gamma(\kappa, \cdot, \cdot)$ is a chain-state function iff for any chain and $0 \leq \ell \leq |\mathsf{chain}|$, $\Gamma(\kappa, \mathsf{chain}, \ell)$ outputs one of the following:

- some $(e, \texttt{optimistic})$: in this case we say that $\mathsf{chain}[\ell]$ is an optimistic block belonging to epoch e (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$);
- some $(e, \texttt{grace})$: in this case we say that $\mathsf{chain}[\ell]$ is a grace block belonging to epoch e (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$);

- or `interim`: in this case we say that $\mathsf{chain}[\ell]$ is an interim block (w.r.t. $\Gamma(\kappa, \cdot, \cdot)$).

We say that a chain-state function $\Gamma(\kappa, \cdot, \cdot)$ is admissible iff for any chain:

1. for any $0 \le \ell \le \ell' \le |\mathsf{chain}|$, if $\mathsf{chain}[\ell]$ belongs to epoch e and $\mathsf{chain}[\ell']$ belongs to epoch e', then $e' \ge e$;
2. for every e: all blocks corresponding to epoch e in chain must appear in a consecutive window, and moreover, all optimistic blocks for epoch e must appear before grace blocks for epoch e;
3. for every epoch e appearing in chain: there must be at least κ grace blocks belonging to epoch e in chain unless chain ends at an epoch-e block.
4. for every chain and every $0 \le \ell \le |\mathsf{chain}|$, $\Gamma(\kappa, \mathsf{chain}, \ell)$ depends only on $\mathsf{chain}[: \ell]$ but not $\mathsf{chain}[\ell + 1 :]$.

Lucky sequence. A sequence of notarized transactions $\{(e_i, s_i, \mathsf{m}_i, V_i)\}_{i \in [m]}$ is said to be a lucky sequence for epoch e iff for all $i \in [m]$, $e_i = e$ and $s_i = i$.

Blockchain linearization. Given an abstract blockchain chain, we do not simply output all transactions in chain in the most natural way. Instead, we adopt an algorithm denoted $\mathsf{linearize}^{\Gamma(\kappa,\cdot,\cdot)}(\mathsf{chain})$ for chain linearization. Henceforth we often write $\mathsf{linearize}(\mathsf{chain})$ for simplicity without explicitly denoting the chain-state function $\Gamma(\kappa, \cdot, \cdot)$.

Our chain linearization algorithm $\mathsf{linearize}(\mathsf{chain})$ is defined as follows: scan through the chain from left to right, and output the following:

1. For each epoch $\mathsf{chain}[\ell : \ell']$ encountered with the epoch number e, output the following in order:
   - first extract the maximal lucky sequence TXs for epoch e from $\mathsf{chain}[: \ell']$ and output $\mathsf{strip}(\mathsf{TXs})$ where $\mathsf{strip}(\cdot)$ will be defined below;
   - if $\mathsf{chain}[\ell]$ is not the end of chain, let TXs' be all remaining records in $\mathsf{chain}[\ell : \ell']$ not contained in TXs, output $\mathsf{strip}(\mathsf{TXs}')$;
2. For each interim $\mathsf{chain}[\ell : \ell']$ encountered, extract all transactions TXs from $\mathsf{chain}[\ell : \ell']$ and output $\mathsf{strip}(\mathsf{TXs})$.

In the above, the function $\mathsf{strip}(\cdot)$ removes signatures from notarized transactions: for a notarized transaction $\mathsf{strip}(e, s, \mathsf{m}, V) := (e, s, \mathsf{m})$; for an unnotarized transaction we define $\mathsf{strip}(\mathsf{m}) := \mathsf{m}$. If the input to $\mathsf{strip}(\cdot)$ is a sequence of transactions, the same operation is applied to each transaction.

3.2.2 Π_{thunder}: Core Protocol for Consistency

Additional notation. A node's view consists of every message (including blockchains) it has received from $\mathcal{Z}$ or over the network. Henceforth we say that a notarized transaction (e, s, m, V) is in a node's view iff (e, s, m) exists in the node's view, and every $(\mathsf{pk}, \sigma) \in V$ exists in the node's view (not necessarily appearing together in the node's view). Multiple notarized transactions can exist for a unique (e, s, m) by taking different subsets of V — but in our presentation below, we always take V to be all the valid votes for (e, s, m) in a node's view, such that if for some tuple (e, s, m) there is a notarized transaction (e, s, m, V) in a node's view, then the choice is unique.

Assumptions. Although not explicitly noted, henceforth in all of our protocols, we assume that whenever an honest node receives any message on the network, if the message has not been broadcast before, the honest node broadcasts the message.

Protocol Π_{thunder}. Below we describe the $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ protocol that is parametrized by an admissible chain-state function $\Gamma(\kappa,\cdot,\cdot)$. Henceforth in our scheme, we often omit explicitly writing the chain-state function $\Gamma(\kappa,\cdot,\cdot)$.

- *Initialize.*
  - Call $(\mathsf{pk},\mathsf{sk}) \leftarrow \Sigma.\mathsf{Gen}(\kappa)$ to generate a signing key pair. Output pk to $\mathcal{Z}$.
  - Wait to receive $\mathsf{committee}$ from $\mathcal{Z}$, and henceforth, validity of votes and acceptability of chains will be defined w.r.t. $\mathsf{committee}$.
  - Fork an instance of the $\Pi_{\text{blockchain}}$ protocol with appropriate parameters determined by ρ, n and Δ[2].
- *Notarize.* Upon receiving notarization request (e,s,m) from $\mathcal{Z}$: if $\mathsf{pk} \in \mathsf{committee}$ and no signature has been produced for (e,s) earlier, compute $\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(e,s,\mathsf{m})$ and broadcast $((e,s,\mathsf{m}),\sigma)$.
- *Propose.* Every round, let chain be the output from the $\Pi_{\text{blockchain}}$ instance.
  - Let TXs be a set containing 1) every notarized transaction (e,s,m,V) in the node's view such that no notarized transaction $(e,s,\mathsf{m},\_)$ has appeared in $\mathsf{chain}[:-0.5\kappa]$; and 2) every unnotarized transaction m in the node's view such that no m or notarized transaction $(e,s,\mathsf{m},\_)$ has appeared in $\mathsf{chain}[:-0.5\kappa]$.
  - Propose TXs to $\Pi_{\text{blockchain}}$.
- *Output.* In every round, let chain be the output from $\Pi_{\text{blockchain}}$.
  - If $\mathsf{chain}[-0.5\kappa]$ is an optimistic block belonging to epoch e:
    a) let $\mathsf{chain}[-\ell]$ be the starting block for epoch e in chain where $\ell \geq 0.5\kappa$.
    b) extract the maximal lucky sequence TXs for epoch e from the node's view so far.
    c) let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[:-(\ell+1)])||\mathsf{strip}(\mathsf{TXs})$.
  - Else, let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[:-0.5\kappa])$.
  - Let LOG be the previous output to $\mathcal{Z}$: if $\overline{\mathsf{LOG}}$ is longer than LOG, output $\overline{\mathsf{LOG}}$; else output LOG to $\mathcal{Z}$.
- *Mempool.* Upon receiving any other message from the network or $\mathcal{Z}$, record the tuple.

Compliant executions. We say that $(\mathcal{A},\mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ iff

- $(\mathcal{A},\mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{blockchain}}$;
- in every view in the support of $\mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A},\mathcal{Z},\kappa)$, $\mathcal{Z}$ always inputs the same $\mathsf{committee}$ to all honest nodes;
- in every view in the support of $\mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A},\mathcal{Z},\kappa)$, more than $\frac{1}{2}$ fraction (or in general, more than α fraction) of the distinct public keys in $\mathsf{committee}$ are output by nodes that remain honest (but not necessarily online) forever.

[2]Unless otherwise noted, all messages sent from the $\Pi_{\text{blockchain}}$ instance or destined for $\Pi_{\text{blockchain}}$ are automatically passed through, but these messages also count towards the view of the current Π_{thunder} protocol instance.

The following theorem says that for any chain-state function f that is admissible, Π^{f}_{thunder} satisfies consistency under compliant executions.

Theorem 8 (Consistency)**.** *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Then, $\Pi^{\Gamma(\kappa,\cdot,\cdot)}_{\text{thunder}}$ satisfies consistency as defined in Section 2.2 w.r.t. any p.p.t. $(\mathcal{A},\mathcal{Z})$ that is compliant w.r.t. $\Pi^{\Gamma(\kappa,\cdot,\cdot)}_{\text{thunder}}$.*

The proof of this theorem will be presented in Section 3.3.1.

3.2.3 Concrete Chain-State Function and Worst-Case Liveness

We will adopt the following chain-state function $\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)$ that is parametrized by a polynomial-time boolean predicate pred henceforth referred to as the "next-epoch" function. Basically, the job of pred is to examine the prefix of some blockchain and decide whether we want to advance to a larger epoch. Specifically, for some chain prefix $\mathsf{chain}[:i]$ if $\mathsf{pred}(\mathsf{chain}[:i],e)=1$ then the blockchain wants to advance to epoch e if it is not already in epoch e — if there are multiple such e's such that the above holds, then the blockchain wants to go the the largest such epoch.

At this moment, we define the chain state function Γ while leaving the pred unspecified. We will show that worst-case liveness is satisfied in compliant executions regardless of the concrete policy pred. Intuitively, our concrete chain state function is very simple: If the blockchain is currently in some epoch e, then the chain will stay in epoch e unless one of the following things happen:

1. either pred (applied to the prefix of the blockchain) wants to go to a larger epoch; or
2. during the current epoch some transaction did not get confirmed for a long time.

If one of the above did happen, then the chain gracefully transitions to an interim ensuring that there are at least κ optimistic blocks for the current epoch e followed by at least κ grace blocks for epoch e. If the blockchain is in an interim and pred wants to go to a next epoch, then we advance to the next epoch immediately. We note that for consistency and worst-case liveness, we in fact only need that there are at least κ grace blocks for each epoch (but not necessarily κ or more optimistic blocks). Here we additionally require that there are at least κ optimistic blocks for each epoch too — this gives the new epoch some time such that the blockchain can pick up possibly stale transanctions that ought to have been confirmed such that we do not exit from the current epoch too soon.

More formally, for any chain, $\Gamma^{\mathsf{pred}}(\kappa,\mathsf{chain},\cdot)$ is inductively defined as the following:

- The $\mathsf{chain}[0] := \mathsf{genesis}$ block is considered an interim block;
- If $\mathsf{chain}[i]$ is an interim block, let e be the largest epoch number such that $\mathsf{pred}(\mathsf{chain}[:i{+}1],e)=1$, but no prefix of $\mathsf{chain}[:i]$ was ever in epoch e:
  - If such an epoch e is found: then $\mathsf{chain}[i+1..i+\kappa]$ are all optimistic blocks for epoch e' (and if $|\mathsf{chain}|<i+\kappa$, then all of $\mathsf{chain}[i+1:]$ are optimistic blocks for epoch e').
  - Else $\mathsf{chain}[i+1]$ is also an interim block;
- If $\mathsf{chain}[i]$ is the ℓ-th optimistic block of some epoch e where $\ell\geq\kappa$:
  - If one of the following two conditions C1 or C2 hold, then $\mathsf{chain}[i+1..i+\kappa]$ are all grace blocks for epoch e, and $\mathsf{chain}[i+\kappa+1]$ is an interim block (and if $|\mathsf{chain}|\leq i+\kappa$ then all of $\mathsf{chain}[i+1:]$ are grace blocks for epoch e):

C1: some m or some notarized transaction $(\_, \_, \mathsf{m}, \_)$ appears in $\mathsf{chain}[: i-0.5\kappa]$ but $\mathsf{linearize}(\mathsf{chain}[: i])$ does not contain m or $(\_, \_, \mathsf{m})$, i.e., if some transaction has not occurred in any lucky sequence even after a sufficiently long time;

C2: there exists some $e' > e$ such that $\mathsf{pred}(\mathsf{chain}[: i+1], e') = 1$, i.e., if the next-epoch policy function wants to switch to a larger epoch than the current one.

– Else $\mathsf{chain}[i+1]$ is an optimistic block of epoch e.

Theorem 9 (Worst-case liveness)**.** *Let $\Gamma(\kappa, \cdot, \cdot) := \Gamma^{\mathsf{pred}}(\cdot, \cdot, \cdot)$ be the chain-state function as specified above for any polynomial-time boolean predicate pred. Let g_0 denote the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter, and let $T_{\text{confirm}}(\kappa) := \frac{3\kappa}{g_0}$. For any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathit{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: suppose that $\mathcal{Z}$ inputs a transaction m to an honest node in round r, then in any round $r' \geq r + T_{\text{confirm}}(\kappa)$, all honest and online nodes' output LOG to $\mathcal{Z}$ will contain some $(\_, \_, \mathsf{m})$ or m.*

The proof of the above theorem is deferred to Section 3.3.2.

3.2.4 Coordination Protocol Π_{ella} and Optimistic Responsiveness

We now describe the full protocol $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ that spells out the leader-based coordination mechanism on top of Π_{thunder} as well as the next-epoch function pred. We will then show under exactly what optimistic conditions our protocol achieves responsiveness.

Description of protocol Π_{ella}. Π_{ella} calls $\Pi_{\text{thunder}}^{\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)}$ where the chain state function $\Gamma(\kappa, \cdot, \cdot) := \Gamma^{\mathsf{pred}}(\kappa, \cdot, \cdot)$ is as defined in Section 3.2.3. We spell out the next-epoch function pred and the rest of Π_{ella} below.

- *Next-epoch function.* The policy function $\mathsf{pred}(\mathsf{chain}, e)$ takes in an abstract blockchain denoted chain and an epoch number e. If there exists a notarized transaction for epoch e in chain, then output 1; else output 0.
- *Initialize*: fork an instance of the $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ protocol.
- *Leader switch*: upon input $\texttt{leader}(e, i)$: if no leader has been recorded for epoch e, record i as the leader for epoch e, and do the following:
  - if current node is i: send a notarization request for a special epoch-start transaction $(e, s = 1, \texttt{start})$, and let $s = 2$;
  - for every notarization request (e, s, m) received earlier from node i, act as if (e, s, m) has just been received from i.
- *Notarization*: upon receiving notarization request (e, s, m) from i: if i has been recorded as the leader for epoch e, forward the notarization request (e, s, m) to $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$; else ignore the request.
- *Leader*: every round: let e be the largest epoch recorded thus far and if current node is recorded as the leader for epoch e:
  - for every m in view such that no m or $(\_, \_, \mathsf{m})$ appears in $\mathsf{linearize}(\mathsf{chain}[: -\kappa])$, if a notarization request has not been broadcast for m earlier, then broadcast the notarization request (e, s, m) and let $s := s + 1$.

- *Other messages*: pass through all other messages between $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and $\mathcal{Z}$; similarly pass through all other messages between $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and the network.

Compliant executions. To guarantee consistency and worst-case liveness, basically we just need the same conditions as our earlier $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ iff $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$.

Lucky epoch. Below we will describe exactly under what optimistic conditions can we achieve responsiveness. Roughly speaking, whenever a lucky epoch begins, after a short warmup time, we can achieve responsiveness. Specifically, during a lucky epoch, the epoch's leader is online and honest and more than $\frac{3}{4}$ fraction (or in general, $1 - \frac{\alpha}{2}$ fraction of the committee remain honest and online.

