The Sleepy Model of Consensus

Iddo Bentov Rafael Pass Elaine Shi

Cornell, CornellTech, Initiative for Crypto-Currency and Contracts (IC3)*

Abstract

The distributed systems literature adopts two primary network models, the synchronous model where honest messages are delivered in the next round, and the partially synchronous (or asynchronous) model where honest messages are subject to unpredictable adversarial delays.

In this paper, we show that more nuanced formal models exist beyond the traditional synchrony and asynchrony stratification — and interestingly, such new models allow us to articulate new robustness properties that traditional models would have failed to capture.

More specifically, we articulate a new formal model called "the sleepy model of consensus", where we classify honest nodes as being either alert or sleepy. Alertness implies that the node is online and has good network connections; whereas sleepiness captures any type of failure or network jitter. We then describe the Sleepy consensus protocol that achieves security as long as at any time, *the number of alert nodes outnumber corrupt ones*. No classical synchronous or asynchronous protocols attain such robustness guarantees, and yet we show how to leverage Nakamoto's blockchain protocol, but without proofs-of-work, to achieve these properties, assuming collision resistant hash functions, the existence of a public-key infrastructure and a common reference string.

1 Introduction

Consensus protocols are at the core of distributed systems — an important and exciting area that has thrived for the past 30 years. In this paper, we consider consensus protocols which roughly speaking, realize a "linearly ordered log" abstraction — often referred to as state machine replication or linearizability in the distributed systems literature. Such protocols must respect two important properties, *consistency* and *liveness*. Consistency ensures that all (correct) nodes have the same view of the log, whereas liveness requires that transactions will be incorporated into the log quickly.

Traditionally, the deployment of consensus protocols has been largely restricted to relatively controlled environments, e.g., hypothetically, a deployment within a company such as Google to support mission-critical applications such as Google Wallet. The scale of deployment has been relatively small, typically involving no more dozens of nodes. Nodes are often owned by the same organization and inter-connected with high-speed networks.

The rapid rise to fame of decentralized cryptocurrencies such as Bitcoin and Ethereum have undoubtedly pushed consensus protocols to newer heights, and have demonstrated to us, that amazingly, it is possible to achieve robust consensus in a decentralized environment that is much

*http://www.initc3.org/

more "hostile" than the traditional comfort zones for consensus deployment. Partly for this reason, Bitcoin is often referred to as the "honeybadger of money".

Consensus in the decentralized setting faces numerous new challenges. Nodes do not have pre-established or long-term trust relations; they can come and go, causing frequent churn; and further, network conditions can vary rapidly over time. The community seem to agree that consensus protocols "more robust" than traditional ones are desired. However, the precise understanding of what "more robust" means seems to be somewhat lacking. All in all, the amazing success of cryptocurrencies motivates us to rethink consensus at a fundamental level: not only in terms of new protocols, but also whether well-accepted formal models are expressive enough to capture the new requirements raised by new application environments.

Although most existing cryptocurrencies adopt *permissionless* consensus protocols where any node can join and leave at any time; in this paper, we instead focus on the classical *permissioned* setting. In permissioned consensus, there is a fixed set of consensus nodes known to everyone in advance. Recently, there is a growing interest in applying permissioned consensus to decentralized environments, partly because permissionless protocols prevalently adopt proofs-of-work which is enormously wasteful — without it (and absent other trust assumptions) permissionless consensus would have been impossible [1]. It is also well-understood by now that one can employ permissionless consensus to bootstrap common knowledge on a committee of nodes, at which point one can switch to permissioned consensus. For example, proofs-of-stake protocols [4, 5, 8, 14, 24, 26, 32, 33, 39, 41] advocate using an existing cryptocurrency such as Bitcoin to distribute an initial set of coins, and then agree on a set of stakeholders such that each node is given voting power that is proportional to their amount of stake. Recent works have also formalized how to securely compose permissionless and permissioned consensus to achieve certain desirable properties [38]. Obviously, since the decentralized setting is conceivably adversarial, in this paper, we are interested in protocols that tolerate Byzantine faults where corrupt nodes can behave arbitrarily.

1.1 The "Sleepy" Model of Consensus

Despite the existence of an a-priori common committee, not all nodes are necessarily online at all times. Nodes can be offline transiently or permanently for various reasons ranging from power outages to bad network connections — such failures seem inevitable in a wide-area deployment. Obviously one cannot hope for offline nodes to participate in the consensus protocol, and we therefore ask the following question — henceforth we refer to this goal as the "sleepy" model of consensus:

> Can we design a consensus protocol that achieves consistency and liveness when the *majority* of *online* nodes are behaving honestly?

It seems completely natural that we may desire such guarantees. However, surprisingly, it turns out that the classical body of literature on consensus protocols fail to provide a satisfactory answer this question! To the best of our knowledge, no known permissioned consensus protocol can realize the stated goal of "sleepy consensus", i.e., achieving security when the majority of online nodes are honest. Roughly speaking, traditional permissioned consensus protocols are secure either in the *synchronous* model, where messages delivered by honest nodes are received by all other honest nodes in the next round; or the *partially synchronous* or *asynchronous* model, where messages can take unknown time to deliver, and possibly subject an adversarial network schedule. Unfortunately

both flavors of protocols would fail to address our needs (at least when applied in a blackbox manner):

- First, the ability for nodes to go to sleep and later wake up breaks synchrony assumptions outright. Traditional synchronous protocols [15, 18, 22] crucially relies on messages being delivered in the next round (or within a known, bounded delay) to reach common knowledge. By contrast, a sleepy node can wake up at an indefinite time in the future. When it wakes, imagine that all pending messages destined for that node including adversarially inserted ones are now delivered. Such a model clearly permits messages to be delivered out of sync, and this immediately violates the synchrony expected by traditional synchronous protocols.

- Second, imagine that no node is sleeping — in this case, the stated goals of sleepy consensus would imply that the protocol must be secure in the presence of honest majority (among all nodes). Unfortunately, this quickly rules out the class of partially synchronous or asynchronous protocols [9, 13, 16, 31, 34, 40] as well. Due to a lower bound by Dwork et al. [16], it is well-understood that consensus is impossible against a one-third coalition in partially synchronous or asynchronous networks — even when allowing additional assumptions such as globally synchronized clocks or the presence of a public-key infrastructure. Indeed, all known partially synchronous or asynchronous protocols [9, 13, 16, 31, 34, 40] crucially rely on more than $\frac{2}{3}$ (or more) fraction of the nodes being honest to attain security.

  We remark, however, that partially synchronous and asynchronous models allow messages to be delayed indefinitely, and therefore these models were actually designed exactly to capture such hostile environments — only that this setting is subject to the $\frac{1}{3}$-lower bound by Dwork et al. [16], and in many applications, a stronger degree of resilience is desired (e.g., requiring security in the presence of honest majority rather than $\frac{2}{3}$ honest).

1.2 Our Contributions

It may now seem hopeless to achieve what we had asked, that is, to simultaneously tolerate

1. up to 50% corruption (among online nodes); and
2. potentially hostile environments where node outages and network jitters are unavoidable.

Indeed, by conventional wisdom, these above two goals would seem inherently incompatible. To model hostile networks, we would need partial synchrony or asynchrony, but then we would be constrained to a well-known $\frac{1}{3}$-lower bound [16], and it would have been impossible to resist a near 50% attack.

In this paper, we show that perhaps surprisingly, achieving "sleepy consensus" is possible. More concretely, we make following contributions.

Defining sleepy consensus. In our sleepy model of consensus, we classify honest nodes into two types, those that are *alert*, and those that are *asleep* (also referred to as sleepy). Alert nodes are assumed to have good network connections and any message delivered by an alert node will arrive at all other alert nodes within Δ delay. A network adversary can arbitrarily reorder or delay messages subject to the Δ constraint. To circumvent the partial synchrony lower bound, we provide an upper bound on Δ as an input parameter to our protocol. Therefore, on the surface, we seem

to take after the synchronous model — particularly the traditional distributed systems literature generally classifies the case of known-Δ as synchronous, since nodes can simply wait for each Δ as a synchronous round.

We stress, however, that the sleepy model can be fundamentally separated from the traditional synchronous model in that we allow nodes to temporarily go to sleep and later wake up again, thus allowing us to broadly capture the unpredictability of wide-area networks. In particular, sleeping not only captures short-term node outages, but also admits situations in which a node temporarily experiences longer than Δ network delay! In this case, the node can be treated as asleep for a short window; when the node wakes up again, all pending network messages are delivered (along with possibly adversarially inserted ones) — in this sense, the sleepy model bears resemblance to the traditional partially synchronous (or asynchronous) models. For this reason, to the best of our knowledge no known synchronous consensus protocols can retain security in our sleepy model — typically these protocols rely on strong synchrony to ensure that all nodes reach common knowledge during a certain time step.

The Sleepy consensus protocol. We present a new protocol called Sleepy, that achieves the above-stated goals in the sleepy model of consensus. Intriguingly, Sleepy is heavily inspired by Nakamoto's celebrated blockchain protocol [35], although we avoid using computationally expensive puzzles.

Theorem 1 (Informal)**.** Assuming the existence of a collision-resistant hash family, a public-key infrastructure, and a common reference string, and assuming that nodes have weakly synchronized clocks, there is a consensus protocol Π_{sleepy} that achieves consistency and liveness in the sleepy model under static corruption, as long as at any moment of time, there are more alert nodes than corrupt ones.

As far as we know, this is the first use of blockchain-style protocols to improve classical consensus protocols. Our proof leverages the formal analysis of the Nakamoto blockchain by Pass et al. [36], but since we no longer rely on proofs-of-work, we need to propose a significant extension to existing proofs to reason about consistency.

The consistency and liveness guarantees we achieve cover all nodes that are currently alert. Obviously, it is not possible to achieve liveness for sleepy nodes since they may simply be disconnected from the rest of the network. However, we guarantee that consistency and liveness will immediately ensue as soon as a sleepy node wakes up again.

Main insight and techniques. Our main technical insight is how to apply core ideas behind blockchain protocols in a non-proof-of-work setting, to solve classical problems in distributed systems. In a proof-of-work blockchain, the adversary's limit in computation power limits his ability to mine many blocks in any window of time. When removing the proof-of-work, we still need to rate limit the adversary in a similar manner. Our idea is to use a random oracle (which we can be removed later) to elect leaders at random for each time step. If the adversary controls ρ fraction of the nodes, he is locked to roughly ρ fraction of the time slots, and he can only extend the blockchain with blocks that correspond to the time slots during which a corrupt node is elected leader.