Formally, given a view, we say that $[T_{\text{start}}, T_{\text{end}}]$ belongs to a lucky epoch corresponding to epoch e and leader i iff the following hold:

- In any round $r \geq T_{\text{start}} + \Delta$, any honest and online node should have received $\texttt{leader}(e, i)$ where i is the common leader that all honest nodes receive for epoch e. Further, prior to T_{start}, no honest node has received from $\mathcal{Z}$ any $\texttt{leader}(e', \_)$ instruction where $e' \geq e$.
- the leader (i.e., node i) is honest and online at in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$;
- more than $\frac{3}{4}$ fraction (or in general, more than $1 - \frac{\alpha}{2}$ fraction) of committee are honest and online [3] in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$.

Optimistic responsiveness in lucky epochs. We say that a protocol Π satisfies $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$: for any duration $[T_{\text{start}}, T_{\text{end}}]$ in view that belongs to a lucky epoch, $[T_{\text{start}} + T_{\text{warmup}}, T_{\text{end}}]$ is a T_{opt}-responsive period in view.

Theorem 10 (Optimistic case responsiveness)**.** *Let g_0 be the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter. For every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$, $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ satisfies $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs for $T_{\text{warmup}} = O(\frac{\kappa}{g_0})$, and $T_{\text{opt}} = 3\delta$ where δ is the actual maximum network delay in view.*

The proof of the above theorem is deferred to Section 3.3.3. We note that Theorem 10 implies the following: informally speaking, if throughout the execution more than $\frac{3}{4}$ fraction of the committee remain honest and online and moreover, the initial epoch's leader remains honest and online, then once nodes enter the initial epoch, after a short warmup period, our protocol Π_{ella} will achieve responsiveness throughout the remainder of the execution (assuming that the underlying blockchain is secure).

Remark 3 (Leader re-election mechanism)**.** *In our scheme earlier, we left it unspecified how the environment $\mathcal{Z}$ will decide when to issue leader-switch instructions of the form $\texttt{leader}(e, i)$ that will cause nodes to start a new leader epoch. This is an application-specific policy decision. At this point, our paper focuses on providing a general framework that enables any application-specific policy decisions. Later in our paper, we will give some suggestions on leader re-election policies that are useful in practice.*

[3] We say that a public key $\mathsf{pk} \in \mathsf{committee}$ is honest and online in round r if some node that is honest and online in round r output pk to $\mathcal{Z}$ earlier.

3.3 Proofs for Basic Thunderella with Static Committee

We now move onto presenting the formal proofs for our basic Thunderella protocol.

Shorthand notations. Henceforth in Section 3.3.1 and 3.3.2, whenever we say that "except with negligible probability over the choice of view, some property ev(view) holds", we formally mean that for any $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, ev(view) holds. Similarly, in Section 3.3.3, whenever we say that "except with negligible probability over the choice of view, some property ev(view) holds", we formally mean the same as above but now referring to $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$ instead.

3.3.1 Consistency

Lemma 2 (Uniqueness)**.** *Assume that the signature scheme is secure, then except with negligible probability over the choice of* view*, the following holds. For any (e, s) pair, if two notarized transactions (e, s, m, V) and (e, s, m', V') appear in* view*, then it must be the case that $\mathsf{m} = \mathsf{m}'$.*

Proof. For V or V' to be a notarization for (e, s, m) and (e, s, m') respectively, there must be more than $\frac{3}{4}M$ number of signatures from distinct members of committee where $M := |\mathsf{committee}|$. Since the adversary controls $f < \frac{1}{2}M$ members of committee, it must be that at least $\frac{3}{4}M - f$ honest members of committee have signed (e, s, m) — otherwise we can leverage $(\mathcal{A}, \mathcal{Z})$ to build a reduction that breaks the signature security. Similarly, at least $\frac{3}{4}M - f$ honest members of committee have signed (e, s, m'). Thus, by the pigeon-hole principle at least one honest member of committee has signed both (e, s, m) and (e, s, m'). By definition of the honest protocol, honest nodes will sign each sequence number no more than once. Thus it holds that $\mathsf{m} = \mathsf{m}'$. □

Fact 4. Let $\Gamma(\kappa, \cdot, \cdot)$ be any admissible chain-state function. Then for any chain and any $0 \le i < j \le |\mathsf{chain}|$, $\mathsf{linearize}^{\Gamma(\kappa,\cdot,\cdot)}(\mathsf{chain}[:i]) \prec \mathsf{linearize}^{\Gamma(\kappa,\cdot,\cdot)}(\mathsf{chain}[:j])$.

Proof. Follows directly from the admissibility definition of Γ and the definition of linearize. □

Let $\overline{\mathsf{LOG}}_i^r$ denote the internal variable $\overline{\mathsf{LOG}}$ of node i in round r.

Lemma 3. *Let $\Gamma(\kappa, \cdot, \cdot)$ be any admissible chain-state function. Except with negligible probability over the choice of* view*, the following holds: for any r and t, for any node i honest and online in round r and any node j honest and online in round t, either $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$ or $\overline{\mathsf{LOG}}_j^t \prec \overline{\mathsf{LOG}}_i^r$.*

Proof. In every round, an honest node's $\overline{\mathsf{LOG}}$ is of the generalized form $\mathsf{linearize}(\mathsf{chain}[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ where

- either TXs is empty and $\ell = 0.5\kappa$ or
- TXs is a non-empty lucky sequence for some epoch e and $\ell \ge 0.5\kappa$.

Suppose that $\overline{\mathsf{LOG}}_i^r := \mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ and $\overline{\mathsf{LOG}}_j^t := \mathsf{linearize}(\mathsf{chain}_j^t[:-\ell'])||\mathsf{strip}(\mathsf{TXs}')$ are output logs by node i in round r and by node j in round t respectively in view. Henceforth we ignore the negligible fraction of views where relevant bad events happen. For the remaining good views, the following statements hold.

By consistency of $\Pi_{\text{blockchain}}$, either $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$ or $\mathsf{chain}_j^t[:-\ell'] \prec \mathsf{chain}_i^r[:-\ell]$. Without loss of generality, henceforth we assume that $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$. We now consider the following cases:

- Case 1: TXs is empty. In this case, it follows directly from Fact 4 that $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$.
- Case 2: TXs is non-empty. In this case, $\mathsf{chain}_i^r[-\ell+1]$ must be the starting block of some epoch e, and $\mathsf{chain}_i^r[-0.5\kappa]$ must be an optimistic block of epoch e. We now consider the following sub-cases:
  - Case 2a: Epoch e has completed in $\mathsf{chain}_j^t[:-\ell']$, i.e., there are two adjacent blocks such that the former belongs to epoch e and the latter does not in $\mathsf{chain}_j^t[:-\ell']$. Let $\mathsf{chain}_j^t[-\ell^*]$ denote the last block of epoch e in $\mathsf{chain}_j^t[:-\ell']$. By Fact 4, it suffices to prove that $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs}) \prec \mathsf{linearize}(\mathsf{chain}_j^t[:-\ell^*])$.
    By definition of the honest protocol, TXs must be a lucky sequence for epoch e in node i's view in round r, and further, by round $r+\Delta$, TXs must exist in any honest node's view. By the admissibility of the chain-state function Γ and the fact that $\mathsf{chain}_i^r[-0.5\kappa]$ is an optmistic block of epoch e, $\mathsf{chain}_j^t[:-\ell^*]$ must be at least 0.5κ longer than chain_i^r. By chain growth upper bound, by round $r+\Delta$, any honest chain must be at most $\ell_i^r + 0.25\kappa$ in length where $\ell_i^r := |\mathsf{chain}_i^r|$. By liveness, a lucky sequence of length $|\mathsf{TXs}|$ for epoch e must appear in $\mathsf{chain}_j^t[:-\ell^*]$. The remainder of the proof follows from definition of $\mathsf{linearize}$ and Lemma 2.
  - Case 2b: Epoch e has not completed in $\mathsf{chain}_j^t[:-\ell']$. Recall that node i outputs $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ in round r. By definition of the honest algorithm, and since $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$, node j's output in round t must be of the form $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs}^*)$ where TXs^* is either empty or some lucky sequence for epoch e in node j's view in round t. By Lemma 2, it holds that either $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$ or $\overline{\mathsf{LOG}}_j^t \prec \overline{\mathsf{LOG}}_i^r$.

□

Theorem 11 (Consistency, restatement of Theorem 8)**.** *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Then, $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ satisfies consistency as defined in Section 2.2 w.r.t. any p.p.t. $(\mathcal{A},\mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$.*

Proof. By construction, an honest node's output log never shrinks. Thus both common prefix and future self-consistency follow directly from Lemma 3 and definition of the honest protocol. □

We additionally prove a corollary that will later be useful for proving worst-case liveness.

Corollary 3. *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Except with negligible probability over the choice of view, the following holds: for any round r, for any node i honest and online in round r, $\mathsf{linearize}(\mathsf{chain}_i^r[:-0.5\kappa])$ is a prefix of LOG_i^r where chain_i^r is node i's $\Pi_{\text{blockchain}}$ output in round r, and LOG_i^r is node i's output log to $\mathcal{Z}$ in round r.*

Proof. We ignore the negligible fraction of views where relevant bad events happen and consider only the remaining good views. By definition of the honest protocol and Lemma 3, $\overline{\mathsf{LOG}}_i^r$ must be a prefix of LOG_i^r. Thus it suffices to prove that $\mathsf{linearize}(\mathsf{chain}_i^r[:-0.5\kappa])$ is a prefix of $\overline{\mathsf{LOG}}_i^r$. If $\mathsf{chain}_i^r[:-0.5\kappa]$ is not an optimistic block of some epoch e, then by definition of the honest protocol, $\overline{\mathsf{LOG}}_i^r = \mathsf{linearize}(\mathsf{chain}_i^r[:-0.5\kappa])$. Thus henceforth we consider only the case when $\mathsf{chain}_i^r[:-0.5\kappa]$ is an optimistic block of some epoch e. In this case, $\overline{\mathsf{LOG}}_i^r = \mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ where $\mathsf{chain}_i^r[-\ell]$ is the last block before epoch e in chain_i^r and TXs is the maximal lucky sequence for epoch e in node i's view in round r. On the other hand, $\mathsf{linearize}(\mathsf{chain}_i^r[:-0.5\kappa]) = \mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs}')$ where TXs' is the maximal lucky sequence for epoch e contained in $\mathsf{chain}_i^r[:-0.5\kappa]$. Since $\mathsf{chain}_i^r[:-0.5\kappa]$ is part of node i's view in round r and by Lemma 2, it must be that $\mathsf{strip}(\mathsf{TXs}') \prec \mathsf{strip}(\mathsf{TXs})$, and the remainder of the proof is straightforward. □

3.3.2 Worst-Case Liveness

Lemma 4 (Worst-case liveness variant)**.** *Let* $\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)$ *be the chain-state function as specified in Section 3.2.3 for an arbitrary polynomial-time next-epoch function* pred*. Let* g_0 *denote the underlying* $\Pi_{\text{blockchain}}$*'s chain growth lower bound parameter. Except with negligible probability over the choice of* view*, the following holds: suppose that* $\mathcal{Z}$ *inputs a transaction* m *to an honest node when its* chain *output from* $\Pi_{\text{blockchain}}$ *has length* ℓ*, then for any honest chain* chain' *in* view *of length at least* $\ell + 2.5\kappa$*, it must hold that* m *or some* $(\_,\_,\mathsf{m})$ *exists in* $\mathsf{linearize}^{\Gamma(\kappa,\cdot,\cdot)}(\mathsf{chain}'[:\ell+2.5\kappa])$*.*

Proof. We ignore the negligible fraction of views where relevant bad events happen. For the remaining good views, the following statements hold.

By liveness of the underlying $\Pi_{\text{blockchain}}$, it must be that either m or some $(\_,\_,\mathsf{m},\_)$ is contained in $\mathsf{chain}'[:\ell+0.25\kappa]$. Let $\mathsf{chain}'[\ell_0]$ denote the first block in chain' that contains either m or some $(\_,\_,\mathsf{m},\_)$ where $\ell_0 \le \ell + 0.25\kappa$. Now consider the following cases:

- Case 1: $\mathsf{chain}'[\ell_0]$ is an interim block. By definition of linearize, it must hold that $\mathsf{linearize}(\mathsf{chain}'[:\ell_0])$ contains either m or some $(\_,\_,\mathsf{m})$. The remainder of the proof follows directly from Fact 4.
- Case 2: $\mathsf{chain}'[\ell_0]$ belongs to some epoch e. By definition of the chain-state function Γ, if $\mathsf{linearize}(\mathsf{chain}'[:\ell_0+\kappa])$ does not contain m or some $(\_,\_,\mathsf{m})$, epoch e of chain' must enter its grace period at length $\ell_0+\kappa+1$ or smaller. Thus epoch e of chain' must end at length $\ell_0+2\kappa < \ell+2.5\kappa$ or smaller. By definition of linearize, it must hold that $\mathsf{linearize}(\mathsf{chain}'[:\ell+2.5\kappa])$ must contain m or some $(\_,\_,\mathsf{m})$.

□

Theorem 12 (Worst-case liveness, restatement of Theorem 9)**.** *Let* $\Gamma(\kappa,\cdot,\cdot)$ *be the chain-state function as specified in Section 3.2.3. Let* g_0 *denote the underlying* $\Pi_{\text{blockchain}}$*'s chain growth lower bound parameter, and let* $T_{\text{confirm}}(\kappa) := \frac{3\kappa}{g_0}$*. For any p.p.t.* $(\mathcal{A},\mathcal{Z})$ *that is compliant w.r.t.* $\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for every* $\kappa\in\mathbb{N}$*, except with* $\mathsf{negl}(\kappa)$ *probability over the choice of* $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A},\mathcal{Z},\kappa)$*, the following holds: suppose that* $\mathcal{Z}$ *inputs a transaction* m *to an honest node in round* r*, then in any round* $r' \ge r + T_{\text{confirm}}(\kappa)$*, all honest and online nodes' output* LOG *to* $\mathcal{Z}$ *will contain some* $(\_,\_,\mathsf{m})$ *or* m*.*

Proof. We ignore the negligible fraction of views where relevant bad events happen. For the remaining good views, the following statements hold. Suppose that in round r, the longest honest chain is of length ℓ_r. By Lemma 4 and Fact 4, for any honest chain denoted chain in view whose length is at least $\ell_r+3\kappa$, m or some $(\_,\_,\mathsf{m})$ must exist in $\mathsf{linearize}(\mathsf{chain}[:-0.5\kappa])$. By chain growth lower bound, in round $r+T_{\text{confirm}}$ or greater, every honest chain must be of length at least $\ell_r+3\kappa$. Thus for any honest chain denoted chain' in round $r+T_{\text{confirm}}$ or greater, m or some $(\_,\_,\mathsf{m})$ must exist in $\mathsf{linearize}(\mathsf{chain}'[:-0.5\kappa])$. The remainder of the proof follows directly from Corollary 3. □

3.3.3 Optimistic Responsiveness

We now prove that Π_{ella} satisfies optimistic responsiveness in lucky epochs.

Lemma 5. *Except with negligible probability over the choice of* view*, the following holds: Suppose that* $[T_{\text{start}}, T_{\text{end}}]$ *belongs to a lucky epoch in* view *corresponding to epoch* e *and leader* i*; and let* $T_{\text{warmup}} = \frac{3\kappa}{g_0}$ *where* g_0 *is the chain growth lower bound parameter of the underlying* $\Pi_{\text{blockchain}}$*. Then, for any honest chain denoted* chain *during* $[T_{\text{start}}+T_{\text{warmup}}, T_{\text{end}}]$*,* $\mathsf{chain}[-0.5\kappa]$ *must be an optimistic block of epoch* e*.*

Proof. Henceforth we ignore the negligible fraction of views where relevant bad events happen. By our definition of lucky epoch, it holds that by the end of round $T_{\text{start}}+3\Delta$, all honest nodes will have in their view a notarized transaction for the epoch e. Let ℓ denote the length of the shortest honest chain in round T_{start}. By chain growth upper bound, no honest chain is greater than $\ell + 0.25\kappa$ in length in round $T_{\text{start}}+3\Delta$. By liveness of the underlying $\Pi_{\text{blockchain}}$, for any honest chain ch whose length is at least $\ell+0.5\kappa$ during $[T_{\text{start}}, T_{\text{end}}]$, $\mathsf{ch}[:\ell+0.5\kappa]$ must contain a notarized transaction for epoch e. Further, assuming that the signature scheme is secure, we also conclude that no honest node will have in their view a notarized transaction for any epoch $e' > e$ in or before round T_{end} since otherwise one can easily construct a reduction that breaks signature security. Thus, for any honest chain ch whose length is at least $\ell + 0.5\kappa + 2\kappa + 0.5\kappa$ during $[T_{\text{start}}, T_{\text{end}}]$, it holds that some block in $\mathsf{ch}[:-0.5\kappa]$ must have entered epoch e — since by the honest protocol, once the chain contains a notarized transaction for epoch e, it takes at most κ optimistic blocks and κ grace blocks and one interim block to enter epoch e. Further, we also conclude that no honest chain can enter epoch e before ℓ because otherwise there must exist a notarized transaction for epoch e in some honest node's view before round T_{start} — and by definition of a lucky epoch, if this happened then we could easily construct a reduction that breaks signature security.

Now, it suffices to show that for any honest chain ch during $[T_{\text{start}}, T_{\text{end}}]$ whose length is at least $\ell + 3\kappa$, $\mathsf{ch}[-0.5\kappa]$ must be an optimistic block of epoch e — since if we can show this, then the lemma follows directly due to the chain growth lower bound and the fact that the shortest chain at time T_{start} is ℓ. To show this claim, let us assume for the sake of contradiction that there exists some honest chain ch^* during $[T_{\text{start}}, T_{\text{end}}]$ whose length is at least $\ell + 3\kappa$, but $\mathsf{ch}^*[-0.5\kappa]$ is not an optimistic block of epoch e. Recall that some block in $\mathsf{ch}^*[:-0.5\kappa]$ must have entered epoch e, and if $\mathsf{ch}^*[-0.5\kappa]$ is not an optimistic block of epoch e, there must exist some grace block of epoch e in $\mathsf{ch}^*[-0.5\kappa]$. Since no honest node has in their view a notarized transaction for epoch $e' > e$ by the earlier argument, the only way to enter the grace period for epoch e is if there exists some ℓ' satisfying $\ell + \kappa \leq \ell' \leq |\mathsf{ch}| - 0.5\kappa$ and some m such that *i)* $\mathsf{linearize}(\mathsf{ch}^*[:\ell'])$ does not contain m or some $(\_, \_, \mathsf{m})$ but m or some $(\_, \_, \mathsf{m})$ appeared in $\mathsf{ch}^*[:\ell' - 0.5\kappa]$; and *ii)* $\mathsf{ch}^*[\ell']$ is an optimistic block of epoch e, and there are at least κ optimistic blocks for epoch e in $\mathsf{ch}^*[:\ell']$.

By consistency, when the leader's chain is first of length $\ell' - 0.25\kappa$ or greater — let r^* denote this round and let ch' denote the leader's chain in round r^* — it must be that m or some $(\_, \_, \mathsf{m})$ is in ch'. By growth upper bound, ch' cannot be longer than ℓ' and thus by consistency, neither m nor $(\_, \_, \mathsf{m})$ is in $\mathsf{linearize}(\mathsf{ch}'[:-\kappa])$. Notice that by chain growth upper bound, $r^* \geq T_{\text{start}}$. Thus by the end of round r^*, the leader i must have sent a notarization request for the transaction m. Thus by time $r^* + 2\Delta$, all honest nodes will have some notarized transaction $(e, s, \mathsf{m}, \_)$ in its view as well as a notarized transaction of the form $(e, s', \_, \_)$ for every $s' \leq s$. By growth upper bound, no honest chain is longer than $\ell' - 0.1\kappa$ in round $r^* + 2\Delta$. By liveness, any honest chain ch' of length at least ℓ' during $[T_{\text{start}}, T_{\text{end}}]$, $\mathsf{ch}'[:\ell']$ must contain some notarized transaction $(e, s, \mathsf{m}, \_)$ as well as a notarized transaction of the form $(e, s', \_, \_)$ for every $s' \leq s$. This means that $\mathsf{linearize}(\mathsf{ch}[:\ell'])$ must contain (e, s, m), and this contradicts our assumption. □

Theorem 13 (Optimistic case responsiveness, restatement of Theorem 10)**.** *Let g_0 be the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter and let $T_{\text{warmup}}(\kappa) = \frac{3\kappa}{g_0}$. For every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{ella}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: Suppose that $[T_{\text{start}}, T_{\text{end}}]$ belongs to a lucky epoch in view. Then, if $\mathcal{Z}$ inputs m to some honest node in some round $t \in [T_{\text{start}} + T_{\text{warmup}}, T_{\text{end}}]$, then every honest node's output LOG will include some $(\_, \_, \mathsf{m})$ or m at time $t + 3\delta$, where δ is the maximum actual network delay between*

$[T_{\text{start}} + T_{\text{warmup}}, T_{\text{end}} + 3\Delta]$ *in* view.

Proof. Suppose that $[T_{\text{start}}, T_{\text{end}}]$ corresponds to the epoch e and leader i in view. In some round $r \leq t + \delta$, the leader i first sees m in its view. We consider the following two cases — further we ignore the negligible fraction of views where relevant bad events happen and consider only the remaining good views.

- Case 1: m or some $(\_, \_, \mathsf{m})$ exists in $\mathsf{linearize}(\mathsf{chain}_i^r[: -\kappa])$. In this case, by consistency of the underlying $\Pi_{\text{blockchain}}$, for every honest chain ch in round $r + \delta$ or greater, m or some $(\_, \_, \mathsf{m})$ exists in $\mathsf{linearize}(\mathsf{ch}[: -0.5\kappa])$. The remainder of the proof follows from Corollary 3.
- Case 2: m or some $(\_, \_, \mathsf{m})$ does not exist in $\mathsf{linearize}(\mathsf{chain}_i^r[: -\kappa])$. In this case, by some round $r' \leq r$, the leader will have sent 1) a notarization request of the form (e, s, m) for some s; and 2) a notarization request of the form $(e, s', \_)$ for every $s' \leq s$. Thus in any round $s \geq r + 2\delta$, every node honest and online will have in its view 1) a notarized transaction of the form $(e, s, \mathsf{m}, \_)$; and 2) for every $s' \leq s$ a notarized transaction of the form $(e, s', \_, \_)$. By definition of the honest protocol and Lemma 5, $(\_, \_, \mathsf{m})$ exists in the output of any honest and online node in round $t + 3\delta$ or after.

□

Chapter 4

Thunderella for Permissioned

In this section, we consider two concrete instantiations of Thunderella in the permissioned setting. We show that 1) in the classical setting, assume the existence of a bare PKI and one-way functions, we can have a state machine replication protocol that tolerates arbitrarily many corruptions, and achieves responsiveness during a lucky epoch; 2) in the permissioned, sleepy setting, assume the existence of a common reference string, a bare PKI, and enhanced trapdoor permutations, we can construct a state machine replication protocol that retains worst-case security as long as the majority of online nodes are honest, and moreover achieves responsiveness during a lucky epoch.

Recall that our optimistic case conditions are more stringent than worst-case conditions, and responsiveness can only be achieved when the optimistic-case conditions hold (i.e., when a lucky epoch occurs). In this section, we shall also prove a lower bound to show that our optimistic-case conditions are in fact tight. This lower bound reflects an inherent tradeoff between the conditions necessary for worst-case security and the conditions necessary for responsive confirmation.

4.1 Permissioned, Classical Environments

Recall that in permissioned, classical environments, $\mathcal{Z}$ spawns all nodes upfront prior to protocol execution and does not spawn any nodes later; and moreover, $\mathcal{Z}$ does not issue `sleep` or `wake` instructions. In this setting, "honest" equates to "honest and online".

Worst-case security under arbitrarily many corruptions. We will show that in a permissioned, classical environment and assuming the existence of a PKI, we can have a state machine replication protocol that achieves consistency and worst-case liveness against arbitrarily many corruptions; and moreover, when "things are good", transactions can confirm in $O(1)$ actual network rounds. The concrete protocol, henceforth referred to as $\Pi_{\text{ella}}^{[\text{DS}]}$, works as follows:

1. We instantiate the Thunderella paradigm using a Dolev-Strong-based blockchain Π_{DS} as the underlying blockchain (see Section 2.4.2).

2. In the classical model, $\mathcal{Z}$ informs all honest nodes upfront the identities of all nodes participating in the protocol. Thus, we can simply have everyone be part of the committee.

3. Let n, f be parameters received from $\mathcal{Z}$ upfront where n denotes the total number of nodes and f denotes an upper bound on the number of corrupt nodes. For a transaction to be notarized, we require that at least $\lfloor \frac{n+f}{2} + 1 \rfloor$ signatures from disctinct nodes be collected. Further, the parameters (n, f) is also passed to the inner Π_{DS} instance.

4. Finally, we make a non-essential modification to our earlier scheme such that if we happen to be lucky, transactions will start confirming "instantly" starting from the very beginning of the protocol without any warmup period. Specifically, we will assume that the genesis is an optimistic block of the initial epoch (i.e., the 0-th epoch); similarly, for every $i \leq 0$, we assume by convention that $\mathsf{chain}[i]$ is an optimistic block of the initial epoch. Moreover, when the protocol begins, every honest node will act as if a $\texttt{leader}(0, 0)$ instruction has been input, i.e., node 0 will act as the leader for the initial epoch.

Regarding the last modification, in comparison, our basic protocol earlier treated the genesis block as an interim block — since in general, if we were in a permissionless setting, we may wish to run the blockchain protocol for a while and use the blockchain protocol to elect a committee (see Section 5.2).

We say that a protocol Π satisfies consistency (or optimistic responsiveness in lucky epochs resp.) in (n, f, Δ)-classical environments iff for every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects (n, f, Δ)-classical execution, Π satisfies consistency (or optimistic responsiveness in lucky epochs resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$.

Theorem 14 (Thunderella for permissioned, classical environments)**.** *For any n and any $f < n$, protocol $\Pi_{\text{ella}}^{[\text{DS}]}$ achieves 1) consistency and T_{confirm}-liveness for some $T_{\text{confirm}} = O(\kappa\Delta)$; and 2) optimistic responsiveness in lucky epochs (where lucky epoch is defined as in Section 3.2.4 for $\alpha := \frac{n-f}{n}$) in (n, f, Δ)-classical environments*[1]*.*

Proof. Straightforward from Theorems 5, 8, 9, and 10 — note also that the proofs of Theorems 8, 9, and 10 hold nonetheless with the non-essential modification mentioned earlier that makes it possible for the protocol to enter a lucky epoch immediately upon execution start. □

The above theorem immediately gives rise to the following corollary (Corollary 4) which says that if the initial leader remains honest, then the entire protocol execution is a lucky epoch — and thus responsiveness holds for the entire execution. Before we present Corollary 4, we first extend our definition of (n, f, Δ)-classical environments (n, f, Δ, S)-classical environments where $\mathcal{Z}$ is not allowed to corrupt nodes in the set S.

Definition 10. We say that some $(\mathcal{A}, \mathcal{Z})$ respects (n, f, Δ, S)-classical execution w.r.t. protocol Π for some set $S \subseteq \{0, 1, \ldots, n-1\}$ iff $(\mathcal{A}, \mathcal{Z})$ respects (n, f, Δ)-classical execution w.r.t. Π and moreover in every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the nodes in the set S remain honest.

We say that a protocol Π satisfies consistency (or responsiveness resp.) in (n, f, Δ, S)-classical environments iff for every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects (n, f, Δ, S)-classical execution, Π satisfies consistency (or responsiveness resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$. Security in $(n, \frac{n-f}{2}, \Delta)$-classical environments is similarly defined. The definitions for (n, f, Δ, S)-classical, static environments or for (n, f, Δ)-classical, static environments are similar except that now $(\mathcal{A}, \mathcal{Z})$ must additionally respect static corruption.

Corollary 4. *For any n, Δ and any $f < n$, protocol $\Pi_{\text{ella}}^{[\text{DS}]}$ satisfies consistency in (n, f, Δ)-classical environments; and satisfies responsiveness $(n, \lceil \frac{n-f}{2} - 1 \rceil, \Delta, \{0\})$-classical environments (see Section 2.5 for the definition of responsiveness).*

Proof. Arises naturally from Theorem 14 and our minor modification of the starting conditions such that the protocol can immediately enter a lucky epoch if the initial epoch happens to be lucky. □

[1] In fact, earlier we parametrized Π_{ella} with looser parameters that worked for Π_{nak} and Π_{sleepy}. It is not difficult to see that through simple reparametrizations, we can in fact achieve tighter parameters where the worst-case confirmation time $T_{\text{confirm}} = O(\Delta)$.

Another useful interpretation of the above corollary is that we can have protocols that are bimodal:

1. if a set of worst-case conditions are satisfied, then the execution preserves consistency and worst-case liveness; and
2. if a set of more stringent, optimistic-case conditions are satisfied, then the execution confirms transactions responsively throughout.

In fact, not only so, Theorem 14 says that we achieve something even stronger: not only for the initial epoch, if in any epoch a set of optimistic-case conditions are satisfied, then this corresponding epoch will start to confirm transactions responsively. Thus, in some sense, Thunderella provides a general framework in which one can try multiple times to bootstrap a fast epoch, and if in any epoch one happens to be lucky, then transactions can confirm responsively. Thunderella guarantees that the transitions between epochs as well as slow mode (i.e., falling back to the blockchain) will respect consistency.

Limits on security and performance tradeoff. So far, we have seen a tradeoff between worst-case security and the conditions required for responsive confirmation. In particular, if one hopes for consistency and worst-case liveness when a larger number of nodes are corrupt, then the conditions necessary for responsive confirmation becomes more stringent. In the following theorem, we show that in fact, such a trade-off is inherent.

Interestingly, the lower bound below (Theorem 15) is in fact a strict generalization of the $\frac{1}{3}$-corruption lower bound for responsive state machine replication stated in Section 2.5. In particular, if we plug in $n = 3f$ into the theorem below, we effectively get a slightly stronger version of Theorem 7 (see Section 2.5) — the difference is that here we additionally require that the initial leader (i.e., node 0) be honest for responsiveness whereas Theorem 7 does not. Like Theorem 7, our proof below can be generalized to a relaxed notion of responsiveness where non-responsive warmup period is allowed.