Although at first sight, the idea seems rather intuitive and straightforward, it turns out that to obtain a provable secure version involves non-trivial challenges both in terms of construction and proofs. In a proof-of-work blockchain, whenever the adversary mines a block, he cannot transfer this

block and concatenate it with any other chain. By contrast, in our setting, if the adversary is elected leader in a certain time step t, he can sign many chains using this time slot! Therefore, absent any further constraints on the adversary's behavior, the protocol cannot be secure. We propose new techniques that leverage blockchain timestamps to constrain the capabilities of an adversary. We show that designing such timestamping mechanisms is tricky: not only must the timestamps sufficiently constrain the adversary, care must also be taken to ensure that the adversary cannot manipulate timestamps to cause alert nodes to get stuck.

On the proof front, too, we need non-trivial new techniques in comparison with previous analysis of proof-of-work blockchains [21, 36]. At a very high level, our adversary can launch more possible attacks that were not possible in proofs-of-work blockchains. Therefore, to prove the consistency property of the Sleepy protocol, we must introduce novel techniques to reason about properties of the induced stochastic process. We defer a more detailed technical roadmap and discussions to Section 1.3.

We stress that while some protocols proposed by the community may bear superficial resemblance to ours [2, 4, 25], we cannot prove any of the existing variants secure. As mentioned, while the high-level idea of the protocol is intuitive, it turns out that numerous subtle details matter crucially to our proof, including how the timestamps are used and what difficulty parameter to set for the leader election — and getting these details correct and being able to leverage them in the security proof is the key technical challenge. Unfortunately, existing proposals [2, 4, 25] lack one or more important ingredients that our proofs crucially rely on (or possible proofs that we could conceive of).

Lower bounds. Given that we show how to achieve security for the honest majority setting (in terms of awake nodes) in the sleepy model, one natural question arises: can we have a consensus protocol with better resilience in the sleepy model, e.g., one that tolerates possibly a corrupt majority? We present a lower bound proving that it is impossible tolerate a 50% (or higher) attack. Informally speaking, if the adversary wields corrupt majority, he can simulate a fake execution trace that is identically distributed as the real one. In this way, if a sleepy node sleeps from the very beginning and wakes up much later, the adversary present protocol messages from both executions to the waking node — and the waking node will have no means of discerning which is real and which is simulated. We formalize this lower bound in Section 5.

In light of this lower bound, our Sleepy consensus protocol achieves optimal resilience.

1.3 Technical Roadmap

The design of our consensus protocols draws inspiration from Bitcoin's proof-of-work blockchain [35]. However, we adapt the protocols to the permissioned setting allowing us to eliminate the need to perform proofs-of-work.

Remark. For simplicity, we describe our scheme assuming there exists a random oracle H. Later we will argue how to replace the random oracle with a common reference string and a pseudorandom function.

Nakamoto in a nutshell. We first review Nakamoto's blockchain protocol [35]. Roughly speaking, players "confirm" transactions by "mining blocks" through solving some computational puzzle

that is a function of the transactions and the history so far. More precisely, each participant maintains its own local "chain" of "blocks" of transactions — called the *blockchain*. Each block consists of a triple $(h_{-1}, \eta, \mathsf{txs})$ where h_{-1} is a pointer to the previous block in chain, txs denotes the transactions confirmed, and η is a "proof-of-work"— a solution to a computational puzzle that is derived from the pair (h_{-1}, txs). The proof of work can be thought of as a "key-less digital signature" on the whole blockchain up until this point. At any point of time, nodes pick the *longest* valid chain they have seen so far and try to extend this longest chain.

Strawman attempt. To remove the proof-of-work from Nakamoto, the most straightforward is the following: instead of rate limiting through computational power, we rate limit by tying each node to randomly selected time steps. A node can only extend the blockchain with a block timestamped t, if he is elected leader in t. More specifically, we redefine the puzzle solution to be of the form $(\mathcal{P}, t)$ where $\mathcal{P}$ is the party's identifier and t is a timestamp. The pair $(\mathcal{P}, t)$ is a "valid puzzle solution" if $\mathsf{H}(\mathcal{P}, t) < D_p$ where H denotes a random oracle (which can be removed later), and D_p is a parameter such that the hash outcome is only smaller than D_p with probability p. If $\mathsf{H}(\mathcal{P}, t) < D_p$, we say that $\mathcal{P}$ is leader in time t.

Now, a node $\mathcal{P}$ that is elected leader in time t can extend a chain with a block that includes the "solution" $(\mathcal{P}, t)$, as well as the previous block's hash h_{-1} and the transactions txs to be confirmed. To verify that the block indeed came from $\mathcal{P}$, we require that the entire contents of the block, i.e., $(h_{-1}, \mathsf{txs}, t, \mathcal{P})$, are signed under $\mathcal{P}$'s public key. In a similar manner as Nakamoto, nodes always choose the longest valid chain they have seen and extend this longest chain.

Of course, just doing this does not work, since nothing prevents the adversary from reusing a time slot for which he is elected leader, and signing multiple blocks, even in the same chain. Naturally, we have to impose constraints on a valid blockchain's timestamps.

Constraints on blocks' timestamps. One natural idea is to require that each valid blockchain must have all distinct timestamps. However, this is not sufficient for constraining the adversary. While the alert nodes will only "mine" in the present time, the adversary can "mine" into the future, or reuse past time slots. Instead, we impose the following stronger constraints:

1. A valid chain must have strictly increasing timestamps; and
2. A valid chain with future timestamps (where "future" is adjusted to account for nodes' clock offsets) are considered invalid and rejected by honest nodes.

There are two important technical issues to resolve. First, it is important to ensure that the timestamp rules do not hamper liveness. In other words, there should not be any way for an adversary to leverage the timestamping mechanism to cause alert nodes to get stuck (e.g., by injecting false timestamps). We prove that with the aforementioned timestamp rules, it is indeed the case that the adversary cannot hamper liveness no matter how it deviates from the protocol.

Second, although our timestamp rules severely constrain the adversary, the adversary is still left with some wiggle room, and gets more advantage than alert nodes. Specifically, as mentioned earlier, the alert nodes only "mine" in the present, and moreover they never try to extend different chains of the same length. By contrast, the adversary can try to reuse past timestamps in multiple chains. We prove that such wiggle room is in some sense insignificant, and the adversary cannot leverage the wiggle room to break the protocol's consistency guarantees. As we explain next, it

turns out that dealing with this wiggle room is also technically challenging, and none of the existing analysis for proof-of-work blockchains [21, 36] would directly apply.

Proof challenges and techniques. Since we are a blockchain-style protocol, a natural idea is to see whether we can borrow proof ideas from existing analysis of the Nakamoto blockchains [36].

The good news is that chain growth and chain quality can be proven in a similar manner as the earlier work by Pass et al [36]. The bad news is that the consistency proof turns out to be much more difficult in our case. Specifically, Pass et al.'s consistency proof relies on showing that the adversary cannot have mined too many blocks in a reasonably long time window. Specifically, the adversary cannot mine more blocks than there are convergence opportunities (to be formally defined in Section 6). It turns out that in the case of the Nakamoto blockchain, if the adversary mines fewer blocks than convergence opportunities in a certain window, then the adversary cannot have caused divergence before that window.

Unfortunately in our case this type of reasoning breaks. In particular, when the adversary mines a block in the Nakamoto blockchain, he cannot reuse this block by concatenating it with any other chain. In our protocol, however, if the adversary gets elected for a single time step, this "earned time slot" can potentially be reused many times in an attack to break consistency — specifically, the adversary can potentially extend many chains with a single earned time slot.

To account for such adversarial reuse of earned time slots, we devise a new proof strategy different from Pass et al. [36] for proving consistency. We consider the occurrence of a certain good event called a pivot point. Informally speaking, a pivot point is a point of time t such that for any window containing t, the adversary is elected less often than the convergence opportunities contained in t. We show that whenever a pivot point t happens, the adversary cannot have caused divergence before t, and further pivot points happen sufficiently often such that only trailing κ blocks in any alert node's chain may be inconsistent.

1.4 Related Work

We briefly review the rich body of literature on consensus, particularly focusing on protocols that achieve security against Byzantine faults where corrupt nodes can deviate arbitrarily from the prescribed behavior.

Models for permissioned consensus. Consensus in the permissioned setting [3, 6, 7, 9, 13, 15, 16, 18–20, 22, 27–31, 40] has been actively studied for the past three decades; and we can roughly classify these protocols based on their network synchrony, their cryptographic assumptions, and various other dimensions.

Roughly speaking, two types of network models are typically considered, the *synchronous* model, where messages sent by honest nodes are guaranteed to be delivered to all other honest nodes in the next round; and *partially synchronous* or *asynchronous* protocols where message delays may be unbounded, and the protocol must nonetheless achieve consistency and liveness despite not knowing any a-priori upper bound on the networks' delay. In terms of cryptographic assumptions, two main models have been of interest, the "*unauthenticated Byzantine*" model [30] where nodes are interconnected with authenticated channels[1]; and the "*authenticated Byzantine*" model [15], where

[1]This terminology clash stems from different terminology adopted by the distributed systems and cryptography communities.

a public-key infrastructure exists, such that nodes can sign messages and such digital signatures can then be transferred.

Permissioned, synchronous protocols. Many feasibility and infeasibility results have been shown. Notably, Lamport et al. [30] show that it is impossible to achieve secure consensus in the presence of a $\frac{1}{3}$ coalition in the "unauthenticated Byzantine" model (even when assuming synchrony). However, as Dolev and Strong show [15], in a synchronous, authenticated Byzantine model, it is possible to design protocols that tolerate an arbitrary number of corruptions. It is also understood that no deterministic protocol fewer than f rounds can tolerate f faulty nodes [15] — however, if randomness is allowed, existing works have demonstrated expected constant round protocols that can tolerate up to a half corruptions [18, 22].

Permissioned, asynchronous protocols. A well-known lower bound by Fischer, Lynch, and Paterson [19] shows if we restrict ourselves to protocols that are deterministic and where nodes do not read clocks, then consensus would be impossible even when only a single node may be corrupt. Known feasibility results typically circumvent this well-known lower bound by making two types of assumptions: 1) randomness assumptions, where randomness may come from various sources, e.g., a common coin in the sky [9, 20, 34], nodes' local randomness [3, 40], or randomness in network delivery [7]; and 2) clocks and timeouts, where nodes are allowed to read a clock and make actions based on the clock's value. This approach has been taken by well-known protocols such as PBFT [13] and FaB [31] that use timeouts to re-elect leaders and thus ensure liveness even when the previous leader may be corrupt.

Another well-known lower bound in the partially synchronous or asynchronous setting is due to Dwork et al. [16], who showed that no protocol (even when allowing randomness or clocks) can achieve security in the presence of a $\frac{1}{3}$ corrupt coalition.