Theorem 15 (Tradeoff between security and performance)**.** *For any n and any $f < n - 1$, any polynomial function $\mathbf{T}_{\text{confirm}}$ in κ and δ and polynomial function T_{warmup} in κ, Δ, and δ, there exists a polynomial Δ in κ, such that no state machine replication protocol Π, even assuming proof-of-work, can simultaneously achieve the following: 1) satisfy $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness $(n, \lceil \frac{n-f}{2} \rceil, \Delta, \{0\})$-classical, static environments, and 2) satisfy consistency in (n, f, Δ)-classical, static environments.*

The proof of this theorem is inspired by the famous partial synchrony lower bound by Dwork et al. [19], and later extended by Pass and Shi [41] to the proof-of-work setting to show a $\frac{1}{3}$ lower bound for responsiveness.

Proof. Consider any fixed n, $f < n - 1$, T_{confirm} and T_{warmup}, Suppose for the sake of contradiction that there exists a protocol Π such that for any Δ, Π simultaneously satisfies $(T_{\text{confirm}}, T_{\text{warmup}})$-responsiveness $(n, \lceil \frac{n-f}{2} \rceil, \Delta, \{0\})$-classical, static environments; and additionally satisfies consistency in (n, f, Δ)-classical, static environments.

Now consider the following p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects (n, f, Δ)-classical execution and static corruption for some choice of Δ to be specified later: upfront $\mathcal{Z}$ spawns n nodes numbered $0, 1, \ldots, n-1$ among which $f < n - 1$ are corrupt, and moreover, node 0, i.e., the initial leader is corrupt. For the initial $T_{\text{warmup}}(\kappa, \Delta, 1)$ time, $\mathcal{Z}$ inputs a dummy transaction to all honest nodes in every round, and $\mathcal{A}$ delivers all honest messages immediately in the next round.

Now, in round $T_{\text{warmup}}(\kappa, \Delta, 1)+1$ $(\mathcal{A}, \mathcal{Z})$ divides the $n-f$ honest nodes into two camps denoted H_1 and H_2 respectively, each of size $\lfloor \frac{n-f}{2} \rfloor$ (if $n-f$ is not even then one honest node will remain and we simply discard it from consideration). Let C denote the set of corrupt nodes including the leader. At this moment, C will simulate in its head two groups of nodes C_1 and C_2 (i.e., effectively each corrupt node including the leader forks itself into two simulated instances), where C_b plays with H_b for $b \in \{1, 2\}$. However, since any corrupt node can only query the proof-of-work oracle at a bounded rate, in our attack described below, the two simulated groups C_1 and C_2 will perform work in a temporally interleaved fashion. In round $T_{\text{warmup}}(\kappa, \Delta, 1) + 1$, the group C_1 is working and the group C_1 is dormant.

Now, the environment $\mathcal{Z}$ generates two random transactions (denoted m_1 and m_2 respectively) from $\{0,1\}^\kappa$, and in every round $r > T_{\text{warmup}}(\kappa, \Delta, 1)$, $\mathcal{Z}$ always inputs m_1 to honest nodes in the first camp and inputs m_2 to honest nodes in the second camp. $\mathcal{A}$ delivers messages sent by honest nodes to other nodes in the same camp immediately in the next round; however, $\mathcal{A}$ delays honest messages in between the two camps up to the maximum delay parameter Δ (which will be set to be sufficiently large later).

Now let us consider the view of honest nodes in the camp H_1. It is not hard to see that for these honest nodes, their view in the first $T_{\text{warmup}}(\kappa, \Delta, 1) + \Delta$ rounds of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ is identically distributed as the following alternative execution $\mathsf{EXEC}^{\Pi}(\mathcal{A}', \mathcal{Z}', \kappa)$ in which $\mathcal{Z}'$ spawns n nodes, among which $\lceil \frac{n-f}{2} \rceil$ nodes are corrupt and crash in round $T_{\text{warmup}}(\kappa, \Delta, 1) + 1$, but the remaining nodes including the leader are honest. $\mathcal{A}'$ delivers messages sent by honest nodes immediately in the next round. Further, $\mathcal{Z}'$ inputs a dummy transaction for every round $r \leq T_{\text{warmup}}(\kappa, \Delta, 1)$ and for every round $r > T_{\text{warmup}}(\kappa, \Delta, 1)$, $\mathcal{Z}'$ generates a random transaction m_1 from $\{0,1\}^\kappa$ and inputs the same transaction to all honest nodes. By the responsiveness requirement, for every Δ, except with negligible probability over the choice of view of the alternative execution $\mathsf{EXEC}^{\Pi}(\mathcal{A}', \mathcal{Z}', \kappa)$, every honest node must include m_1 in their output logs by time $T_{\text{warmup}}(\kappa, \Delta, \delta = 1)+T_{\text{confirm}}(\kappa, \delta = 1)+1$ where T_{confirm} is independent of Δ. Thus regardless of the choice of Δ, except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, honest nodes in the first camp must include m_1 in their output logs by time $T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta = 1) + 1$ as well.

Now, starting in round $T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta = 1) + 1$, C_1 becomes dormant and C_2 starts playing. Whenever a node in C_2 sends a message to honest nodes in any round $r \geq T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta = 1) + 1$, the message is delivered instantly (but it appears that the message was sent in round $r - T_{\text{confirm}}(\kappa, \delta = 1)$ and got delayed for $\delta' = T_{\text{confirm}}(\kappa, \delta = 1)$ rounds). Whenever C_2 receives any message from any honest node in any round $r \geq T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta = 1) + 1$, it acts as if the message was received in $r - T_{\text{confirm}}(\kappa, \delta = 1)$. The view of the honest camp H_2 is identically distributed as the following alternate execution: the environment spawns n nodes, among which $\lceil \frac{n-f}{2} \rceil$ nodes are corrupt and crash in round $T_{\text{warmup}}(\kappa, \Delta, 1)+1$, but the remaining nodes including the leader are honest. The adversary delays messages sent by honest nodes by a maximum of $\delta' = T_{\text{confirm}}(\kappa, \delta = 1)$ — note that since C_2 started playing late, here we are charging this offset to the network delay. Similarly as before, in this alternate execution, the environment inputs a dummy transaction for every round $r \leq T_{\text{warmup}}(\kappa, \Delta, 1)$ and for every round $r > T_{\text{warmup}}(\kappa, \Delta, 1)$, it generates a random transaction m_2 from $\{0,1\}^\kappa$ and inputs the same transaction to all honest nodes. In this alternate execution, by the responsiveness requirement, for every Δ, except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, honest nodes in H_2 must include m_2 in their output logs by time $T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta') + 1$.

We now consider a sufficiently large Δ such that $\Delta > T_{\text{confirm}}(\kappa, \delta')$. In this case, since m_1 and m_2 are sampled at random from a high-entropy distribution, except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, in round $T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta') + 1$, the output

logs of honest nodes in the first camp do not include m_2 and the output logs of honest nodes in the second camp do not include m_1 since there is no information flow between the two honest camps thus far.

Summarizing the above, we conclude that except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, in round $T_{\text{warmup}}(\kappa, \Delta, 1) + T_{\text{confirm}}(\kappa, \delta') + 1$, the output logs of honest nodes in the first camp must be inconsitent with the output logs of honest nodes in the second camp — but this violates our assumption that Π satisfies consistency w.r.t. $(\mathcal{A}, \mathcal{Z})$ and thus we reach a contradiction. □

4.2 Permissioned, Sleepy Environments

In the classical model, we treat crashed nodes as corrupt, and thus crashes count towards the corruption budget. Recent works [43] considered a new model where crashed nodes are treated as "honest but sleeping" and they do not count towards the corruption budget. In such a permissioned, sleepy environment, earlier work showed the following interesting results [43]:

- First, a meaingful question to ask in this model is whether state machine replication is possible when the majority of *online* nodes are honest — this question was answered in the affirmative by earlier work [43] where they proposed a protocol called "sleepy consensus" (see Theorem 6 of Section 2.4.3).
- Earlier work [42, 43] also showed that indeed, honest majority is necessary to achieve state machine replication in a sleepy environment. Further, this lower bound arises due to the possibility of "late joining" [42, 43], i.e., an honest node can be asleep in the beginning and wake up later. As a result, unlike the permissioned, classical setting, here one cannot hope for security when arbitrarily many among those online are corrupt.

Let $\Pi_{\text{ella}}^{[\text{Sleepy}]}$ be Π_{ella} where the underlying blockchain is instantiated with Π_{sleepy} in the permissioned, sleepy setting, and the committee being all nodes spawned. We say that a protocol Π satisfies consistency (or liveness, responsiveness in lucky epochs resp.) in (n, ρ, Δ)-static, sleepy environments iff for every $(\mathcal{A}, \mathcal{Z})$ that respects (n, ρ, Δ)-sleepy execution and static corruption, Π satisfies consistency (or liveness, responsiveness in lucky epochs resp.) w.r.t. $(\mathcal{A}, \mathcal{Z})$.

Theorem 16 (Thunderella for permissioned, sleepy environments)**.** *For any n, any Δ, and any positive constant $\rho < \frac{1}{2}$, $\Pi_{\text{ella}}^{[\text{Sleepy}]}$ satisfies 1) consistency and T_{confirm}-liveness for some $T_{\text{confirm}} = O(\kappa\Delta)$; and 2) optimistic responsiveness in lucky epochs (where lucky epoch is defined as in Section 3.2.4 for $\alpha := 1 - \rho$) in (n, ρ, Δ)-static, sleepy environments.*

Proof. Straightforward from Theorems 6, 8, 9, and 10. □

We note that although the above theorem is stated for static corruption, our Thunderella paradigm can also generalize to the adaptively secure version of sleepy consensus [43] with parameters adjusted accordingly.

4.2.1 Practical Considerations for Consortium Blockchains

A consortium blockchain is one of the most important emerging applications for permissioned consensus. In a consortium blockchain, a number of banks (e.g., on the order of hundreds) would like to create a distributed ledger to allow fast inter-bank settlement and reduce back-office manual labor. Thunderella is a good fit for such consortium blockchains — particularly, its simplicity will

facilitate reconfiguration and operational maintainence. A suitable instantiation for this setting is to rely on Sleepy consensus as the underlying blockchain and use Thunderella to add a layer of voting to accelerate transaction confirmation. While the naive strategy is to have all registered consensus nodes act as the committee, it would be desirable to down-select the committee to reduce bandwidth consumption. We discuss how to achieve this below.

Bandwidth reduction and down-selection of committee. To reduce the bandwidth consumption associated with the voting step, instead of having all registered nodes act as the committee, we can down-select a committee of size $\Theta(\kappa)$ at random. The most naive approach is to randomly select a subset of the nodes to act as the committee upfront using a random oracle (assuming that users' public keys must be registered upfront before the random oracle is determined). Due to a simple application of the Chernoff bound, as long as overall the adversary controls only $\rho < \frac{1}{2} - \epsilon$ fraction of online nodes, then among the committee the adversary can only control minority as well. However, such an approach would tolerate only static corruptions since an adaptive adversary can observe which $\Theta(\kappa)$ nodes get elected as the committee, and then corrupt the committee specifically.

To tolerate adaptive corruptions (and adaptive sleepiness), we can rely on verifiable random functions (VRFs) in a similar fashion as Micali [36] — concretely, such a VRF can be instantiated in practice using a unique signature scheme and a random oracle. Let us assume that a random oracle H is chosen after all nodes' public keys are registered with the PKI. Let $\Sigma := (\mathsf{USign}, \mathsf{Vf})$ denote a *unique* signature scheme — this is a separate signature scheme than the one used to notarized transactions. Further, each node will register both public keys (one for the unique signature scheme and one for notarizing transactions) with the PKI.

- The idea is that now each sequence number (e, s) will select a different $\Theta(\kappa)$-sized committee, and the committee formation is unpredictable to the adversary in advance.

  For each sequence number (e, s), user i evaluates

  $$r := H(\mathsf{USign}_{\mathsf{sk}_i}(e, s))$$

  and if $r < D_p$, user i is eligible to vote on a transaction for the sequence number (e, s) where D_p is an appropriate difficulty parameter chosen such that the committee size (counting only those that are online) is roughly $\Theta(\kappa)$.

  It is not difficult to see that given r and pk_i, everyone can verify that i is indeed eligible to vote on (e, s).

- Users rely on a *forward-secure* signing scheme to notarize transactions. In a forward-secure signature scheme, a signing key corresponding to a certain timestamp T can only be used to sign transactions stamped with a sequence number T or greater. Further, knowing a signing key for T, a computationally bounded adversary cannot forge signatures for any sequence number smaller than T.

  Whenever nodes sign a new notarization request for the sequence number (e, s), they update the signing key to the timestamp (e, s) and then perform the signing. After notarizing a transaction with the sequence number (e, s), the signing key is immediately updated to the next timestamp $(e, s + 1)$, and the old copy immediately erased from memory. In this way, even if the adversary immediately corrupts a node that has just notarized a transaction with the sequence number (e, s), he is unable to make the node change his mind and sign an equivocating transaction for the same (e, s).

Using techniques similar to Micali [36], it is not difficult to show that assuming random oracle and the erasure model, among every committee, the majority of online committee members are honest as long as overall, the adversary controls no more than $\rho = \frac{1}{2} - \epsilon$ fraction of the online nodes (for an arbitrarily small positive constant ϵ) — and this holds even under adaptive corruptions (and adaptive sleepiness).

Chapter 5

Thunderella for Permissionless

In this section, we will consider how to instantiate the Thunderella paradigm in a permissionless framework, e.g., as the underlying consensus mechanism for a decentralized cryptocurrency. Specifically, we will discuss *i)* how to elect and rotate the committees over time (Sections 5.1, 5.2 and 5.3), and *ii)* practical considerations for leader election (Section 5.4). Throughout this section, we will also describe how to achieve fairness and incentive compatibility in a decentralized setting.

5.1 Thunderella with Robust Committee Reconfiguration

In the permissionless setting, it is not be known upfront who will participate in the protocol. We will suggest two approaches for performing committee reconfiguration: 1) use the underlying blockchain to establish the identities of recent miners [40,41], and have recently online miners form the committee; 2) stake-holders can act as the committee [3,4,7,14,15,31,36,44].

In such committee election policies, typically the committee will evolve over time. We will describe a robust committee reconfiguration framework on top of Thunderella. An important goal for our committee reconfiguration framework is to defend against *posterior corruption*. To explain why it is important to resist posterior corruption, let us consider a proof-of-stake application where stake-holders are asked to vote in the consensus mechanism. It is possible that the set of users who hold (possibly the majority of) stake sometime in the past would sell their stake at some point — from that point on, they may be incentivized to misbehave, e.g., to create a fork and double spend their old money. Thus our goal is to maintain worst-case security as long as the majority (or α fraction for a tunable parameter α) of the committee remain honest (but not necessarily online) during their term of appointment and shortly afterwards as well. However, sometime after the committee finished their term of appointment, afterwards even if the entire past committee become corrupt, the corrupt past committee should not be able to sign into the past and break the worst-case guarantees.

Roadmap. In this section, we will describe a general framework for robust committee election that defends against posterior corruption. In subsequent sections, we will describe two concrete committee rotation strategies: electing the committee either from the set of recently online miners or from recent stake.

Intuition. Recall that we would like to have a general framework for robust committee reconfiguration tolerating posterior corruption attacks. Our idea is to tag each notarized transaction with a clock number denoted c, and each clock number can have a different committee denoted

$\mathsf{committee}_c$. In our concrete realization, we will use the block length as a rough notion of time, and thus the clock number roughly corresponds to the time at which the transaction is proposed. Based on this, a notarized transaction will now be of the form $(e, s, c, c', \mathsf{m}, V)$, where

- e denotes the epoch number,
- s denotes the sequence number within the epoch,
- m denotes the transaction, and V denotes a collection of sufficiently many signatures from $\mathsf{committee}_c$ on the message $(e, s, c, c', \mathsf{m})$,
- c denotes the clock number for the current transaction, and
- c' denotes the intended clock number for the next transaction — the inclusion of c' uniquely determines the (e, s, c, c') tuple of the immediate next transaction in the lucky sequence, such that any notarized lucky sequence for an epoch must be unique.

Intuitively, this clock number c represents the approximate length of honest (and online) nodes' blockchains at the time that the leader of epoch e proposed a notarization request $(e, s, c, c', \mathsf{m})$. Additionally, we require that a notarized transaction with the clock number c be incorporated in the blockchain sufficiently soon w.r.t. c. Specifically, a notarized transaction with the clock number c must be incorporated into the blockchain by length $c + 2\kappa$ — if it is incorporated later than this, it is not considered as a *legitimate* inclusion in the blockchain and will simply be ignored. Correspondingly, we require that the committees are chosen such that each committee $\mathsf{committee}_c$ will remain honest (but not necessarily online) till the time honest chain lengths are roughly $c + 4\kappa$ — this makes sure that even if an entire past committee becomes corrupt a posteriori (i.e., after the notarized transactions get incorporated into the blockchain), they cannot overwrite the past.

Finally, when an honest node outputs a lucky sequence of notarized transactions in the optimistic mode, it will make sure that the lucky sequence being output will have enough time to be legitimately incorporated into the blockchain should a grace period ensue immediately.

We now describe our new protocol $\widetilde{\Pi}_{\text{thunder}}$ more formally. Specifically, our description will roughly follow that of Section 3.2 (where we described a protocol Π_{thunder} for the case of static committee), but we will focus on describing the new modifications on top of Π_{thunder}.

5.1.1 Protocol $\widetilde{\Pi}_{\text{thunder}}$: Consistency and Worst-Case Liveness

Notarized transactions. Notarized transactions are defined in a very similar way as in Section 3.2 except now we need to incorporate an extra clock number c and for each c there is a different committee denoted $\mathsf{committee}_c$. We say that a tuple $(e, s, c, c', \mathsf{m}, V)$ is a *notarized* transaction w.r.t. some committee of public keys denoted $\mathsf{committee}_c$ iff

- For each $(\mathsf{pk}, \sigma) \in V$, $\mathsf{pk} \in \mathsf{committee}_c$ and moreover σ is a valid signature for $(e, s, c, c', \mathsf{m})$ under pk — in this case, we also say that (pk, σ) is a *valid vote* for $(e, s, c, c', \mathsf{m})$ w.r.t. $\mathsf{committee}_c$.
- There are more than $\frac{3}{4} \cdot \left|\mathsf{committee}_c\right|$ votes in V with distinct pks (or more generally, we can also replace $\frac{3}{4}$ with $1 - \frac{\alpha}{2}$ if we wish to tolerate $1 - \alpha$ fraction of corrupt committee in the worst case)[1].

If $(e, s, c, c', \mathsf{m}, V)$ is a notarized transaction w.r.t. $\mathsf{committee}_c$, we also say that V is a *valid notarization* for $(e, s, c, c', \mathsf{m})$ w.r.t. $\mathsf{committee}_c$.

[1]Note that by this definition, if $\mathsf{committee}_c = \emptyset$, no transaction can be notarized w.r.t. to $\mathsf{committee}_c$.

Lucky sequence. To define the notion of a lucky sequence, we make the additional assumption that each epoch number e defines the starting clock number, i.e.,

$$e := (\widetilde{e}, c_0)$$

where $\widetilde{e}$ denotes the actual epoch number and c_0 is the starting clock number.

A sequence $\{e_i, s_i, c_i, c_i', \mathsf{m}_i, \_\}_{i \in \{1,2,\ldots,\ell\}}$ is said to be a lucky sequence iff c_1 agrees with the starting clock number for epoch e, and moreover, for all $i \in [\ell - 1]$, $e_i = e_{i+1}$, $s_i = i$, and $c_{i+1} = c_i'$. In other words, like before, we still require that the sequence numbers increment one by one; but here we additionally require that the next clock number c_{i+1} should always agree with the c_i' value declared in the previous transaction,

Note that the above definition ensures the uniqueness of the lucky sequence as long as every $\mathsf{committee}_c$ has sufficiently many nodes that remain honest sufficiently long. In particular, it is not possible to have two committees that are both eligible for notarizing a certain sequence number s, since inductively, as long as the lucky sequence is unique up to sequence number $s - 1$, then the next clock number will be uniquely determined (this will be proved formally later).

Chain linearization. The chain linearization algorithm $\mathsf{linearize}$ is defined almost identically as in Section 3.2 (using the new notion of lucky sequence), and with the following modification:

- Any transaction of the form $(e, s, c, c', \mathsf{m}, V)$ can be *legitimately included* in a blockchain in between lengths $[c, c + 2\kappa]$ — if some transaction of this form appears before length c or after length $c + 2\kappa$, it is simply ignored.

  Later in our compliance rules, we will state the requirement that with probability 1, if an honest and online node's chain is of length c or greater, then the honest node must have recorded $\mathsf{committee}_c$ and moreover all honest and online nodes record the same $\mathsf{committee}_c$ for every c — this way, the validity of any $(e, s, c, c', \mathsf{m}, V)$ tuple in any honest chain is unambiguously defined in every view.

- Further, we modify the definition of the $\mathsf{strip}(\cdot)$ function accordingly, such that $\mathsf{strip}(e, s, c, c', \mathsf{m}, V) := (e, s, c, c', \mathsf{m})$.

Protocol $\widetilde{\Pi}^{\Gamma(\cdot,\cdot,\cdot)}_{\text{thunder}}$. We now describe our new protocol $\widetilde{\Pi}_{\text{thunder}}$ that provides a committee reconfiguration framework. In particular, we highlight some important new modifications in blue. Like before, although not explicitly noted, we assume that whenever an honest node receives any message on the network, if the message has not been broadcast before, the honest node broadcasts the message.

- *Initialize.* Fork an instance of $\Pi_{\text{blockchain}}$ with appropriate parameters. Unless otherwise noted, all messages sent from the $\Pi_{\text{blockchain}}$ instance or destined for $\Pi_{\text{blockchain}}$ are automatically passed through, but these messages also count towards the view of the current $\widetilde{\Pi}_{\text{thunder}}$ protocol instance.

- *Committee selection.* On receiving $\mathtt{elect}(c, \mathsf{committee}_c)$ from $\mathcal{Z}$, if no committee has been recorded for clock number c earlier, record $\mathsf{committee}_c$ as the committee for c.

- *Notarize.* Upon receiving notarization request $(e, s, c, c', \mathsf{m})$ from $\mathcal{Z}$: if $\mathsf{committee}_c$ has been recorded and $\mathsf{pk} \in \mathsf{committee}_c$, and further no signature has been produced for (e, s) earlier, then compute $\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(e, s, c, c', \mathsf{m})$ and broadcast $((e, s, c, c', \mathsf{m}), \sigma)$.

- *Propose.* Every round, let chain be the output from the $\Pi_{\text{blockchain}}$ instance.

  - Let TXs be a set containing 1) every notarized transaction[2] $(e, s, c, c', \mathsf{m}, V)$ in the node's view such that no notarized transaction $(e, s, c, c', \mathsf{m}, \_)$ was legitimately included in $\mathsf{chain}[: -0.5\kappa]$; and 2) every unnotarized transaction m in the node's view such that no m or notarized transaction $(e, s, c, c', \mathsf{m}, \_)$ was legitimately included in $\mathsf{chain}[: -0.5\kappa]$.
  - Additionally, a node can generate a signing key pair $(\mathsf{pk}, \mathsf{sk})$ and include pk in the block payload as well (such that the environment $\mathcal{Z}$ can choose the committees based on observed public keys). The corresponding secret key sk is kept secret.
  - Propose $\mathsf{TXs}||\mathsf{pk}$ to $\Pi_{\text{blockchain}}$.

- *Output.* In every round, let chain be the output from $\Pi_{\text{blockchain}}$.
  - If $\mathsf{chain}[-0.5\kappa]$ is an optimistic block belonging to epoch e:
    a) let $\mathsf{chain}[-\ell]$ be the starting block for epoch e in chain where $\ell \geq 0.5\kappa$.
    b) extract the maximal lucky sequence TXs for epoch e from the node's view so far such that for every $(e, s, c, c', \mathsf{m}, V) \in \mathsf{TXs}$: either some valid $(e, s, c, c', \mathsf{m}, \_)$ is already legitimately contained in $\mathsf{chain}[: -0.5\kappa]$, or $c + 2\kappa \geq |\mathsf{chain}| + 0.5\kappa$, i.e., the clock number c must be large enough for the notarized transaction to be picked up by the grace period should $\mathsf{chain}[-0.5\kappa + 1]$ enter a grace period.
    c) let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[: -(\ell + 1)])||\mathsf{strip}(\mathsf{TXs})$.
  - Else, let $\overline{\mathsf{LOG}} := \mathsf{linearize}(\mathsf{chain}[: -0.5\kappa])$.
  - Let LOG be the previous output to $\mathcal{Z}$: if $\overline{\mathsf{LOG}}$ is longer than LOG, output $\overline{\mathsf{LOG}}$; else output LOG to $\mathcal{Z}$.

- *Mempool.* Upon receiving any other message from the network or $\mathcal{Z}$, record the tuple.

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ iff

- *Blockchain compliance.* $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{blockchain}}$;

- *Consistency and timeliness of committee selection.* For every κ, in every view in the support of $\mathsf{EXEC}^{\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, for every $c \in \mathbb{N}$, if in some round an honest and online node's chain is of length $c - 0.25\kappa$ or greater, then the node must have recorded $\mathsf{committee}_c$; and moreover, all honest nodes must record the same $\mathsf{committee}_c$.

- *Resilience.* For every κ, in every view in the support of $\mathsf{EXEC}^{\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A}, \mathcal{Z}, \kappa)$, for every clock number c, more than $\frac{1}{2}$ fraction (or in general, more than α fraction) of the distinct public keys in $\mathsf{committee}_c$ are output by nodes that remain honest (but not necessarily online) till $\mathsf{len2time}(c + 4\kappa, \mathsf{view})$ where $\mathsf{len2time}(\ell, \mathsf{view})$ denotes the earliest round in which the shortest honest chain is at least ℓ in length.

Importantly, here we require that the underlying blockchain be secure, and moreover, *the majority of every committee must remain honest (but not necessarily online) during its term of appointment, and shortly afterwards as well.* We show that $\widetilde{\Pi}_{\text{thunder}}$ satisfies consistency and worst-case liveness in compliant executions. In particular, these worst-case guarantees hold even when some past committee can be entirely corrupt at a future point of time, i.e., our $\widetilde{\Pi}_{\text{thunder}}$ scheme tolerates

[2] If $\mathsf{committee}_c$ has not been recorded by an honest node, then any tuple $(\_, \_, c, \_, \_, \_)$ is not considered as a notarized transaction at this moment (but may later become notarized when $\mathsf{committee}_c$ has been recorded.

posterior corruption. As earlier works noted [15, 30, 36, 41] formally or informally, the ability to tolerate posterior corruption is important in proof-of-stake contexts where stake-holders are asked to vote on transactions.

We now formally state our worst-case guarantees in compliant executions in the theorems below. The proofs of these theorems are deferred to Section 5.1.3.

Theorem 17 (Consistency)**.** *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Then, $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ satisfies consistency as defined in Section 2.2 w.r.t. any p.p.t. $(\mathcal{A},\mathcal{Z})$ that is compliant w.r.t. $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$.*

Chain state function. Consistency holds for any admissible chain-state function. To prove worst-case liveness, we instantiate $\Gamma^{\mathsf{pred}}(\cdot,\cdot,\cdot)$ with the concrete chain state function described in Section 3.2.3. At this moment, we leave pred unspecified, since our worst-case liveness guarantees hold regardless of what pred is.

Theorem 18 (Worst-case liveness)**.** *Let $\Gamma(\kappa,\cdot,\cdot) := \Gamma^{\mathsf{pred}}(\cdot,\cdot,\cdot)$ be the chain-state function as specified in Section 3.2.3 for any polynomial-time boolean predicate pred. Let g_0 denote the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter, and let $T_{\text{confirm}}(\kappa) := \frac{3\kappa}{g_0}$. For any p.p.t. $(\mathcal{A},\mathcal{Z})$ that is compliant w.r.t. $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa\in\mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view}\leftarrow \mathit{EXEC}^{\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}}(\mathcal{A},\mathcal{Z},\kappa)$, the following holds: suppose that $\mathcal{Z}$ inputs a transaction m to an honest node in round r, then in any round $r'\geq r+T_{\text{confirm}}(\kappa)$, all honest and online nodes' output LOG to $\mathcal{Z}$ will contain some $(\_,\_,\_,\_,\mathsf{m})$ or m.*

5.1.2 Protocol $\widetilde{\Pi}_{\text{ella}}$: Optimistic Responsiveness

We now describe the leader-based coordination mechanism for ensuring responsiveness during lucky epochs. This part is captured in protocol $\widetilde{\Pi}_{\text{ella}}$.

Description of $\widetilde{\Pi}_{\text{ella}}$. $\widetilde{\Pi}_{\text{ella}}$ calls $\widetilde{\Pi}_{\text{thunder}}^{\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)}$. We spell out the next-epoch function pred and the rest of $\widetilde{\Pi}_{\text{ella}}$ below. Most of the protocol is similar to our earlier Π_{ella} which works for a static committee. We will highlight the important differences in blue.

- *Next-epoch function.* The policy function $\mathsf{pred}(\mathsf{chain},e)$ takes in an abstract blockchain denoted chain and an epoch number e. If there exists a notarized transaction for epoch e in chain, then output 1; else output 0. Notice that our definition here depends on $\mathsf{committee}_c$ for each c of concern — later we shall mention that in a compliant execution, all honest nodes record the same $\mathsf{committee}_c$ for every c and thus the definition is unambiguous.
- *Initialize*: fork an instance of the $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ protocol.
- *Leader switch*: upon input $\texttt{leader}(e,i)$: parse $e := (\widetilde{e},c)$ where c denotes the starting clock number, if no leader has been recorded for epoch $\widetilde{e}$, record the tuple (e,i), and do the following:
  - if current node is i: wait till the chain output from $\Pi_{\text{blockchain}}$ is of length at least c, and then send a notarization request for a special epoch-start transaction $(e,s=1,c,c',\texttt{start})$ where c is the starting clock number as mentioned above and c' is the node's current chain length; and let $s=2$;

  - for every notarization request $(e, s, c, c', \mathsf{m})$ received earlier from node i, act as if $(e, s, c, c', \mathsf{m})$ has just been received from i.

- *Notarization*: upon receiving notarization request $(e, s, c, c', \mathsf{m})$ from i: if i has been recorded as the leader for epoch e, forward the notarization request $(e, s, c, c', \mathsf{m})$ to $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$; else ignore the request.