Permissionless consensus. The permissionless model did not receive sufficient academic attention, perhaps partly due to the existence of strong lower bounds such as what Canetti et al. showed [1]. Roughly speaking, we understand that without making additional trust assumptions, not many interesting tasks can be achieved in the permissionless model where authenticated channels do not exist between nodes.

Amazingly, cryptocurrencies such as Bitcoin and Ethereum have popularized the permissionless setting, and have demonstrated to us, that perhaps contrary to the common belief, highly interesting and non-trivial tasks can be attained in the permissionless setting. Underlying these cryptocurrency systems is a fundamentally new type of consensus protocols commonly referred to as proof-of-work blockchains [35]. Upon closer examination, these protocols circumvent known lower bounds such as those by Canetti et al. [1] and Lamport et al. [30] since they rely on a new trust assumption, namely, proofs-of-work, that was not considered in traditional models.

Formal understanding of the permissionless model has just begun [21, 36–38]. Notably, Garay et al. [21] formally analyze the Nakamoto blockchain protocol in synchronous networks. Pass et al. [36] extend their analysis to asynchronous networks. More recently, Pass and Shi [38] show how to perform committee election using permissionless consensus and then bootstrap instances of permissioned consensus — in this way, they show how to asymptotically improve the response time for permissionless consensus.

Finally, existing blockchains are known to suffer from a selfish mining attack [17], where a coalition wielding $\frac{1}{3}$ of the computation power can reap up to a half of the rewards. Pass and Shi [37] recently show how to design a fair blockchain (called Fruitchains) from any blockchain protocol with positive chain quality. Since our Sleepy consensus protocol is a blockchain-style protocol, we also inherit the same selfish mining attack. However, we can leverage the same techniques as Pass and Shi [37] to build a fair blockchain from Sleepy.

2 Definitions

2.1 Protocol Execution Model

We assume a standard Interactive Turing Machine (ITM) model [10–12] often adopted in the cryptography literature.

(Weakly) synchronized clocks. We assume that all nodes can access a clock that ticks over time. In the more general form, we allow nodes clocks to be offset by a bounded amount — commonly referred to as weakly synchronized clocks. We point out, that it is possible to apply a general transformation such that we can translate the clock offset into the network delay, and consequently in the formal model we may simply assume that nodes have synchronized clocks without loss of generality.

Specifically, without loss of generality, assume nodes' clocks are offset by at most Δ, where Δ is also the maximum network delay — if the two parameters are different, we can always take the maximum of the two incurring only constant loss. Below we show a transformation such that we can treat weakly synchronized clocks with maximum offset Δ as setting with synchronized clocks but with network delay 3Δ. Imagine the following transformation: honest nodes always queue every message they receive for exactly Δ time before "locally delivering" them. In other words, suppose a node i receives a message from the network at local time t, it will ignore this message for Δ time, and only act upon the received message at local time $t + \Delta$. Now, if the sender of the message (say, node j) is honest, then j must have sent this message during its own local time $[t - 2\Delta, t + \Delta]$. This suggests that if an honest node j sends a message at its local time t, then any honest node i must locally deliver the message during its local time frame $[t, t + 3\Delta]$.

Therefore henceforth in this paper we consider a model with a globally synchronized clocks (without losing the ability to express weak synchrony). Each clock tick is referred to as an atomic *time step*. Nodes can perform unbounded polynomial amount of computation in each atomic time step, as well as send and receive polynomially many messages.

Network delivery. The adversary is responsible for delivering messages between nodes. We assume that the adversary $\mathcal{A}$ can *delay* or *reorder* messages arbitrarily, as long as it respects the constraint that all messages sent from honest nodes must be received by all honest nodes in at most Δ time steps.

Corruption model. We consider a static model of corruption. Prior to protocol start, the environment $\mathcal{Z}$ must declare which nodes are corrupt. Further, prior to protocol start, $\mathcal{Z}$ can issues instructions of the form

$$(\texttt{sleep}, i, t_0, t_1)$$

This will cause node i to sleep during $[t_0, t_1]$.

When a sleepy node wakes up, $(\mathcal{A}, \mathcal{Z})$ is required to deliver an unordered set of messages containing

- all the pending messages that node i would have received (but did not receive) had it not slept; and
- any polynomial number of adversarially inserted messages of $(\mathcal{A}, \mathcal{Z})$'s choice.

To summarize, a node can be in one of the following states:

1. *Honest.* An honest node can either be *awake* or *asleep* (or sleeping/sleepy). Henceforth we say that a node is *alert* if it is honest and awake. When we say that a node is *asleep* (or sleeping/sleepy), it means that the node is honest and asleep.
2. *Corrupt.* Without loss of generality, we assume that all corrupt nodes are *awake*.

Public-key infrastructure. We assume the existence of a public-key infrastructure (PKI). Specifically, we adopt the same technical definition of a PKI as in the Universal Composition framework [10]. Specifically, we shall assume that the PKI is an ideal functionality $\mathcal{F}_{\text{CA}}$ that does the following:

- On initialize: $\mathsf{cmt} = \emptyset$
- On first receive $\texttt{register}(\mathsf{pk})$ from $\mathcal{P}$: add $(\mathsf{pk}, \mathcal{P})$ to cmt
- On receive $\texttt{lookup}(\mathcal{P})$: return the stored pk corresponding to $\mathcal{P}$ or $\perp$ if none is found.

2.2 Notational Conventions

Negligible functions. A function $\mathsf{negl}(\cdot)$ is said to be *negligible* if for every polynomial $p(\cdot)$, there exists some κ_0 such that $\mathsf{negl}(\kappa) \leq \frac{1}{p(\kappa)}$ for all $\kappa \geq \kappa_0$.

Variable conventions. In this paper, unless otherwise noted, all variables are by default (polynomially bounded) functions of the security parameter κ. Whenever we say $\mathsf{var_0} > \mathsf{var_1}$, this means that $\mathsf{var_0}(\kappa) > \mathsf{var_1}(\kappa)$ for every $\kappa \in \mathbb{N}$. Variables may also be functions of each other. How various variables are related will become obvious when we define derived variables and when we state parameters' admissible rules for each protocol.

Importantly, *whenever a parameter does not depend on κ, we shall explicitly state it by calling it a constant.*

Protocol conventions. We adopt the universal composition framework [10–12] for formal modeling. Each protocol instance and functionality is associated with a session identifier *sid*. We omit writing this session identifier explicitly without risk of ambiguity. We assume that ideal functionalities simply ignore all messages from parties not pertaining to the protocol instance of interest.

3 Preliminaries: Blockchain Formal Abstraction

In this section, we define the formal abstraction and security properties of a blockchain. As Pass and Shi [38] recently show, a blockchain abstraction implies a classical state machine replication abstraction. Our definitions follow the approach of Pass et al. [36], which in turn are based on earlier definitions from Garay et al. [21], and Kiayias and Panagiotakos [23].

Since our model distinguishes between two types of honest nodes, alert and sleepy ones, we define chain growth, chain quality, and consistency for alert nodes. However, we point out the following: 1) if chain quality holds for alert nodes, it would also hold for sleepy nodes; 2) if consistency holds for alert nodes, then sleepy nodes' chains should also satisfy common prefix and future self-consistency, although obviously sleepy nodes' chains can be much shorter than alert ones.

Notations. For some $\mathcal{A}, \mathcal{Z}$, consider some view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$; we use the notation $|\mathsf{view}|$ to denote the number of time steps in the execution, view^t to denote the prefix of view up until time step t.

We assume that in every time step, the environment $\mathcal{Z}$ provides a possibly empty input to every honest node. Further, in every time step, an alert node sends an output to the environment $\mathcal{Z}$. Given a specific execution trace view with non-zero support where $|\mathsf{view}| \geq t$, let i denote a node that is alert at time t in view, we use the following notation to denote the output of node i to the environment $\mathcal{Z}$ at time step t,

$$\text{output to } \mathcal{Z} \text{ by node } i \text{ at time } t \text{ in } \mathsf{view}: \quad \mathsf{chain}_i^t(\mathsf{view})$$

where chain denotes an extracted ideal blockchain where each block contains an ordered list of transactions. Sleepy nodes stop outputting to the environment until they wake up again.

3.1 Chain Growth

The first desideratum is that the chain grows proportionally with the number of time steps. Let,

$$\mathsf{min\text{-}chain\text{-}increase}_{t,t'}(\mathsf{view}) = \min_{i,j} |\mathsf{chain}_j^{t+t'}(\mathsf{view})| - |\mathsf{chain}_i^t(\mathsf{view})|$$

$$\mathsf{max\text{-}chain\text{-}increase}_{t,t'}(\mathsf{view}) = \max_{i,j} |\mathsf{chain}_j^{t+t'}(\mathsf{view})| - |\mathsf{chain}_i^t(\mathsf{view})|$$

where we quantify over nodes i, j such that i is alert at view^t and j is alert at $\mathsf{view}^{t+t'}$.

Let $\mathsf{growth}^{t_0,t_1}(\mathsf{view}, \Delta, T) = 1$ iff the following two properties hold:

- **(consistent length)** for all time steps $t \leq |\mathsf{view}| - \Delta$, $t + \Delta \leq t' \leq |\mathsf{view}|$, for every two players i, j such that in view i is alert at t and j is alert at t', we have that $|\mathsf{chain}_j^{t'}(\mathsf{view})| \geq |\mathsf{chain}_i^t(\mathsf{view})|$
- **(chain growth lower bound)** for every time step $t \leq |\mathsf{view}| - t_0$, we have

$$\mathsf{min\text{-}chain\text{-}increase}_{t,t_0}(\mathsf{view}) \geq T.$$

- **(chain growth upper bound)** for every time step $t \leq |\mathsf{view}| - t_1$, we have

$$\mathsf{max\text{-}chain\text{-}increase}_{t,t_1}(\mathsf{view}) \leq T.$$

In other words, $\mathsf{growth}^{t_0,t_1}$ is a predicate which tests that a) alert parties have chains of roughly the same length, and b) during any t_0 time steps in the execution, all alert parties' chains increase by at least T, and c) during any t_1 time steps in the execution, alert parties' chains increase by at most T.

Definition 1 (Chain growth)**.** A blockchain protocol Π satisfies (T_0, g_0, g_1)-chain growth, if for all compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$, $T \geq T_0$, $t_0 \geq \frac{T}{g_0}$ and $t_1 \leq \frac{T}{g_1}$ the following holds:

$$\Pr\left[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{growth}^{t_0,t_1}(\mathsf{view}, \Delta, \kappa) = 1\right] \geq 1 - \mathsf{negl}(\kappa)$$

3.2 Chain Quality

The second desideratum is that the number of blocks contributed by the adversary is not too large.