- *Leader*: every round: let e be the largest epoch recorded thus far; if current node is recorded as the leader for epoch e and moreover, a start request of the form $(e, s = 1, \_, \_, \texttt{start})$ has been broadcast:

  - for every m in view such that no m or $(\_, \_, \_, \_, \mathsf{m})$ appears in $\mathsf{linearize}(\mathsf{chain}[: -\kappa])$, if a notarization request has not been broadcast for m earlier, then broadcast a notarization request for m — we will describe below how this is done.
  - if the previous notarization request broadcast contained a c' value that is smaller than the current chain length, then broadcast a notarization request for a dummy transaction $\mathtt{dummy}||\mathsf{nonce}$ for a freshly chosen nonce.
  - To broadcast a notarization request for any m, do the following: 1) broadcast the tuple $(e, s, \widetilde{c}, \widetilde{c}', \mathsf{m})$ where $\widetilde{c}$ is the c' value declared in the previous notarization request, and where the new $\widetilde{c}'$ is equal to the current chain length; and 2) let $s := s + 1$.

- *Other messages*: pass through all other messages between $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and $\mathcal{Z}$; similarly pass through all other messages between $\widetilde{\Pi}_{\text{thunder}}^{\Gamma(\kappa,\cdot,\cdot)}$ and the network.

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\widetilde{\Pi}_{\text{ella}}$ iff $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\widetilde{\Pi}_{\text{thunder}}^{\Gamma^{\mathsf{pred}}(\kappa,\cdot,\cdot)}$ where $\Gamma^{\mathsf{pred}}(\kappa, \cdot, \cdot)$ is now concretely instantiated as mentioned above.

Lucky epoch. We now formally define the notion of a lucky epoch. Similarly as before, we will prove that whenever there is a lucky epoch, after a short warmup time, all transactions will confirm responsively (except for a negligible fraction of views). Formally, we say that $[T_{\text{start}}, T_{\text{end}}]$ is a lucky epoch in view iff the following hold:

- In any round $r \geq T_{\text{start}} + \Delta$, any honest and online node should have received the same instruction $\texttt{leader}(e, i)$. Further, prior to T_{start}, no honest node has received from $\mathcal{Z}$ any $\texttt{leader}(e', \_)$ where e' is the same or a larger epoch than e.

- Let c_0 denote the starting clock number for epoch e, then the leader i's chain must be no shorter than $c_0 - \kappa$ in round T_{start}.

- The leader (i.e., node i) is honest and online at in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$;

- For every $c \geq c_0$ such that $\mathsf{committee}_c$ was ever input to any honest node in or before T_{end}, it must be that $\mathsf{committee}_c$ is non-empty and more than $\frac{3}{4}$ fraction (or in general, more than $1 - \frac{\alpha}{2}$ fraction) of $\mathsf{committee}_c$ are honest and online in any round during $[\mathsf{len2time}(c - 0.5\kappa), \mathsf{len2time}(c + 0.5\kappa)]$.

The following theorem states that our $\widetilde{\Pi}_{\text{ella}}$ achieves optimistic responsiveness in lucky epochs after a short warmup time. Its proof is deferred to Section 5.1.3.

Theorem 19 (Optimistic responsiveness in lucky epochs)**.** *Let g_0 be the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter. For every p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\widetilde{\Pi}_{\text{ella}}$, $\widetilde{\Pi}_{\text{ella}}$ satisfies $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs for $T_{\text{warmup}} = O(\frac{\kappa}{g_0})$, and $T_{\text{opt}} = 3\delta$ where δ is the actual maximum network delay in* view*.*

5.1.3 Proofs: Robust Committee Reconfiguration Framework

Worst-Case Guarantees

Henceforth in this subsection, whenever we say that "except with negligible probability over the choice of view, some property ev(view) holds", we formally mean that for any $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\widetilde{\Pi}^{\Gamma(\kappa,\cdot,\cdot)}_{\text{thunder}}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\widetilde{\Pi}^{\Gamma(\kappa,\cdot,\cdot)}_{\text{thunder}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, $\mathsf{ev}(\mathsf{view})$ holds.

Lemma 6 (Uniqueness)**.** *Assume that the signature scheme is secure, then except with negligible probability over the choice of* view*, the following holds. For any (e, s, c) tuple, if $(e, s, c, c', \mathsf{m}, V)$ is a notarized transaction in some honest node's view by $\mathsf{len2time}(c + 4\kappa, \mathsf{view})$, and similarly $(e, s, \widetilde{c}, \widetilde{c}', \widetilde{\mathsf{m}}, \widetilde{V})$ is also a notarized transaction in some honest node's view by $\mathsf{len2time}(c+4\kappa, \mathsf{view})$, then it must be the case that $(c', \mathsf{m}) = (\widetilde{c}', \widetilde{\mathsf{m}})$.*

Proof. Similar to the earlier proof of Lemma 2, except that now we additionally rely on the fact that α fraction of $\mathsf{committee}_c$ will remain honest till $\mathsf{len2time}(c + 4\kappa, \mathsf{view})$. □

Henceforth, we say that an honest and online node i *promptly outputs* a notarized transaction $(e, s, c, c', \mathsf{m}, \_)$ in round r in view, if it included $(e, s, c, c', \mathsf{m})$ in its output in round r, but $(e, s, c, c', \mathsf{m}) \notin \mathsf{chain}^r_i[: -0.5\kappa]$.

Lemma 7. *Assume that the signature scheme is secure. Except with negligible probability over the choice of* view*, the following holds: for any epoch e, any sequence number s, if for $b \in \{0, 1\}$, the notarized transaction $(e, s, c, c'_b, \mathsf{m}_b, V_b)$ either is legitimately included in some honest chain in* view*, or is promptly output by some honest (and online) node in* view*, then it holds that*

(a) for $b \in \{0, 1\}$, $(e, s, c, c'_b, \mathsf{m}_b, V_b)$ must have appeared in view *before round $\mathsf{len2time}(c+4\kappa, \mathsf{view})$; and*

(b) $(e, s, c, c'_0, \mathsf{m}_0) = (e, s, c, c'_1, \mathsf{m}_1)$.

Proof. To show this, by Lemma 6 and definition of lucky sequence, it suffices to show the following ignoring the negligible fraction of views where relevant bad events happen:

1. Any notarized transaction with the clock number c that legitimately appears in some honest chain ch must appear in view before $\mathsf{len2time}(c + 4\kappa, \mathsf{view})$. For any notarized transaction contained in $\mathsf{ch}[\ell^*]$, its clock number $c \geq \ell^* - 2\kappa$. Since $\mathsf{len2time}(\ell^* + 2\kappa) \leq \mathsf{len2time}(c + 4\kappa)$, it suffices to show that any notarized transaction that appears in $\mathsf{ch}[\ell^*]$ must appear in view before $\mathsf{len2time}(\ell^* + 2\kappa, \mathsf{view})$ — and this follows from consistency of the underlying $\Pi_{\text{blockchain}}$.

2. For any notarized transaction with the clock number c cannot be promptly output by any honest (and online) node in or after $\mathsf{len2time}(c+4\kappa, \mathsf{view})$. Suppose that some honest (and online) node i promptly output some notarized transaction with the clock number c in round r. It must hold that $c \geq |\mathsf{chain}^r_i| - 1.5\kappa$. By chain growth upper bound, we have that $r \leq \mathsf{len2time}(c + 4\kappa)$.

□

We now prove the counterpart of Lemma 3 but for our new protocol $\widetilde{\Pi}_{\text{thunder}}$.

Lemma 8. *Let $\Gamma(\kappa,\cdot,\cdot)$ be any admissible chain-state function. Except with negligible probability over the choice of* view*, the following holds: for any r and t, for any node i honest and online in round r and any node j honest and online in round t, either $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$ or $\overline{\mathsf{LOG}}_j^t \prec \overline{\mathsf{LOG}}_i^r$.*

Proof. In every round, an honest node's $\overline{\mathsf{LOG}}$ is of the generalized form $\mathsf{linearize}(\mathsf{chain}[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ where

- either TXs is empty and $\ell = 0.5\kappa$ or
- TXs is a non-empty lucky sequence for some epoch e and $\ell \geq 0.5\kappa$.

Suppose that $\overline{\mathsf{LOG}}_i^r := \mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ and $\overline{\mathsf{LOG}}_j^t := \mathsf{linearize}(\mathsf{chain}_j^t[:-\ell'])||\mathsf{strip}(\mathsf{TXs}')$ are output logs by node i in round r and by node j in round t respectively in view. Henceforth we ignore the negligible fraction of views where relevant bad events happen. For the remaining good views, the following statements hold.

By consistency of $\Pi_{\text{blockchain}}$, either $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$ or $\mathsf{chain}_j^t[:-\ell'] \prec \mathsf{chain}_i^r[:-\ell]$. Without loss of generality, henceforth we assume that $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$. We now consider the following cases:

- Case 1: TXs is empty. In this case, it follows directly from Fact 4 that $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$.
- Case 2: TXs is non-empty. In this case, $\mathsf{chain}_i^r[-\ell+1]$ must be the starting block of some epoch e, and $\mathsf{chain}_i^r[-0.5\kappa]$ must be an optimistic block of epoch e. We now consider the following sub-cases:
  - Case 2a: Epoch e has completed in $\mathsf{chain}_j^t[:-\ell']$, i.e., there are two adjacent blocks such that the former belongs to epoch e and the latter does not in $\mathsf{chain}_j^t[:-\ell']$. Let $\mathsf{chain}_j^t[-\ell^*]$ denote the last block of epoch e in $\mathsf{chain}_j^t[:-\ell']$. By Fact 4, it suffices to prove that $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs}) \prec \mathsf{linearize}(\mathsf{chain}_j^t[:-\ell^*])$.

    Let $\ell_i^r := |\mathsf{chain}_i^r|$ be the length of node i's chain in round r. By definition of the honest protocol, TXs must be a lucky sequence for epoch e in node i's view in round r, and further, for every notarized transaction $(e,s,c,c',m,V) \in \mathsf{TXs}$, unless some $(e,s,c,c',m,\_)$ is already legitimately contained in $\mathsf{chain}_i^r[:-0.5\kappa]$ it must be that $c+2\kappa \geq \ell_i^r + 0.5\kappa$, i.e., $c \geq \ell_i^r - 1.5\kappa$. By round $r+\Delta$, TXs must exist in any honest node's view. By the admissibility of the chain-state function Γ and the fact that $\mathsf{chain}_i^r[-0.5\kappa]$ is an optmistic block of epoch e, $\mathsf{chain}_j^t[:-\ell^*]$ must be at least 0.5κ longer than chain_i^r. By chain growth upper bound, by round $r+\Delta$, any honest chain must be at most $\ell_i^r + 0.25\kappa$ in length. By liveness, for any honest chain denoted ch in view whose length is at least $\ell_i^r + 0.5\kappa$, all notarized transactions in TXs must appear in $\mathsf{ch}[:\ell_i^r + 0.5\kappa]$ (and thus $\mathsf{chain}_j^t[:-\ell^*]$ must include TXs). We now prove that they are in fact *legitimately* included in $\mathsf{ch}[:\ell_i^r + 0.5\kappa]$ (i.e., the clock numbers of notarized transactions in TXs are not too early w.r.t. where they are contained in ch.) This is true since we know that any notarized transaction in TXs that is not already legitimately contained in $\mathsf{chain}_i^r[:-0.5\kappa]$ has a clock number $c \leq \ell_i^r - 1.5\kappa$; and further, by consistency $\mathsf{chain}_i^r[:-0.5\kappa] \prec \mathsf{ch}[:\ell_i^r + 0.5\kappa]$.

    The remainder of the proof follows directly from definition of $\mathsf{linearize}$ and Lemma 7.

– Case 2b: Epoch e has not completed in $\mathsf{chain}_j^t[:-\ell']$. Recall that node i outputs $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs})$ in round r. By definition of the honest algorithm, and since $\mathsf{chain}_i^r[:-\ell] \prec \mathsf{chain}_j^t[:-\ell']$, node j's output in round t must be of the form $\mathsf{linearize}(\mathsf{chain}_i^r[:-\ell])||\mathsf{strip}(\mathsf{TXs}^*)$ where TXs^* is either empty or some lucky sequence for epoch e in node j's view in round t such that any notarized transaction in TXs^* must either have been legitimately included in $\mathsf{chain}_j^t[:-0.5\kappa]$ or it is "promptly output" by j in round t. By Lemma 7, it holds that either $\overline{\mathsf{LOG}}_i^r \prec \overline{\mathsf{LOG}}_j^t$ or $\overline{\mathsf{LOG}}_j^t \prec \overline{\mathsf{LOG}}_i^r$.

□

Proof of consistency. Theorem 17 now follows in a straightforward fashion since by constructions, honest nodes' output logs never shrink. Thus both common prefix and future self-consistency follow from Lemma 8 and definition of the honest protocol.

Proof of worst-case liveness. The proof of worst-case liveness is almost identical to the proof in Section 3.3.2 (except that now in proving the counterpart of Corollary 3, we apply Lemma 7 in place of Lemma 2). We thus omit repeating essentially the same proof and refer the reader to Section 3.3.2.

Optimistic Responsiveness

We now prove the optimistic responsiveness of our protocol $\widetilde{\Pi}_{\text{ella}}$ in lucky epochs. Henceforth in this subsection, whenever we say that "except with negligible probability over the choice of view, some property $\mathsf{ev}(\mathsf{view})$ holds", we formally mean that for any $(\mathcal{A}, \mathcal{Z})$ that is compliant w.r.t. $\widetilde{\Pi}_{\text{ella}}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\widetilde{\Pi}_{\text{ella}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, $\mathsf{ev}(\mathsf{view})$ holds.

Fact 5. Except with negligible probability over the choice of view, the following holds: suppose $[T_{\text{start}}, T_{\text{end}}]$ belongs to a lucky epoch e in view in which i is the leader. Then, if node i broadcasts a notarization request of the form $(e, s, c, c', \mathsf{m})$ in round $r \in [T_{\text{start}}, T_{\text{end}}]$, it holds that

(a) In any round $r' \geq r + 3\delta$, all honest nodes will have obtained a valid notarization V for the tuple $(e, s, c, c', \mathsf{m})$; and

(b) For any honest chain denoted ch in view of length $c + 0.25\kappa$, $\mathsf{ch}[:c + 0.25\kappa]$ legitimately includes some notarized transaction of the form $(e, s, c, c', \mathsf{m}, V)$.

Proof. Henceforth we ignore the negligible fraction of views where relevant bad events take place. By honest protocol definition, if the leader i proposes some notarization request of the form $(e, s, c, c', \mathsf{m})$ in round $r \in [T_{\text{start}}, T_{\text{end}}]$, it holds that the leader's chain must be of length $c - 1$ in round $r - 1$. By chain growth upper bound, in round r, all honest and online nodes' chains must be of length at least $c - 0.25\kappa$. Further, in round $r + \delta$, all honest and online nodes' chains must be of length at most $c + 0.25\kappa$. By our compliance rules, in any round $r' \geq r$, any honest and online node must have recorded the same tuple $\mathsf{committee}_c$. By definition of a lucky epoch and definition of honest protocol, it holds that more than $\frac{3}{4}$ fraction of $\mathsf{committee}_c$ remain honest and online during $[r, r + \delta]$. By honest protocol definition, it is not hard to see that claim (a) holds. Now claim (b) also follows by liveness and definition of "legitimately include". □

We now prove the equivalent of Lemma 5 but now for our new $\widetilde{\Pi}_{\text{ella}}$.

Lemma 9. *Except with negligible probability over the choice of* view*, the following holds: Suppose that* $[T_{\text{start}}, T_{\text{end}}]$ *belongs to a lucky epoch in* view *corresponding to epoch* e *and leader* i*; and let* $T_{\text{warmup}} = \frac{4\kappa}{g_0}$ *where* g_0 *is the chain growth lower bound parameter of the underlying* $\Pi_{\text{blockchain}}$*. Then, for any honest chain denoted* chain *during* $[T_{\text{start}} + T_{\text{warmup}}, T_{\text{end}}]$*,* $\mathsf{chain}[-0.5\kappa]$ *must be an optimistic block of epoch* e*.*

Proof. The proof would have been the same as that of Lemma 5 if in round T_{start}, the leader's chain were exactly of length c_0 where c_0 is the starting epoch number for epoch c. In particular, due to Lemma 7 and Fact 5, every argument made in the proof of Lemma 5 would still follow.

Now, taking into account that our new "lucky epoch" definition permits the leader's chain to be $c_0 - \kappa$ or longer during round T_{start}. We can simply redo the proof of Lemma 7 by replacing T_{start} in the proof with $\widetilde{T}_{\text{start}}$ where $\widetilde{T}_{\text{start}}$ is the earliest round in view in which the leader's chain is of length c_0 or greater. By growth lower bound, it holds that $\widetilde{T}_{\text{start}} \leq T_{\text{start}} + \kappa/g_0$ — note that in our new lemma here, the T_{warmup} is redefined as $4\kappa/g_0$, and the extra κ/g_0 accounts for the difference between $\widetilde{T}_{\text{start}}$ and T_{start}. □

Proof of optimistic responsiveness. The remainder of the proof for optimistic responsiveness would follow in the same manner as the proof for Π_{ella} (for a static committee), but where the new $T_{\text{warmup}} = 4\kappa/g_0$.

5.2 Recent Blockchain Miners As Committee

So far, we have described a general framework for committee reconfiguration but we have not described exactly what committee election policy would be suitable. In this section, we will describe one concrete strategy that leverages the underlying blockchain to elect a set of recently online blockchain miners as the committee. Henceforth in this section, we assume that the underlying blockchain is a proof-of-work blockchain (the mechanism also applies to proof-of-stake blockchains, but for proof-of-stake blockchains simpler mechanisms exist for electing a stake-based committee). We would like to guarantee that as long as the adversary controls ρ fraction of overall nodes (i.e., computational power), then it must be the case that roughly $1 - \rho$ fraction of the committee will remain honest sufficiently long, i.e., till sometime after its term of appointment. We will formally prove this property (which is required by our earlier committee rotation framework $\widetilde{\Pi}_{\text{ella}}$) under the assumption of a mildly adaptive adversary. Specifically, a τ-mildly adaptive adversary is a relaxation of a fully adaptive adversary since it takes at least τ time to adaptively corrupt nodes.

5.2.1 Preliminary: Mildly Adaptive Corruptions

Let τ, τ' be functions in κ. In a τ-mildly adaptive corruption model, roughly speaking, it takes the adversary at least $\tau(\kappa)$ rounds to corrupt an honest node, where τ is referred to as the agility parameter [41]. Similarly, we can a corresponding agility parameter for sleepiness too called $\tau'(\kappa)$. In other words, it takes the adversary $\tau'(\kappa)$ rounds to put an honest node to sleep.

To formally model this, we can bake this into the underlying model — but to do it more cleanly in our general modeling framework, we choose to model mildly adaptive corruption as an additional constraint on $(\mathcal{A}, \mathcal{Z})$. Formally, we say that $(\mathcal{A}, \mathcal{Z})$ respects $(n, \rho, \Delta, \tau, \tau')$-permissionless execution w.r.t. some protocol Π iff for every κ, for every view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold:

1. $(\mathcal{A}, \mathcal{Z})$ respects Δ-bounded network delay w.r.t. Π;

2. if $\mathcal{Z}$ issues `corrupt` to any honest node i in round r in view, in some round $r' \leq r - \tau(\kappa)$, $\mathcal{Z}$ must output `precorrupt`(i). Note that the adversary receives node i's internal states and gets to control node i when the actual `corrupt` instruction is issued to i.

3. if $\mathcal{Z}$ issues `sleep` to any honest node i in round r in view, in some round $r' \leq r - \tau'(\kappa)$, $\mathcal{Z}$ must output `presleep`(i). The node actually falls asleep upon receiving `sleep`.

4. in every round, the total number of precorrupt nodes, corrupt nodes, and honest and online nodes equals to n where precorrupt nodes include every i such that $\mathcal{Z}$ has said `precorrupt`(i) but i has not become corrupt yet.

5. in every round, the total number of precorrupt nodes and corrupt nodes is less than ρn.

6. $\mathcal{Z}$ informs all honest nodes of the parameters (n, ρ, Δ) upon spawning.

In particular, in the above definition, if the agility parameter $\tau = 0$, τ-mildly adaptive is equivalent to fully adaptive. On the other hand, if $\tau = \infty$, then τ-mildly adaptive is equivalent to static corruption. Similarly if $\tau' = 0$, then we model adaptive sleepiness.

Terminology. Henceforth, if $\mathcal{Z}$ has not output `precorrupt`(i), then we say that i is *intact*. If $\mathcal{Z}$ has not output `corrupt`(i), we say that i is *honest* — basically honest nodes include those that are intact (i.e., have not received `precorrupt`) or those that are precorrupt but have not actually become corrupt. When $\tau = 0$, intact and honest mean the same.

5.2.2 Preliminary: Committee Election in a Permissionless Environment

In a permissionless environment, the set of nodes who participate in the protocol are not known in advance. It would be desirable to have a protocol that allows us to elect a good committee in a permissionless environment, where a committee is good iff the fraction of corrupt (and precorrupt) nodes in the committee is not too much larger than the fraction of overall corruption ρ. A recent work called Fruitchains [40] showed a protocol for achieving this above goal relying on any abstract blockchain protocol. Fruitchains [40] shows that given any blockchain protocol with positive chain quality, we can construct a blockchain protocol where for every block length, we can "elect a committee" among which, informally speaking, at most $(1+\epsilon)\rho$ fraction can be corrupt (or precorrupt) where ϵ is an arbitrarily small constant and ρ is the overall percentage of corrupt nodes. In other words, if the adversary controls at most ρ fraction of corrupt (and precorrupt) nodes, he cannot increase his representation in the elected committee by more than any arbitrarily small ϵ fraction. We will formalize this property as *fruit quality*. We define fruit quality in the more general model that allows mildly adaptive corruptions as described in Section 5.2.1.

Fruit quality. Henceforth, given a blockchain protocol $\Pi_{\mathsf{blockchain}}$ and an efficiently computable function fruits, we say that the pair $(\Pi_{\mathsf{blockchain}}, \mathsf{fruits})$ satisfies (K_f, L_f, μ_f) fruit quality w.r.t. $(\mathcal{A}, \mathcal{Z})$ iff there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, except with negligible probability over the choice of $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\mathsf{blockchain}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds:

- Given any honest chain denoted chain in view, for every length $\ell \geq K_f$ that is not too short, it holds that $\mathsf{fruits}(\mathsf{chain}[:\ell])$ outputs a set of "fruit records" such that more than μ_f fraction of the fruit records are *intact* at most L_f blocks ago w.r.t. $\mathsf{chain}[:\ell]$ — here we say that a fruit record f is *intact* at most L_f blocks ago w.r.t. $\mathsf{chain}[:\ell]$ iff f was contained in $\mathcal{Z}$'s input to some *intact* node when its chain length was at least $\ell - L_f$.

Although our definition of fruit quality is a slight variant in comparison with the original Fruitchains paper [40] — here we chose to define this variant as it is easier to work with — the Fruitchains work [40] essentially proved the following lemma:

Lemma 10 (Fruit quality [40])**.** *For any n, Δ, and any positive constant ρ, suppose that there exists a blockchain protocol Π such that for any positive constant ϵ, Π satisfies $\epsilon\kappa$-consistency, $(\epsilon\kappa, \epsilon\kappa, \mu)$-chain quality[3] for some positive μ, and $(\epsilon\kappa, g_0, g_1)$-chain growth for some positive g_0, g_1 in (n, ρ, Δ)-permissionless environments, then there exists another blockchain protocol $\widetilde{\Pi}$, an efficiently computable function* fruits*, and some constants C_0, C_1 such that for any positive constant ϵ, η, $(\widetilde{\Pi}, \mathsf{fruits})$ satisfies $\epsilon\kappa$-consistency, $(\epsilon\kappa, \epsilon\kappa, \mu)$-chain quality, $(\epsilon\kappa, g_0, g_1)$-chain growth, and $(C_0\kappa, C_1\kappa, \mu_f)$-fruit quality where $\mu_f := 1 - (1+\eta)\rho$ in (n, ρ, Δ)-permissionless environments.*

Henceforth we will use the terms "fruit quality" and "committee quality" interchangably. In particular, ideal fruit quality (or committee quality) would be exactly $1 - \rho$ where ρ denotes the overall fraction of corrupt nodes.

Overview of Fruitchains. A strawman idea for electing committees is simply to elect the miners of recent κ consecutive blocks as the committee, and to achieve consistency of the committee, a node should also remove $\Theta(\kappa)$ number of trailing blocks which might not have stablized. Indeed, some variant of this idea was adopted in some recent works such as Byzcoin [32] who incorrectly claimed that they could achieve almost optimal "committee quality".

An important observation is that such a strawman scheme does not result in almost ideal committee quality due to a well-known selfish mining attack [21, 23, 39]. We use a proof-of-work blockchain as an example below although selfish mining is also relevant to non-proof-of-work blockchains [15, 30, 43]. In such an attack, whenever a corrupt node mines a block, it does not release the block immediately but withholds it to itself. Whenever an honest miner mines a block at the same length, the adversary now performs a network rushing attack and makes sure that the adversary's private chain is received first by other honest nodes, such that honest nodes would now choose the adversary's private chain to extend from. Effectively, for every block mined by corrupt nodes, the adversary can erase one block worth of honest work, thus effectively "erasing" a fraction of the honest mining power. Suppose that the adversary controls ρ fraction of the total mining power and the honest nodes wield $1 - \rho$ fraction. As argued above, the adversary can effectively erase ρ fraction of the honest mining power, and thus from a back-of-the-envelop calculation (which can in fact be formalized [23, 39]), we can only attain roughly $\frac{1-2\rho}{1-2\rho+\rho} = 1 - \frac{\rho}{1-\rho}$ chain quality (*c.f.*, ideal chain quality would be $1 - \rho$). Thus the strawman scheme described above achieves only $1 - \frac{\rho}{1-\rho}$ committee quality. For example, when the adversary controls $\frac{1}{3}$ fraction of overall mining power, he controls roughly $\frac{1}{2}$ of the blocks. Similarly, if adversary controls close to $\frac{1}{2}$ fraction of overall mining power, he controls almost all blocks! Although the above selfish mining assumes that honest nodes pick the chain received earlier to break ties in length, a similar analysis would apply to other ways of tie breaking (but with possibly slightly less degradation in chain quality).

Fruitchain's idea defends against such selfish mining attacks by introducing *conceptually* two independent mining processes (in reality, the two processes are piggybacked on top of each other such that no extra mining is required on top of Nakamoto): one for mining *blocks* and one for mining *fruits*. Whenever an honest node mines a fruit, it will broadcast this fruit and honest nodes would then include the fruits in blockchains[4]. Since the work of mining the fruits cannot be erased

[3] The definition of chain quality is also adapted to count intact (rather than honest) blocks in the case of mildly adaptive corruptions.

[4] The Fruitchains paper [40] requires that blocks contain fruits and fruits contain transactions. In our paper, we

by an adversary, Fruitchain proves that in any sufficiently large window of consecutive blocks, the fraction of recently mined intact fruits approximates the overall fraction of intact nodes.

Henceforth in our paper will leverage Fruitchain's results as a blackbox in the manner stated by Lemma 10.

5.2.3 Detailed Protocol

We describe a concrete instantiation of our general committee election framework. This concrete protocol, henceforth denoted Π^*, is almost identical to $\widetilde{\Pi}_{\text{ella}}$ with the following modifications:

- The underlying $\Pi_{\text{blockchain}}$ is concretely instantiated with one that has almost ideal "fruit quality" where fruits denotes the fruit election function. Specifically, we assume that there are fixed constants C_0 and C_1 such that for any positive constants ϵ, η, $(\Pi_{\text{blockchain}}, \mathsf{fruits})$ satisfies $\epsilon\kappa$-consistency, $(\epsilon\kappa, \epsilon\kappa, \mu)$-chain quality for some positive μ, $(\epsilon\kappa, g_0, g_1)$-chain growth, and $(C_0\kappa, C_1\kappa, \mu_f)$-fruit quality where $\mu_f = 1 - (1+\eta)\rho$.
- Committee selection is now concretely instantiated with the following strategy: whenever the chain output from $\Pi_{\text{blockchain}}$ is of length $c - 0.5\kappa$,
  a) if $c - 0.5\kappa < C_0\kappa$, act as if $\texttt{elect}(c, \emptyset)$ is input from $\mathcal{Z}$ — note that by our definitions earlier, in this case, no transaction can be notarized w.r.t. the c-th committee;
  b) else act as if $\texttt{elect}(c, \mathsf{fruits}(\mathsf{chain}[: c - \kappa]))$ has been input from $\mathcal{Z}$.

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. Π^* iff

- $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. $\Pi_{\text{blockchain}}$; and
- $(\mathcal{A}, \mathcal{Z})$ respects $(n, \rho, \Delta, \tau, 0)$-permissionless execution w.r.t. Π^* for $\tau \geq \frac{(C_1+6)\kappa}{g_0}$ where g_0 is the underlying $\Pi_{\text{blockchain}}$'s growth lower bound parameter.

Theorem 20 (Concrete committee rotation strategy in a permissionless setting)**.** *For any p.p.t.$(\mathcal{A}, \mathcal{Z})$ compliant w.r.t. Π^*, the above protocol Π^* satisfies consistency, T_{confirm}-liveness, and $(T_{\text{warmup}}, T_{\text{opt}})$-optimistic responsiveness in lucky epochs w.r.t. $(\mathcal{A}, \mathcal{Z})$ for*

$$T_{\text{confirm}} := O(\frac{\kappa}{g_0}), \quad T_{\text{warmup}} := O(\frac{\kappa}{g_0}), \text{ and } \quad T_{\text{opt}} := 3\delta$$

where g_0 is the underlying $\Pi_{\text{blockchain}}$'s chain growth lower bound parameter.