Given a chain, we say that a block $B := \mathsf{chain}[j]$ is honest w.r.t. view and prefix $\mathsf{chain}[:j']$ where $j' < j$ if in view there exists some node i alert at some time $t \leq |\mathsf{view}|$, such that 1) $\mathsf{chain}[:j'] \prec \mathsf{chain}_i^t(\mathsf{view})$, and 2) $\mathcal{Z}$ input B to node i at time t. Informally, for an honest node's chain denoted chain, a block $B := \mathsf{chain}[j]$ is honest w.r.t. a prefix $\mathsf{chain}[:j']$ where $j' < j$, if earlier there is some alert node who received B as input when its local chain contains the prefix $\mathsf{chain}[:j']$.

Let $\mathsf{quality}^T(\mathsf{view}, \mu) = 1$ iff for every time t and every player i such that i is alert at t in view, among any consecutive sequence of T blocks $\mathsf{chain}[j+1..j+T] \subseteq \mathsf{chain}_i^t(\mathsf{view})$, the fraction of blocks that are honest w.r.t. view and $\mathsf{chain}[:j]$ is at least μ.

Definition 2 (Chain quality)**.** A blockchain protocol Π has $(T_0, \mu)-$*chain quality*, if for all compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$ and every $T \geq T_0$ the following holds:

$$\Pr\left[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{quality}^T(\mathsf{view}, \mu) = 1\right] \geq 1 - \mathsf{negl}(\kappa)$$

3.3 Consistency

Roughly speaking, consistency stipulates common prefix and future self-consistency. Common prefix requires that all honest nodes' chains, except for roughly $O(\kappa)$ number of trailing blocks that have not stabilized, must all agree. Future self-consistency requires that an honest node's present chain, except for roughly $O(\kappa)$ number of trailing blocks that have not stabilized, should persist into its own future. These properties can be unified in the following formal definition (which additionally requires that at any time, two alert nodes' chains must be of similar length).

Let $\mathsf{consistent}^T(\mathsf{view}) = 1$ iff for all times $t \leq t'$, and all players i, j (potentially the same) such that i is alert at t and j is alert at t' in view, we have that the prefixes of $\mathsf{chain}_i^t(\mathsf{view})$ and $\mathsf{chain}_j^{t'}(\mathsf{view})$ consisting of the first $\ell = |\mathsf{chain}_i^t(\mathsf{view})| - T$ records are identical — this also implies that the following must be true: $\mathsf{chain}_j^{t'}(\mathsf{view}) > \ell$, i.e., $\mathsf{chain}_j^{t'}(\mathsf{view})$ cannot be too much shorter than $\mathsf{chain}_i^t(\mathsf{view})$ given that $t' \geq t$.

Definition 3 (Consistency)**.** A blockchain protocol Π satisfies T_0-*consistency*, if for all compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists some negligible function negl such that for every $\kappa \in \mathbb{N}$ and every $T \geq T_0$ the following holds:

$$\Pr\left[\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{consistent}^T(\mathsf{view}) = 1\right] \geq 1 - \mathsf{negl}(\kappa)$$

Note that a direct consequence of consistency is that at any time, the chain *lengths* of any two alert players can differ by at most T (except with negligible probability).

4 Sleepy Consensus

In this section, we will describe our Sleepy consensus protocol. For simplicity, we will describe our scheme pretending that there is a random oracle H. Later, we observe that it is not hard to replace the random oracle with a common reference string and a pseudo-random function. We assume that the random oracle H instance is not shared with other protocols, and that the environment $\mathcal{Z}$ is not allowed to query the random oracle H directly, although it can query the oracle indirectly through $\mathcal{A}$.

4.1 Format of Real-World Blocks

Before we describe our protocol, we first define the format of valid blocks and valid blockchains.

We use the notation *chain* to denote a real-world blockchain. Our protocol relies on an $\mathsf{extract}$ function that extracts an ordered list of transactions from *chain* which alert nodes shall output to the environment $\mathcal{Z}$ at each time step. A blockchain is obviously a chain of blocks. We now define a valid block and a valid blockchain.

Valid blocks. We say that a tuple

$$B := (h_{-1}, \mathsf{txs}, \mathsf{time}, \mathcal{P}, \sigma, h)$$

is a valid block iff

1. $\Sigma.\mathsf{Ver}_{\mathsf{pk}}((\mathsf{h}_{-1}, \mathsf{txs}, \mathsf{time}); \sigma) = 1$ where $\mathsf{pk} := \mathcal{F}_{\mathrm{CA}}.\mathtt{lookup}(\mathcal{P})$; and
2. $h = \mathsf{d}(h_{-1}, \mathsf{txs}, \mathsf{time}, \mathcal{P}, \sigma)$, where $\mathsf{d} : \{0,1\}^* \to \{0,1\}^\kappa$ is a collision-resistant hash function — technically collision resistant hash functions must be defined for a family, but here for simplicity we pretend that the sampling from the family has already been done before protocol start, and therefore d is a single function.

Valid blockchain. Let $\mathsf{eligible}^t(\mathcal{P})$ be a function that determines whether a party $\mathcal{P}$ is an eligible leader for time step t (see Figure 1 for its definition). Let *chain* denote an ordered chain of real-world blocks, we say that *chain* is a valid blockchain w.r.t. $\mathsf{eligible}$ and time t iff

- $\mathit{chain}[0] = \mathit{genesis} = (\bot, \bot, \mathsf{time} = 0, \bot, \bot, h = \vec{0})$, commonly referred to as the genesis block;
- $\mathit{chain}[-1].\mathsf{time} \leq t$; and
- for all $i \in [1..\ell]$, the following holds:
  1. $\mathit{chain}[i]$ is a valid block;
  2. $\mathit{chain}[i].h_{-1} = \mathit{chain}[i-1].h$;
  3. $\mathit{chain}[i].\mathsf{time} > \mathit{chain}[i-1].\mathsf{time}$, i.e., timestamps are strictly increasing; and
  4. let $t := \mathit{chain}[i].\mathsf{time}$, $\mathcal{P} := \mathit{chain}[i].\mathcal{P}$, it holds that $\mathsf{eligible}^t(\mathcal{P}) = 1$.

Protocol $\Pi_{\text{sleepy}}(p)$

On input `init`() from $\mathcal{Z}$:

let $(\mathsf{pk}, \mathsf{sk}) := \Sigma.\mathsf{gen}()$, register pk with $\mathcal{F}_{\text{CA}}$, let $chain := genesis$

On receive $chain'$:

assert $|chain'| > |chain|$ and $chain'$ is valid w.r.t. $\mathsf{eligible}$ and the current time t;

$chain := chain'$ and gossip $chain$

Every time step:

- receive input `transactions`(txs) from $\mathcal{Z}$
- let t be the current time, if $\mathsf{eligible}^t(\mathcal{P})$ where $\mathcal{P}$ is the current node's party identifier:

  let $\sigma := \Sigma.\mathtt{sign}(\mathsf{sk}, chain[-1].h, \mathsf{txs}, t)$, $h' := \mathsf{d}(chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \sigma)$,

  let $B := (chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \sigma, h')$, let $chain := chain||B$ and gossip $chain$
- output $\mathsf{extract}(chain)$ to $\mathcal{Z}$ where $\mathsf{extract}$ is the function outputs an ordered list containing the txs extracted from each block in $chain$

Subroutine $\mathsf{eligible}^t(\mathcal{P})$:

return 1 if $\mathsf{H}(\mathcal{P}, t) < D_p$ and $\mathcal{P}$ is a valid party of this protocol; else return 0

Figure 1: The sleepy consensus protocol. The difficulty parameter D_p is defined such that the hash outcome is less than D_p with probability p

4.2 The Sleepy Consensus Protocol

We present our basic Sleepy consensus protocol in Figure 1. The protocol takes a parameter p as input, where p corresponds to the probability each node is elected leader in a single time step. All nodes that just spawned will invoke the `init` entry point. During initialization, a node generates a signature key pair and registers the public key with the public-key infrastructure $\mathcal{F}_{\text{CA}}$.

Now, our basic Sleepy protocol proceeds very much like a proof-of-work blockchain, except that instead of solving computational puzzles, in our protocol a node can extend the chain at time t iff it is elected leader at time t. To extend the chain with a block, a leader of time t simply signs a tuple containing the previous block's hash, the node's own party identifier, the current time t, as well as a set of transactions to be confirmed. Leader election can be achieved through a public hash function H that is modeled as a random oracle.

Remark on how to interpret the protocol for weakly synchronized clocks. As mentioned earlier, in practice, we would typically adopt the protocol assuming nodes have weakly synchronized clocks instead of perfect synchronized clocks. Section 2.1 described a general protocol transformation that allows us to treat weakly synchronized clocks as synchronized clocks in formal reasoning (but adopting a larger network delay). Specifically, when deployed in practice assuming weakly synchronized clocks with up to Δ clock offset, alert nodes would actually queue each received message for Δ time before locally delivering the message. This ensures that alert nodes will not reject

other alert nodes' chains mistakenly thinking that the timestamp is in the future (due to clock offsets).

4.3 Compliant Executions

Our protocol can be proven secure as long as a set of constraints are expected, such as the number of alert vs. corrupt nodes. Below we formally define the complete set of rules that we expect $(\mathcal{A}, \mathcal{Z})$ to respect to prove security.

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{sleepy}}(p)$-compliant if the following holds for any $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{sleepy}}}(\mathcal{A}, \mathcal{Z}, \kappa)$ with non-zero support — henceforth whenever the context is clear, we often say that $(\mathcal{A}, \mathcal{Z})$ is Π_{sleepy}-compliant omitting the protocol parameter p.

- **Initialization.** At the start of the execution, the following happens. First, $\mathcal{Z}$ can spawn a set of either honest or corrupt nodes. Next, $\mathcal{Z}$ issues `sleep` instructions and declares for each honest node exactly when they will sleep.

  At this point, the protocol execution starts. $\mathcal{A}$ cannot query the random oracle H prior to protocol start.

- **Number of awake nodes.** Let alert^t and $\mathsf{corrupt}$ be defined as below:

  - $\mathsf{alert}^t(\mathsf{view})$ outputs the number of nodes that are alert at time t in view.
  - $\mathsf{corrupt}(\mathsf{view})$ outputs the number of corrupt nodes in view.

  For every honest node i that is honest at time t in view, it must hold that

  $$\mathsf{alert}^t(\mathsf{view}) + \mathsf{corrupt}(\mathsf{view}) = n$$

  While we make this simplifying assumption, note that it is not hard to extend the proof to allow up to a constant factor variation in n.

- **Resilience.** We require that there exists a constant $\phi > 0$, such that for every $t \leq |\mathsf{view}|$,

  $$\frac{\mathsf{alert}^t(\mathsf{view})}{\mathsf{corrupt}(\mathsf{view})} \geq 1 + \phi$$

  Informally, we require that at any point of time, there are more alert nodes than corrupt ones by a ϕ margin, where ϕ is an arbitrarily small constant.

- **Admissible parameters.** The parameters (p, n, ϕ, Δ) are Π_{sleepy}-admissible (to be defined later), where p is an input parameter to Π_{sleepy} whereas the rest are parameters determined by $(\mathcal{A}, \mathcal{Z})$.

We now define what it means for the parameters (p, n, ϕ, Δ) to be admissible. Before doing so, we first define some useful notations.

Useful notations. Recall that p is the probability that a node is elected leader in a given time step. $1+\phi$ is the minimum ratio of alert nodes over corrupt ones across time. n is the total number of awake nodes at any given time. We define a set of intermediate variables α, β, and γ which are defined as functions of p, n, ϕ, and possibly Δ. These will become useful later.

1. Let $\alpha := 1-(1-p)^{\frac{n(1+\phi)}{2+\phi}}$ be the probability that some alert node is elected leader in one round; and
2. Let $\beta := 1-(1-p)^{\frac{n}{2+\phi}}$ be the probability that some corrupt node is elected leader in one round;
3. Let $\gamma := \frac{\alpha}{1+\Delta\alpha}$. γ is a "discounted" version of α which takes into account the fact that messages sent by alert nodes can be delayed by Δ time steps; γ corresponds to alert nodes' "effective" proportion among all awake nodes.

Admissible parameters. We say that the parameters (p, n, ϕ, Δ) are Π_{sleepy}-admissible iff the following holds:

- $np\Delta < 1$
- There exists a constant ψ such that

$$(1-2\alpha(\Delta+1))\alpha > (1+\psi)\beta \tag{1}$$

  where α and β are earlier in this section.

4.4 Theorem Statement

Theorem 2 (Security of Π_{sleepy})**.** For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0\kappa$, Π_{sleepy} satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0 consistency against any Π_{sleepy}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, with the following parameters:

- chain growth lower bound parameter $g_0 = (1-\epsilon)\gamma$;
- chain growth upper bound parameter $g_1 = (1+\epsilon)np$; and
- chain quality parameter $\mu = (1-\epsilon)(1-\frac{\beta}{\alpha})$;

where α, β, γ are defined as in Section 4.3.

The proof of this theorem will be presented in Sections 6 and 7.

Remark on removing the random oracle. Although we described our scheme assuming a random oracle H, it is not hard to observe that we can replace the random oracle with a common reference string crs and a pseudo-random function PRF. Specifically, the common reference string crs is randomly generated after $\mathcal{Z}$ spawns all corrupt nodes and commits to when each honest node shall sleep. Then, we can simply replace calls to $\mathsf{H}(\cdot)$ with with $\mathsf{PRF}_{\mathsf{crs}}(\cdot)$.

5 Lower Bound

We show that in the sleepy model, honest majority (among awake nodes) is necessary for achieving consensus. Intuitively, imagine that there is a sleepy node who sleeps from protocol start to some time t^* at which point it wakes up. If there are more corrupt nodes than alert ones, the adversary can always simulate a fake execution trace that is identically distributed as the real one; and now the sleepy node that just woke up cannot discern which one is real and which one simulated.

Although we state our theorem for blockchain protocols, the same lower bound (with identical proofs) holds for a broader class of consensus protocols (for a formulation in the UC framework, see Pass and Shi's definition [38]). Pass and Shi [38] also show that the blockchain abstraction implies a general consensus (i.e., state machine replication) abstraction.

Theorem 3 (Majority honest is necessary)**.** In the sleepy execution model, it is not possible to realize a blockchain protocol if there can be as many corrupt nodes than alert nodes — and this lower bound holds even assuming static corruption and the existence of a public-key infrastructure.

Proof. For any protocol that achieves liveness (or in the case of blockchains, chain growth), there exists a $(\mathcal{A}, \mathcal{Z})$ pair that does the following that can break consistency with constant probability if there are as many corrupt nodes as alert ones.

- At the beginning of protocol execution, $\mathcal{Z}$ spawns k alert nodes, and k corrupt ones as well. Additionally, $\mathcal{Z}$ spawns a sleepy node denoted i^* and makes it sleep from protocol start to some future time t^*.
- When protocol execution starts, $\mathcal{A}$ first has all corrupt nodes remain silent and not participate in the actual protocol execution;
- However, $\mathcal{A}$ simulates a protocol execution with the k corrupt nodes. Suppose that $\mathcal{Z}$ generates transaction inputs following some distribution $\mathcal{D}$ for the real execution. Now $\mathcal{A}$ uses the same distribution to generate simulated transactions for the simulated execution. We henceforth assume that two random samples from $\mathcal{D}$ are different with constant probability.
- When the sleepy node i^* wakes up at time t^*, $\mathcal{A}$ delivers node i protocol messages from both the real and simulated executions.
- Since the real and simulated executions are identically distributed to the newly joining node i, there cannot exist an algorithm that can output the correct log with probability more than $\frac{1}{2}$.

□

6 Proofs: Analyzing A Simplified Ideal Protocol

In this section, we start by analyzing a very simple ideal protocol denoted Π_{ideal}, where nodes interact with an ideal functionality $\mathcal{F}_{\text{tree}}$ that keeps track of all valid chains at any moment of time. Later in Section 7, we will show that the real-world protocol Π_{sleepy} securely emulates the ideal-world protocol.

$\mathcal{F}_{\text{tree}}(p)$

On `init`: $\mathsf{tree} := \mathsf{genesis}$, $\mathsf{time}(\mathsf{genesis}) := 0$

On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{A}$ or internally:

if $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] := \begin{cases} 1 & \text{with probability } p \\ 0 & \text{o.w.} \end{cases}$

return $\Gamma[\mathcal{P}, t]$

On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B})$ from $\mathcal{P}$: let t be the current time:

assert $\mathsf{chain} \in \mathsf{tree}$, $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$, and $\texttt{leader}(\mathcal{P}, t)$ outputs 1

append B to chain in tree, record $\mathsf{time}(\mathsf{chain}||\mathsf{B}) := t$, and return "succ"

On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B}, t')$ from corrupt party $\mathcal{P}^*$: let t be the current time

assert $\mathsf{chain} \in \mathsf{tree}$, $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$, $\texttt{leader}(\mathcal{P}^*, t')$ outputs 1, and $\mathsf{time}(\mathsf{chain}) < t' \leq t$

append B to chain in tree, record $\mathsf{time}(\mathsf{chain}||\mathsf{B}) = t'$, and return "succ"

On receive $\texttt{verify}(\mathsf{chain})$ from $\mathcal{P}$: return $(\mathsf{chain} \in \mathsf{tree})$

Figure 2: Ideal functionality $\mathcal{F}_{\text{tree}}$.

6.1 Simplified Ideal Protocol Π_{ideal}

We first define a simplified protocol Π_{ideal} parametrized with an ideal functionality $\mathcal{F}_{\text{tree}}$ — see Figures 2 and 3. $\mathcal{F}_{\text{tree}}$ flips random coins to decide whether a node is the elected leader for every time step, and an adversary $\mathcal{A}$ can query this information through the `leader` query interface. Finally, alert and corrupt nodes can call $\mathcal{F}_{\text{tree}}.\texttt{extend}$ to extend known chains with new blocks if they are the elected leader for a specific time step. $\mathcal{F}_{\text{tree}}$ keeps track of all valid chains, such that alert nodes will call $\mathcal{F}_{\text{tree}}.\texttt{verify}$ to decide if any chain they receive is valid. Alert nodes always store the longest valid chains they have received, and try to extend it.

Notations. Given some view sampled from $\mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, we say that a $\mathsf{chain} \in \mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{tree}$ has an $\mathcal{F}_{\text{tree}}$-timestamp (or simply timestamp for short) of t if $\mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{time}(\mathsf{chain}) = t$. When an adversary extends a chain, $\mathcal{F}_{\text{tree}}$ enforces that the chain must have strictly increasing timestamps, and that the chain's timestamp must be no greater than the current time.

We say that a node $\mathcal{P}$ (alert or corrupt) mines a $\mathsf{chain}' = \mathsf{chain}||\mathsf{B}$ in time t if $\mathcal{P}$ called $\mathcal{F}_{\text{tree}}.\texttt{extend}(\mathsf{chain}, \mathsf{B})$ or $\mathcal{F}_{\text{tree}}.\texttt{extend}(\mathsf{chain}, \mathsf{B}, \_)$ at time t, and the call returned "succ". Note that if an alert node mines a chain at time t, then the chain's $\mathcal{F}_{\text{tree}}$-timestamp must be t as well. By contrast, if a corrupt node mines a chain at time t, the chain's timestamp may not be truthful — it may be smaller than t.

We say that $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{ideal}}(p)$-compliant iff the pair is $\Pi_{\text{sleepy}}(p)$-compliant. Since the protocols' compliance rules are the same, we sometimes just write compliant for short.

Theorem 4 (Security of Π_{ideal})**.** For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \kappa$, Π_{sleepy} satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0 consistency against any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, with the following parameters:

Protocol Π_{ideal}

On `init`: $\mathsf{chain} := \mathsf{genesis}$

On receive chain': if $|\mathsf{chain}'| > |\mathsf{chain}|$ and $\mathcal{F}_{\text{tree}}.\texttt{verify}(\mathsf{chain}') = 1$: $\mathsf{chain} := \mathsf{chain}'$, gossip chain

Every time step:

- receive input B from $\mathcal{Z}$
- if $\mathcal{F}_{\text{tree}}.\texttt{extend}(\mathsf{chain}, \mathsf{B})$ outputs "succ": $\mathsf{chain} := \mathsf{chain}||\mathsf{B}$ and gossip chain
- output chain to $\mathcal{Z}$

Figure 3: Ideal protocol Π_{ideal}

- chain growth lower bound parameter $g_0 = (1-\epsilon)\gamma$;
- chain growth upper bound parameter $g_1 = (1+\epsilon)np$; and
- chain quality parameter $\mu = (1-\epsilon)(1-\frac{\beta}{\alpha})$;

where α, β, γ are defined as in Section 4.3.

We will now prove the above Theorem 4.

Intuitions and differences from Nakamoto's ideal protocol. The key difference between our ideal protocol and Nakamoto's ideal protocol as described by Pass et al. [36] is the following. In Nakamoto's ideal protocol, if the adversary succeeds in extending a chain with a block, he cannot reuse this block and concatenate it with other chains. Here in our ideal protocol, if a corrupt node is elected leader in some time step, he can extend many possible chains. He can also instruct $\mathcal{F}_{\text{tree}}$ to extend chains with timestamps in the past, as long as the chain's timestamps are strictly increasing.