Proof. By Theorems 17, 18, and 19, it suffices to prove that "consistency and timeliness of committee selection" and "resilience" conditions (see Section 5.1.1) are satisfied for all but a negligible fraction of the views — these follow in a straightforward fashion from the definition of τ-agility, chain growth lower bound, and fruit quality. □

Finally, it is not difficult to see that under mildly-adaptive sleepiness for a sufficiently large τ', and assuming that $\rho < \frac{1}{4} - \epsilon$, then except with negligible probability, it is guaranteed that more than $\frac{3}{4}$ of every $\mathsf{committee}_c$ are honest and online during $[\mathsf{len2time}(c - 0.5\kappa), \mathsf{len2time}(c + 0.5\kappa)]$. More specifically, if $(\mathcal{A}, \mathcal{Z})$ respects $(n, \frac{1}{4} - \epsilon, \Delta, \tau, \tau')$-permissionless execution where ϵ is an arbitrarily small positive constant, $\tau = \tau' > C\kappa\Delta$ for an appropriately large constant C, then, the condition for a lucky epoch boils down to the following: $[T_{\text{start}}, T_{\text{end}}]$ is a lucky epoch in view as long as the following hold:

assume a more practical variant where blocks contain both fruits and transactions, and fruits contain only a public key of the fruit's miner. Lemma 10 is also expressed for this variant.

- In any round $r \geq T_{\text{start}} + \Delta$, any honest and online node should have received the same instruction $\texttt{leader}(e, i)$. Further, prior to T_{start}, no honest node has received from $\mathcal{Z}$ any $\texttt{leader}(e', \_)$ where e' is the same or a larger epoch than e.

- Let c_0 denote the starting clock number for epoch e, then the leader i's chain must be no shorter than $c_0 - \kappa$ in round T_{start}.

- The leader (i.e., node i) is honest and online at in any round $t \in [T_{\text{start}}, T_{\text{end}} + 3\Delta]$;

5.2.4 Reward Distribution and Incentive Compatibility

In a decentralized cryptocurrency, it is important to incentivize participation, such that users will be financially driven to participate in the decentralized consensus — the broader the participation, the more decentralized (and thus more secure) the overall system is. Further, if the underlying blockchain is proof-of-work, users also need to be remunerated to offset the costs associated with participation.

Incentive compatible reward distribution can be achieved using the same ideas as Fruitchains [40]. For each block $\mathsf{chain}[\ell]$, the block reward and the transaction fees for all transactions contained in this block[5] can be distributed to the recent set of fruits, i.e., $\mathsf{fruits}(\mathsf{chain}[:\ell])$. Pass and Shi [40] show that such a reward distribution mechanism achieves ϵ-Nash equilibrium against any coalition that controls only minority of the total computation power (for an arbitrarily small positive constant ϵ). In other words, as long as the adversary controls less than minority of the computation power, it cannot earn more than ϵ fraction more than its fair share of rewards.

5.3 Recent Stake-Holders As Committee

No matter whether the underlying blockchain is proof-of-work or proof-of-stake [8, 15, 30, 36], it may make sense to select recent stake-holders to act as the committee. In Thunderella, we ask that the stake-based committee help accelerate transaction confirmation. Of course, if the underlying blockchain is also proof-of-stake (e.g., Snow White [15]), then the stake-based committee is in fact serving two purposes: 1) execute the underlying consensus protocol (e.g., Snow White [15]); and 2) help accelerate transaction confirmation in optimistic situations.

The selection of a stake-based committee in Thunderella is no different from the earlier works on proof-of-stake blockchains. Two classes of approaches have been proposed [7, 8, 15, 36]. In both approaches, committee re-election happens periodically per epoch, and the election process takes place on the blockchain (i.e., a committee is decided by looking back at a stabilized prefix of the blockchain). We describe the two committee election strategies below:

1. Have (possibly a random subset of) the recent stake-holders act as the committee. In this case, as Daian et al. [15] suggested, it would be a good idea to *limit the liquidity of the cryptocurrency*, such that the present committee cannot immediately sell all of its stake and be incentivized to attack the protocol — recall that since our committee rotation framework $\widetilde{\Pi}_{\text{ella}}$ defends against posterior corruption, i.e., a *past* committee who have entirely sold their stake well after their term of appointment cannot overwrite history and break consensus. Here the limited liquidity requirement prevents the *present* stake-holders from selling their stake and defecting too quickly. We defer more detailed discussions of the *limited liquidity* assumption to Daian et al. [15].

[5] Here a transaction tx is considered to be contained in the block iff this is the first block in which some notarized or unnotarized version of tx first appears.

2. Stake-holders can express interest in participation by putting down collateral [7, 8]. This collateral is frozen during the stake-holder's term of appointment. Should there be evidence that a stake-holder misbehaved during the protocol, its collateral may be taken away as penalty [7].

5.3.1 Fair Committee Down-Selection and Incentive Compatibility

In both cases, if too many stake-holders are eligible to vote, we may consider random down-selection of the committee to $\Theta(\kappa)$-size to reduce bandwidth consumption. Since committee members are rewarded for their participation, such down-selection should be *fair* [40], such that an adversary controlling ρ fraction of recent stake cannot control more than $(1 + \epsilon)\rho$ of the committee where ϵ is an arbitrarily small positive constant.

Strawman idea. A strawman idea is to rely on a random oracle H to perform the down-selection, i.e., if $H(\mathsf{pk}, (e, s, c)) < D_p$ where D_p is an appropriate difficulty parameter, then pk is considered a committee member for the tuple (e, s, c). Unfortunately, this idea is prone to an adaptive chosen-key attack, where an adversary, having seen the random oracle H, can choose keys that are guaranteed to be elected for a specific committee c.

Dealing with chosen-key attacks. To deal with such an adaptive chosen-key attack, we can adopt an idea described by Snow White [15], where the public keys must be registered on the blockchain before a random nonce is generated to seed the new random oracle. More formally, A public key pk is considered an eligible voter for the tuple (e, s, c) iff

1. pk holds sufficient stake in the prefix of the chain $\mathsf{chain}[: c - 2\kappa]$; (henceforth, we assume that some minimal stake S is required to become a voter, and if a public key pk has kS amount of stake, it is regared as k separate public keys);

2. We rely on $\Theta(\kappa)$ blocks $\mathsf{chain}[c - 2\kappa : c - \kappa]$ to generate a nonce r to seed the new random oracle. Specifically, every miner will embed a sufficiently high-entropy random string into every block it mines; and the nonce is obtained by concatenating the random strings contained in this $\Theta(\kappa)$-sized window. Although an adversary can arbitrarily manipulate the strings contained in corrupt blocks, by chain quality, there is at least one honest block among $\Theta(\kappa)$ consecutive blocks — and thus the resulting nonce must have sufficiently high-entropy;

3. $H(r, \mathsf{pk}, (e, s, c)) < D_p$ where D_p is an appropriate difficulty parameter such that each committee will be roughly $\Theta(\kappa)$ in size.

Defense against adaptive corruptions through secret committee election. One possible threat is that the adversary may adaptively corrupt committee members after having observed who are elected to a committee. To thwart such an attack, we can rely on verfiable random functions in a similar fashion as Section 4.2.1 — and this idea is inspired by Algorand [36]. Such a VRF allows us to elect a new committee for each (e, s, c) — but who is elected is hidden from the adversary in advance. In particular, the newly seeded random oracle generated in the aforementioned manner will instead be used to implement a VRF that determines if a certain public key is an eligible voter for the tuple (e, s, c). Further, the signature scheme for performing notarization is replaced with a forward-secure signature scheme as described in Section 4.2.1, such that the signing key evolves upon each usage, and the old state of the signing key is erased and purged from memory immediately after usage. Just as we argued in Section 4.2.1, such a scheme defends against adaptive corruptions in the random oracle model assuming erasure.

Penalty mechanisms. Another useful idea is to explicitly penalize committee members that have misbehaved (e.g., signed equivocating transactions) [7, 8]. If committee members put down collatoral to participate, penalty can be implemented by taking away their collateral and rewarding the party who submits cryptographic evidence of cheating (e.g., doubly signed signatures). Additionally, it would be useful to dis-incentivize free-riding, i.e., stake-holders who register to vote but do not end up participating in an attempt to get free rewards. This can be achieved by adjusting the reward mechanism to give more share to those who have participated (e.g., as cryptographically evident from the signatures they signed).

5.4 Leader As an Acceleration Service

Acceleration service. Another question is how to elect the leader in a decentralized deployment. An important observation here is that the leader is only an accelerator and need not be trusted. Even when the leader starts to behave arbitrarily, the worst thing it can do is that our protocol will fall back to the performance of today's blockchains (e.g., Bitcoin or Ethereum's blockchains). On the other hand, to get faster performance, it is desirable if the leader is well-provisioned, e.g., in terms of network resources. To obtain a really fast optimistic path, we can even have the leader open direct IP links to every committee member, for propagating notarization proposals and collecting votes. Once a proposal has been notarized, the notarized tuple (with all signatures) can then be propagated over a peer-to-peer overlay network (e.g., Bitcoin or Ethereum's peer-to-peer network) to everyone else.

One desirable approach is for nodes to bid to become the leader and the leader can potentially charge a percentage of the transaction fees to offset its costs of running an acceleration service. The network should prefer to choose a leader that is well-provisioned and has a fast link to everyone.

Anyone interested in providing such an acceleration service should be aware of the possibility of a Distributed Denial-of-Service (DDoS) attack on the leader. Such a DDoS attack can transiently slow down transaction confirmation until a new leader takes over. There are numerous approaches that a leader can employ to protect itself against such DDoS attacks. For example, it can rely on commercially available DDoS protection services (the associated costs can be offset by collecting fees from running the acceleration service). The leader can also be placed behind geographically distributed proxies and not present a public IP address. The leader can also agilely move from one IP address to another to reduce the risk of being DDoS'ed. Even in the presence of an actual DDoS attack, we can always rely on any form of *heuristic* offchain mechanism to hand-off to a backup leader — this hand-off mechanism is not on any security critical path, and any suitable heursitics can be adopted, since our protocol is secure even when the logical leader is corrupt.

Monitoring the leader. The network can together monitor the leader and vote the leader out whenever suspicious behavior is detected. For example, one concern might be that the leader in Thunderella can possibly reorder transactions (just like miners in Bitcoin or Etheurem). Fundamentally, since the network has delay up to Δ, for transactions that are issued Δ rounds apart from each other, it may not be possible to determine their absolute order. However, if the leader attempts to reorder transactions more drastically, e.g., by more than Δ rounds, it is possible for other peers to detect it. For example, a peer can monitor the leader by comparing the leader's proposed order against the order and times at which the peer observes transactions himself. If sufficiently many stake-holders do not like the leader, then they can jointly vote the current leader out and reelect a new one.

Bibliography

[1] Personal communication with Vitalik Buterin.

[2] Hagit Attiya, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, 1994.

[3] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. In *Financial Cryptography Bitcoin Workshop*, 2016.

[4] Iddo Bentov, Charles Lee, Alex Mizrahi, and Meni Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake. In *Proceedings of the ACM SIGMETRICS 2014 Workshop on Economics of Networked Systems, NetEcon*, 2014.

[5] Kenneth P. Birman and Thomas A. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of the Eleventh ACM Symposium on Operating System Principles, SOSP 1987, Stouffer Austin Hotel, Austin, Texas, USA, November 8-11, 1987*, pages 123–138, 1987.

[6] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, 2006.

[7] Vitalik Buterin. `https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c`, 2017.

[8] Vitalik Buterin and Vlad Zamfir. Casper. `https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/`, 2015.

[9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[10] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *Theory of Cryptography*, pages 61–85. Springer, 2007.

[11] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, 2003.

[12] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. Unbeatable consensus. In *DISC*, 2014.

[13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[14] User "cunicula" and Meni Rosenfeld. Proof of stake brainstorming. `https://bitcointalk.org/index.php?topic=37194.0`, August 2011.

[15] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016.

[16] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *ICDCN*, 2016.

[17] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, October 1990.

[18] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *Siam Journal on Computing - SIAMCOMP*, 12(4):656–666, 1983.

[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

[20] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment i: Crash failures. In *TARK*, pages 149–169, 1986.

[21] Ittay Eyal and Emin Gun Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, 2014.

[22] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. Cryptology ePrint Archive, 2016/1048.

[23] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[24] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *Proceedings of the 5th European Conference on Computer Systems*, EuroSys '10, pages 363–376, New York, NY, USA, 2010. ACM.

[25] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual byzantine agreement. *SIAM J. Comput.*, 31(3):838–865, 2001.

[26] Maurice Herlihy, Yoram Moses, and Mark R. Tuttle. Transforming worst-case optimal solutions for simultaneous tasks into all-case optimal solutions. In *PODC*, 2011.

[27] Amir Herzberg and Shay Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–1196, 2000.

[28] Flavio P. Junqueira, Benjamin C. Reed, and Marco Serafini. Zab: High-performance broadcast for primary-backup systems. In *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, DSN '11, pages 245–256, 2011.

[29] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, February 2009.

[30] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. `http://eprint.iacr.org/2016/889`.

[31] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. `https://peercoin.net/assets/paper/peercoin-paper.pdf`, 2012.

[32] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. *CoRR*, abs/1602.06997, 2016.

[33] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 45–58, 2007.

[34] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

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

[36] Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

[37] Yoram Moses and Michel Raynal. No double discount: Condition-based simultaneity yields limited gain. *Inf. Comput.*, 214:47–58, May 2012.

[38] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[39] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[40] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.

[41] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, 2017.

[42] Rafael Pass and Elaine Shi. Rethinking large-scale consensus (invited paper). In *CSF*, 2017.

[43] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Asiacrypt*, 2017.

[44] User "QuantumMechanic". Proof of stake instead of proof of work. `https://bitcointalk.org/index.php?topic=27787.0`, July 2011.

[45] Yee Jiun Song and Robbert van Renesse. Bosco: One-step byzantine asynchronous consensus. In *DISC*, pages 438–450, 2008.

[46] Dominic Williams. How to achieve near-instant finality in public blockchains using a VRF. `http://string.technology/2017/02/03/stanford-conference.en/`, 2017.

[47] Gavin Wood. Ethereum: A secure decentralized transaction ledger. `http://gavwood.com/paper.pdf`, 2014.

Global Signing Functionality

We define the following global signing functionality $\mathcal{G}_{\text{sign}}^{\Sigma}$ that is parametrized by a signature scheme Σ. This functionality can easily be realized assuming a (global) bare public key infrastructure.

$\mathcal{G}_{\text{sign}}^{\Sigma}$

- On receiving `initialize` from some honest node i: call $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$, notify $\mathcal{A}$ of (i, pk_i), and return pk_i.
- On receiving $\texttt{setkey}(\mathsf{pk}_i, \mathsf{sk}_i)$ from some corrupt node i: record $(\mathsf{pk}_i, \mathsf{sk}_i)$ if a key pair has not been recorded for i.
- On honest node i becoming corrupt: disclose sk_i to $\mathcal{A}$.
- On receiving $\texttt{sign}(\mathsf{msg})$ from party i in protocol instance sid: if a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ has been recorded for i, then return $\sigma := \Sigma.\mathsf{Sign}(\mathsf{sk}_i, (sid, \mathsf{msg}))$; else return $\perp$.
- On receiving $\texttt{query}(i)$: return pk_i.

Interactions between $\mathcal{Z}$ and $\mathcal{G}_{\text{sign}}$. We assume that the following types of interactions can happen between $\mathcal{Z}$ and $\mathcal{G}_{\text{sign}}$:

- $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$ acting as the adversary $\mathcal{A}$ or any corrupt node;
- $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$ acting as an honest party, but only for non-challenge protocol sessions (i.e., where the session identifier must be different from the challenge protocol session that we are concerned about). This models the fact that there may be other rogue protocols running possibly concurrently with the challenge protocol session.