Although our $\mathcal{F}_{\text{tree}}$ allows the adversary to claim potentially false timestamps, we can rely on the following timestamps invariants in our proofs: 1) honest blocks always have faithful $\mathcal{F}_{\text{tree}}$-timestamps; and 2) any chain in $\mathcal{F}_{\text{tree}}$ must have strictly increasing timestamps. Having observed these, we show that Pass et al.'s chain growth and chain quality proofs [36] can be easily adapted for our scenario.

Unfortunately, the main challenge is how to prove consistency. As mentioned earlier, our adversary is much more powerful than the adversary for the Nakamoto blockchain and can launch a much wider range of attacks where he reuses the time slots during which he is elected. In Sections 6.4 and 6.5, we present new techniques for analyzing the induced stochastic process.

6.2 Chain Growth Lower Bound

The chain growth lower bound proof is almost identical to that of Pass et al. [36]. The only difference is that in their Lemma 6.3 which is an inductive proof, the case where j is "corrupt" at time $s-1$ in REAL is replaced with j is "asleep" at time $s-1$ in REAL. The rest of the proof all follows literally.

6.3 Chain Quality

Chain quality proof can be done in a similar manner as Pass et al. [36], however, we need to reinterpret the definitions of a few random variables. We present the modified proof below.

First, we prove two simple facts that upper bound the number of effective time steps (in which at least one alert or corrupt node is elected leader), and the number of adversarial time steps (in which at least one corrupt node is elected leader).

Given a view, let $\mathbf{Q}(\mathsf{view})[t_0 : t_1]$ denote the number of time steps in which at least one alert or corrupt node is elected leader. The following fact upper bounds the effective time steps, i.e., a time step in which at least one alert or corrupt node is elected leader.

Fact 1 (Upper bound on effective time slots)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any t, any $t_0 \leq t_1 \leq |\mathsf{view}|$ such that $t_1 - t_0 = t$, for any positive constant ϵ and any κ,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{Q}(\mathsf{view})[t_0 : t_1] > (1+\epsilon)npt\right] \leq \exp(-\frac{\epsilon^2 npt}{3})$$

Proof. By a straightforward application of the Chernoff bound. □

Given a view, let $\mathbf{A}(\mathsf{view})[t_0 : t_1]$ denote the number of time steps in which at least one corrupt node is elected leader during the window $[t_0 : t_1]$.

Fact 2 (Upper bound on adversarial time slots)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any t, any $t_0 \leq t_1 \leq |\mathsf{view}|$ such that $t_1 - t_0 = t$, for any positive constant ϵ and any κ,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{A}(\mathsf{view})[t_0 : t_1] > (1+\epsilon)\beta t\right] \leq \exp(-\frac{\epsilon^2 \beta t}{3})$$

Proof. Due to a straightforward application of the Chernoff bound. □

For convenience, we will also define the following related variables Q_t and A_t:

$$Q_t(\mathsf{view}) := \max\{|\mathsf{chain}| \;\big|\; \mathsf{chain} \in \mathcal{F}_{\text{tree}}.\mathsf{tree}, \mathsf{time}(\mathsf{chain}[:-1]) - \mathsf{time}(\mathsf{chain}[:1]) \leq t\}$$

where $\mathcal{F}_{\text{tree}} := \mathcal{F}_{\text{tree}}(\mathsf{view})$ and $\mathsf{time}(\mathsf{chain}) := \mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{time}(\mathsf{chain})$ denotes the $\mathcal{F}_{\text{tree}}$-timestamp of chain — we often omit writing view for simplicity. In other words, Q_t is the maximum number of blocks in any $\mathsf{chain} \in \mathcal{F}_{\text{tree}}.\mathsf{tree}$ *not necessarily rooted at genesis* such that the beginning and ending timestamps are at most t apart.

Similar, we redefine the random variable $A_t(\mathsf{view})$ to mean the following:

$$A_t(\mathsf{view}) := \max\{a(\mathsf{chain}) \;\big|\; \mathsf{chain} \in \mathcal{F}_{\text{tree}}.\mathsf{tree}, \mathsf{time}(\mathsf{chain}[:-1]) - \mathsf{time}(\mathsf{chain}[:1]) \leq t\}$$

where $a(\mathsf{chain})$ returns the number of blocks in chain that were mined by $\mathcal{A}$.

Note that due to a simple union bound as well as Facts 1 and 2, we have the following fact — which is analogous to Lemmas 6.7 and 6.8 of Pass et al. [36]:

Fact 3. For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any positive t, for any positive constant ϵ and any κ,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : Q_t(\mathsf{view}) > (1+\epsilon)npt\right] \leq \exp(-\frac{\epsilon^2 npt}{3}) \cdot |\mathsf{view}|$$

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : A_t(\mathsf{view}) > (1+\epsilon)\beta t\right] \leq \exp(-\frac{\epsilon^2 \beta t}{3}) \cdot |\mathsf{view}|$$

Chain quality proof. Now, just like Pass et al. [36], we can proceed to prove chain quality — below we will try to preserve the same notation as in Pass et al.'s chain quality proof. We can consider the $\mathsf{chain} := \mathsf{chain}_i^r(\mathsf{view})$ of any node i honest at any time $r \leq |\mathsf{view}|$, and a sequence of T blocks $\mathsf{chain}[j+1..j+T] \subset \mathsf{chain}_i^r$, such that $\mathsf{chain}[j]$ is not adversarial (either an honest block or genesis); and $\mathsf{chain}[j+T+1]$ is not adversarial either (either an honest block or $\mathsf{chain}[j+T]$ is end of chain_i^r). Note that for an honest block, its $\mathcal{F}_{\text{tree}}$-timestamp must be faithful, i.e., corresponding to the time step in which the block was mined (recall that the $\mathcal{F}_{\text{tree}}$-timestamp of $\mathsf{genesis}$ is 0). Consequently, by definition of Π_{ideal} and $\mathcal{F}_{\text{tree}}$, the $\mathcal{F}_{\text{tree}}$-timestamps of all blocks in $\mathsf{chain}[j+1..j+T]$ must be bounded in between r' and $r'+t$, where r' denotes the time step in which the honest (or genesis) block $\mathsf{chain}[j]$ was mined, and $r'+t$ denotes the time step in which $\mathsf{chain}[j+T+1]$ is mined (or let $r'+t := r$ if $\mathsf{chain}[j+T]$ is end of chain_i^r).

The rest of the proofs follow in exactly the same manner as Pass et al.'s chain quality proof — except that whenever they apply their Lemmas 6.7 or 6.8, we can now plug in our new definitions of the random variables $Q_t(\mathsf{view})$ and $A_t(\mathsf{view})$.

6.4 Consistency: Proof Intuition

Since this is the most non-trivial part of our proof and where we significantly depart from earlier blockchain proofs [21, 36], we will first explain the intuition before presenting the formal proof.

Review: consistency proof for the Nakamoto blockchain. We first review how Pass et al. [36] proved consistency for the Nakamoto blockchain, and explain why their proof fails in our setting. This will help to clarify the challenges of the proof. To prove consistency, Pass et al. defines the notion of a convergence opportunity. A convergence opportunity is a period of time in which 1) there is a Δ-long period of silence in which no honest node mines a block; and 2) followed by a time step in which a *single* honest node mines a block; and 3) followed by yet another Δ-long period of silence in which no honest node mines a block. Whenever there is a convergence period, and suppose that at the beginning of the convergence period the maximum chain length of any honest node is ℓ. Then, it is not hard to see that there can be at most one honest block (if any) in position $\ell+1$ in any honest node's chain — since after the first period of silence, all honest nodes' chain must be of length at least ℓ; and after the second period of silence, all honest nodes' chain length must be at least $\ell+1$. Therefore, after the convergence period, no honest node will ever mine at position $\ell+1$ again. However, recall that within the convergence period, only a single honest node ever mines a block.

Now, Pass et al. [36] observes that for the adversary to cause divergence at some time s or earlier, for every convergence opportunity after time s, the adversary must mine a chain of length

$\ell + 1$ where ℓ is the maximum chain length of any honest node at the beginning of the convergence period. This means that from time

$$(s - [\text{small block withholding window}])$$

onward, the adversary must have mined more blocks than the number of convergence opportunities since s.

Pass et al. [36] then goes to show that if s is sufficiently long ago, this cannot happen — in other words, there has to be more convergence opportunities than adversarially mined blocks in any time window, even when adjusted for block withholding attacks. Proving an upper bound on adversarially mined blocks in any window is relatively easy, therefore most of their proof focuses on lower bounding the number of convergence opportunities within any time window.

Why their proof breaks in our setting. The consistency proof by Pass et al. [36] crucially relies on the following fact: when an adversary successfully extends a chain with a block, he cannot simply transfer this block at no cost to extend any other chain. For this reason, to mine a chain of length $\ell + 1$ for each different ℓ will require separate computational effort, and no effort can ever be reused.

This crucial observation fails to hold in our protocol. If a corrupt node is elected in a certain time step t, he can now use this earned time slot to extend multiple chains, *possibly at different lengths*. Recall that Pass et al's consistency proof relies on arguing that the adversary cannot have mined chains of many different lengths. Unfortunately, in our case, such an argument will not work. In particular, how many times the adversary is elected leader (the direct analogy of how many times an adversary mines a block in a proof-of-work blockchain) does not translate to how many chain lengths the adversary can attack (by composing an adversarial chain of that length). It now appears that a fundamentally new proof strategy is necessary.

Roadmap of our proof. Our proof strategy is the following. We will define a good event called a pivot point. Roughly speaking, a pivot point is a point of time t, such that if one draws any segment of time $[t_0, t_1]$ that contains t, the number of adversarial time slots in that window is smaller than the number of convergence opportunities. We show that if there is such a pivot point t in view, the adversary cannot have caused divergence prior to t. We then show that pivot points happen every now and then, and particularly, in any sufficiently long time window there must exist such a pivot point. This then implies that if one removes sufficiently many trailing blocks from an alert node's chain (recall that by chain growth, block numbers and time roughly translate to each other), the remaining prefix must be consistent with any other alert node.

6.5 Consistency: the Proof

Convergence opportunity. Given a view, we say that $[T - \Delta, T_\delta]$ is a convergence opportunity iff

- For any $t \in [T - \Delta, T)$, no node alert at time t is elected leader;
- A single node alert at T is elected leader at time T;
- For any $t \in (T, T + \Delta]$, no node alert at time t is elected leader.

In other words, a convergence opportunity is a Δ-period of silence in which no alert node is elected leader, followed by a time step in which a single alert node is elected leader, followed by another Δ-period of silence in which no alert node is elected leader.

Let T denote the time in which a single alert node is elected leader during a convergence opportunity. For convenience, we often use T to refer to the convergence opportunity. We say that a convergence opportunity T is contained within a window $[t' : t]$ if $T \in [t' : t]$.

Henceforth, we use the notation $\mathbf{C}(\mathsf{view})[t' : t]$ to denote the number of convergence opportunities contained within the window $[t' : t]$ in view. We use the notation $\mathbf{A}(\mathsf{view})[t' : t]$ to denote the number of time steps in which corrupt nodes are elected leader during $[t' : t]$ in view.

Backward pivot. Given a view, a time step t is said to be a backward pivot in view, if for any $t' \leq t$, it holds that $\mathbf{C}(\mathsf{view})[t' : t] > \mathbf{A}(\mathsf{view})[t' : t]$ or $\mathbf{A}(\mathsf{view})[t' : t] = 0$.

Forward pivot. Given a view, a time step t is said to be a forward pivot in view, if for any $t' \geq t$, it holds that $\mathbf{C}(\mathsf{view})[t : t'] > \mathbf{A}(\mathsf{view})[t' : t]$ or $\mathbf{A}(\mathsf{view})[t' : t] = 0$.

Pivot. Given a view, a time step is said to be a pivot in view if it is both a backward and a forward pivot in view.

It is not hard to see that an equivalent definition for a pivot point is the following: given a view, a time step is said to be a pivot in view if for any $t_0 \leq t \leq t_1$, it holds that $\mathbf{C}(\mathsf{view})[t_0 : t_1] > \mathbf{A}(\mathsf{view})[t_0 : t_1]$ or $\mathbf{A}(\mathsf{view})[t_0 : t_1] = 0$.

Divergence. Given any two chains $\mathsf{chain}_0, \mathsf{chain}_1 \in \mathcal{F}_{\mathrm{tree}}.\mathsf{tree}$, we say that they diverge at time t if their longest common prefix has an $\mathcal{F}_{\mathrm{tree}}$-timestamp before t.

Fact 4 (Uniqueness of an honest block in any convergence opportunity)**.** Given any view, let i be honest at time r and j be honest at $r' \geq r$ in view. Suppose there is a convergence opportunity $T < r - \Delta$ in view, and let ℓ denote the maximum chain length of an alert node at $T - \Delta$, it holds that if $\mathsf{chain}_i^r(\mathsf{view})[: \ell + 1]$ and $\mathsf{chain}_j^{r'}(\mathsf{view})[: \ell + 1]$ are both honest, then they must be the same.

Proof. This was proved by Pass et al. [36], the same proof applies to our setting. □

Lemma 1 (There are many convergence opportunities)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any $0 < t_0 < t_1 \leq |\mathsf{view}|$, any positive κ, any positive constant ϵ, there exists a constant ϵ' that depends on ϵ, the following holds where $t := t_1 - t_0$:

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\mathrm{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{C}(\mathsf{view})[t_0 : t_1] < (1 - \epsilon)(1 - 2\alpha(\Delta + 1))\alpha t\right] < \exp(-\epsilon' \beta t)$$

Proof. Same as the proof by Pass et al. [36]. □

We remark that this above lemma is the core technical contribution of Pass et al. [36]'s consistency proof. Our proof will build upon their conclusions — but as we show, on top of their analayis, we would need non-trivial new techniques to prove consistency for Sleepy.

Lemma 2 (Divergence cannot happen before a pivot)**.** Let view be any execution trace where the bad events related to Lemma 1 do not happen. Let i be honest at time r and j be honest at $r' \geq r$ in view, if $0 < t < r - \frac{\kappa}{\beta}$ is a pivot in view, then chain_i^r and $\mathsf{chain}_j^{r'}$ cannot diverge at t in view.

Proof. First, we perform the following post-processing each $\mathsf{chain} \in \{\mathsf{chain}_i^r, \mathsf{chain}_j^{r'}\}$:

- Suppose there are c convergence opportunities before $r - \Delta$. Let $\ell_1, \ell_2, \ldots, \ell_c$ denote the maximum chain length of an alert node at the beginning of each convergence period.
- $\mathsf{chain} \leftarrow \mathsf{chain}[\ell_1 + 1, \ell_2 + 1, \ldots, \ell_c + 1]$, i.e., extract positions $\ell_1 + 1, \ell_2 + 1, \ldots, \ell_c + 1$ from chain.

At the end of this post-processing, now chain_i^r and $\mathsf{chain}_j^{r'}$ contain only positions corresponding to convergence opportunities. It is easy to see that if the original chain_i^r and $\mathsf{chain}_j^{r'}$ diverge at some time t_d, then the post-processed chain_i^r and $\mathsf{chain}_j^{r'}$ also diverge at time t_d. Henceforth, whenever we refer to chain_i^r and $\mathsf{chain}_j^{r'}$, we mean the post-processed version (unless otherwise noted).

Suppose that chain_i^r and $\mathsf{chain}_j^{r'}$ diverge at t, where $t < r - \frac{\kappa}{\beta}$ is a pivot. By Lemma 1, there must be at least one convergence opportunity between the window (t, r). Obviously this also guarantees that there are blocks whose $\mathcal{F}_{\text{tree}}$-timestamps are greater than t in the post-processed chain_i^r and $\mathsf{chain}_j^{r'}$. We now look at the first block $\mathsf{B} \in \mathsf{chain}_i^r$ (or $\mathsf{chain}_j^{r'}$) whose $\mathcal{F}_{\text{tree}}$-timestamp is greater than or equal to t, and we argue that this block must be honest.

For the sake of reaching a contradiction, suppose that this block B is not honest in chain_i^r (the proof for $\mathsf{chain}_j^{r'}$ is the same). We now find the maximal sequence of adversarial blocks $\mathsf{chain}_i^r[a:b]$ such that $\mathsf{B} \in \mathsf{chain}_i^r[a:b]$. Now $\mathsf{chain}_i^r[a-1]$ is an honest block (or genesis), and so is $\mathsf{chain}_i^r[b+1]$ (or b is the last block of chain_i^r which we will treat specially at the end). Since t is a pivot point, the number of times the adversary is elected leader between $(\mathsf{time}(\mathsf{chain}_i^r[a-1]), \mathsf{time}(\mathsf{chain}_i^r[b+1]))$ must be smaller than the number of convergence opportunities within the same time window. This conflicts with the fact that all blocks in $\mathsf{chain}_i^r[a:b]$ are adversarial. Note that here we abuse the notation $\mathsf{time}(\mathsf{chain}_i^r[a-1])$ in the most natural manner — since earlier $\mathsf{time}()$ was defined over chains that have not been post-processed. In case b is the last block of chain_i^r, we abuse notation and define $\mathsf{time}(\mathsf{chain}_i^r[b+1])$ to be r.

Therefore, in both chain_i^r and $\mathsf{chain}_j^{r'}$, the first block with a timestamp $\geq t$ is honest. By a symmetric argument, in both chain_i^r and $\mathsf{chain}_j^{r'}$, the first block with a timestamp $\leq t$ is also honest. Therefore, either t itself is a convergence opportunity and both chains have honest blocks at time t, or t is not a convergence opportunity and t is in between two honest blocks in both chains. Now in both chains, find the first block with timestamp $\geq t$, Due to Fact 4, the prefix up to this block in both chain_i^r and $\mathsf{chain}_j^{r'}$ must be identical. However, this conflicts with our hypothesis that chain_i^r and $\mathsf{chain}_j^{r'}$ at time t. □

Lemma 3 (Adversarial time slots vs. convergence opportunities)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists some positive constant η that depends on ψ, such that for any $0 < t_0 < t_1 \leq |\mathsf{view}|$, for any positive κ, the following holds where $t := t_1 - t_0$:

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{A}(\mathsf{view})[t_0 : t_1] \geq \mathbf{C}(\mathsf{view})[t_0 : t_1]\right] < \exp(-\eta\beta t)$$

Proof. Due to Fact 2, for any positive ϵ_1,

$$\Pr\left[\mathbf{A}[t_0 : t_0 + t] > (1 + \epsilon_1)\beta t\right] < \exp(-\frac{\epsilon_1^2 \beta t}{3})$$

Due to Lemma 1, for any positive ϵ_2, there exists positive ϵ' that depends on ϵ_2, such that

$$\Pr\left[\mathbf{C}[t_0 : t_0 + t] < (1 - \epsilon_2)(1 - 2\alpha(\Delta + 1))\alpha t\right] \leq \exp(-\epsilon'\beta t)$$

Since we know that there exists some constant ψ such that $(1 - 2\alpha(\Delta + 1))\alpha > (1 + \psi)\beta$, it holds that for sufficiently small constants ϵ_1 and ϵ_2, it holds that

$$(1 + \epsilon_1)\beta t < (1 - \epsilon_2)(1 - 2\alpha(\Delta + 1))\alpha t$$

The rest of the proof is straightforward. □

Lemma 4 (Probability that any given time is a backward pivot)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a constant $c(\psi)$ that depends on ψ, such that for any positive κ, for any $0 < t \leq |\mathsf{view}|$,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : t \text{ is a backward pivot in } \mathsf{view}\right] \geq c(\psi)$$

Proof. For simplicity, for $t' < t$, let $\mathsf{bad}(t')$ denote the bad event that $\mathbf{C}[t' : t] \leq \mathbf{A}[t' : t]$. Let η be a suitable constant defined as in Lemma 3. Let $t_c := \max(1, t - \frac{5}{\beta\eta})$.

We have the following:

$$\begin{aligned}
&\Pr[t \text{ is a backward pivot}] \geq \Pr[t \text{ is a backward pivot and } \mathbf{A}[t_c : t] = 0] \\
&\geq \Pr[\mathbf{A}[t_c : t] = 0] \cdot \Pr\left[\text{for any } t' < t_c\text{: } \overline{\mathsf{bad}}(t') \,|\mathbf{A}[t_c : t] = 0\right] \\
&\geq \Pr[\mathbf{A}[t_c : t] = 0] \cdot \Pr\left[\text{for any } t' < t_c\text{: } \overline{\mathsf{bad}}(t')\right] \\
&\geq (1 - \beta)^{t_c} \cdot (1 - \Pr[\mathsf{bad}(t_c - 1)] - \Pr[\mathsf{bad}(t_c - 2)] \ldots - \Pr[\mathsf{bad}(1)]) && \text{union bound} \\
&\geq \left(\frac{1}{4}\right)^{\frac{5}{\eta}} \cdot \left(1 - e^{-5} - e^{-10} - e^{-15} - \ldots\right) && \text{Lemma 3} \\
&\leq \text{const}(\eta) < 1
\end{aligned}$$

where $\text{const}(\eta)$ denotes some constant that depends on η (which in turn depends on ψ). □

Lemma 5 (Probability that any given time is a forward pivot)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a constant $c(\psi)$ that depends on ψ, such that for any positive κ, for any $0 < t \leq |\mathsf{view}|$,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : t \text{ is a forward pivot in } \mathsf{view}\right] \geq c(\psi)$$

Proof. The proof is similar to the proof for backward pivot point, since the definitions are symmetric w.r.t. to the point being considered. □

Corollary 1 (Probability that any time step is a pivot)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a constant $c(\psi)$ that depends on ψ, such that for any positive κ, for any $0 < t \leq |\mathsf{view}|$,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : t \text{ is a pivot in } \mathsf{view}\right] = c(\psi)$$

Proof. Let $t_c = 2\Delta$, since $np\Delta < 1$, $2\beta < np$, and $\Delta \geq 1$, we have that $(1-\beta)^{t_c} \geq 0.25$. Further, since $(1-2\alpha(\Delta+1))\alpha t > (1+\psi)\beta$, we have that $2\alpha\Delta < 1$, and therefore $(1-\alpha)^{t_c} \geq 0.25$.

$$
\begin{aligned}
&\Pr\left[t \text{ is a pivot}\right] \\
\geq &\Pr\left[\begin{array}{l} t \text{ is a backward pivot and } \mathbf{A}[t:t+t_c] = 0 \\ \text{and no alert node is leader in } [t:t+t_c] \text{ and } t \text{ is a forward pivot} \end{array}\right] \\
\geq &\Pr\left[t \text{ is a backward pivot}\right] \cdot \Pr\left[\mathbf{A}[t:t+t_c] = 0\right] \cdot \Pr\left[\text{no alert node is leader in } [t:t+t_c]\right] \\
&\cdot \Pr\left[t+t_c \text{ is a forward pivot } \mid \text{ no alert node is leader in } [t:t+t_c]\right] \\
\geq &\Pr\left[t \text{ is a backward pivot}\right] \cdot (1-\beta)^{t_c} \cdot (1-\alpha)^{t_c} \cdot \Pr\left[t+t_c \text{ is a forward pivot}\right] \\
= &c'(\psi)
\end{aligned}
$$

□

Given a view, we say that $\mathsf{many\text{-}pivots}^w(\mathsf{view}) = 1$ iff for any $0 \leq s < r \leq |\mathsf{view}|$ such that $r - s > w$, there must exist a pivot during the window (s, r).

Theorem 5 (There are many pivot points)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function negl such that for any κ, the following holds where $w = \frac{2\kappa}{\beta}$:

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{many\text{-}pivots}^w(\mathsf{view}) = 1\right] < \mathsf{negl}(\kappa)$$

We now prove the above theorem.

Let $s_1 := s + 1$, and for $i = 2$ to $\lfloor \frac{(r-s)\beta - \kappa}{3\sqrt{\kappa}} \rfloor$, let $s_i = s_{i-1} + \frac{3\sqrt{\kappa}}{\beta}$. Our strategy is to check for each $i = 1, 2, \ldots, \lfloor \frac{(r-s)\beta - \kappa}{3\sqrt{\kappa}} \rfloor$, whether s_i is a pivot. Let $\mathbf{G}_i$ denote the event that s_i is a pivot.

A view is said to be bad if there exists $t_0 \leq t_1 \leq |\mathsf{view}|$ where $t_1 - t_0 \geq \frac{\sqrt{\kappa}}{\beta}$, such that

$$\mathbf{A}(\mathsf{view})[t_0 : t_1] \geq \mathbf{C}(\mathsf{view})[t_0 : t_1]$$

Due to Lemma 3, it is not hard to see that only $\mathsf{negl}(\kappa)$ fraction of views are bad.

Fact 5. Conditioned on views that are not considered bad by the above definition for windows of length at least $\frac{\sqrt{\kappa}}{\beta}$, every $\mathbf{G}_i$ is independent of $\{\mathbf{G}_j\}_{j \neq i}$.

Proof. For views where the above bad events do not happen, s_i is a pivot iff s_i is a pivot w.r.t. to the window $(s_i - \frac{\sqrt{\kappa}}{\beta}, s_i + \frac{\sqrt{\kappa}}{\beta})$. Further, since $np\Delta < 1$ and $\beta < \frac{np}{2}$, we have that for every positive κ, $\frac{\sqrt{\kappa}}{\beta} > 2\Delta$. Therefore, it is easy to see that every $\mathbf{G}_i$ is independent of $\{\mathbf{G}_j\}_{j \neq i}$. □

We now return to the proof of Theorem 5. Let $\ell := \lfloor \frac{(r-s)\beta - \kappa}{3\sqrt{\kappa}} \rfloor$. Now, conditioned on good views where the aforementioned bad events do not happen, we have the following due to independence:

$$\Pr[\overline{\mathbf{G}}_1, \ldots, \overline{\mathbf{G}}_\ell] = \Pr[\overline{\mathbf{G}}_1] \Pr[\overline{\mathbf{G}}_2] \ldots \Pr[\overline{\mathbf{G}}_\ell] = \frac{1}{(1 - c(\psi))^\ell}$$

If $r - s > \frac{2\kappa}{\beta}$, we have that $(r-s)\beta - \kappa > \kappa$ and $\ell \geq \frac{\sqrt{\kappa}}{3}$, therefore it holds that there exists some negligible function negl such that

$$\Pr[\overline{\mathbf{G}}_1, \ldots, \overline{\mathbf{G}}_\ell] \leq \mathsf{negl}(\kappa)$$

Proof of consistency. Ignore the negligible fraction of views where the bad events we care about happen. By Lemma 2 and the above theorem, let $s = r - \frac{3\kappa}{\beta}$, then chain_i^r and $\mathsf{chain}_j^{r'}$ must diverge after s. Let $\mathsf{chain}_i^r\langle < s\rangle$ and $\mathsf{chain}_j^{r'}\langle < s\rangle$ denote the prefix of the chains with timestamps less than s. It holds that $\mathsf{chain}_i^r\langle < s\rangle = \mathsf{chain}_j^{r'}\langle < s\rangle$. Finally, due to Fact 3, at most $(1+\epsilon)np(r-s) = \frac{3(1+\epsilon)np\kappa}{\beta} < 6(1+\epsilon)\kappa$ blocks in chain_i^r can have a timestamp between s and r.

6.6 Chain Growth Upper Bound

We say that a chain is *first accepted* by honest nodes at time t in view iff 1) at any time $t' < t$, no alert node ever outputs some chain' to $\mathcal{Z}$ such that $\mathsf{chain} \prec \mathsf{chain}'$; and 2) at time t, some alert node outputs chain' to $\mathcal{Z}$ where $\mathsf{chain} \prec \mathsf{chain}'$.

No long block withholding. Let $\mathsf{withhold\text{-}time}(\mathsf{view})$ be the longest number of time steps t such that in view: 1) at some time in view, the adversary mines a chain with purported $\mathcal{F}_{\text{tree}}$-timestamp r; and 2) chain is first accepted by honest nodes at time $r+t$ in view.

Lemma 6 (No long block withholding)**.** Assume that $\gamma > (1+\epsilon_0)\beta$ for some constant $\epsilon_0 \in (0,1)$. Then for every Π_{ideal}-compliant p.p.t. $(\mathcal{A},\mathcal{Z})$ pair, for every constant $0 < \epsilon < 1$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A},\mathcal{Z},\kappa) : \mathsf{withhold\text{-}time}(\mathsf{view}) > \epsilon t\right] \leq \mathsf{negl}(\beta t)\cdot \mathsf{poly}(\kappa)$$

Proof. Given our new definition of $\mathsf{withhold\text{-}time}$, we can prove exactly the same "no long block withholding" lemma as Pass et al. [36], using our new definition of $A_t(\mathsf{view})$ in Section 6.3. We omit the full proof since the proof is almost identical to that of Pass et al. [36]. □

Chain growth upper bound proof. The proof is identical to that of Pass et al. — we only need to plug in our new $Q_t(\mathsf{view})$ definition (see Section 6.3) and our new "no long block withholding" lemma.

7 Proofs: Real World Emulates the Ideal World

We now show that the real-world protocol Π_{sleepy} securely emulates the ideal-world protocol Π_{ideal}. This can be shown using a standard simulation paradigm as described below. We construct the following simulator $\mathcal{S}$.

- $\mathcal{S}$ internally simulates $\mathcal{F}_{\text{CA}}$. At the start of execution, $\mathcal{S}$ honestly generates a $(\mathsf{pk}_i, \mathsf{sk}_i)$ pair for each honest node i, and registers pk_i on behalf of honest node i with the internally simulated $\mathcal{F}_{\text{CA}}$.

  Whenever $\mathcal{A}$ wishes to interact with $\mathcal{F}_{\text{CA}}$, $\mathcal{S}$ simply forwards messages in between $\mathcal{A}$ and the internally simulated $\mathcal{F}_{\text{CA}}$.

- Whenever $\mathcal{S}$ receives a hash query of the form $\mathsf{H}(\mathcal{P}, t)$ from $\mathcal{A}$ or from internally, $\mathcal{S}$ checks if the query has been asked before. If so, simply return the same answer as before.

  If not, $\mathcal{S}$ checks if $\mathcal{P}$ is a party identifier corresponding to this protocol instance. If not, $\mathcal{S}$ generates a random number of appropriate length and returns it. Else if the mapping succeeds,

$\mathcal{S}$ queries $b \leftarrow \mathcal{F}_{\text{tree}}.\texttt{leader}(\mathcal{P}, t)$. If $b = 1$, $\mathcal{S}$ rejection samples a random string h of appropriate length, until $h < D_p$; it then returns h. Else if $b = 0$, $\mathcal{S}$ rejection samples a random string h of appropriate length, until $h \geq D_p$; it then returns h.

- $\mathcal{S}$ keeps track of the "real-world" chain for every honest node i. Whenever it sends *chain* to $\mathcal{A}$ on behalf of i, it updates this state for node i. Whenever $\mathcal{A}$ sends *chain* to honest node i, $\mathcal{S}$ checks the simulation validity (see Definition 4) of *chain*. If *chain* is simulation valid and moreover *chain* is longer than the current real-world chain for node i, $\mathcal{S}$ also saves *chain* as the new real-world chain for node i.

- Whenever an honest node with the party identifier $\mathcal{P}$ sends chain to $\mathcal{S}$, $\mathcal{S}$ looks up the current real-world state *chain* for node $\mathcal{P}$. The simulator now computes a new chain using the real-world algorithm: let $(\mathsf{pk}, \mathsf{sk})$ be the key pair for node $\mathcal{P}$, let t be the current time, and let $\mathsf{B} := \mathsf{chain}[-1]$.

  If $\mathsf{eligible}^t(\mathcal{P})$ where the hash function H is through internal query to the simulator itself:

  let $\sigma := \Sigma.\texttt{sign}(\mathsf{sk}, \mathit{chain}[-1].h, \mathsf{B}, t)$, $h' := \mathsf{d}(\mathit{chain}[-1].h, \mathsf{B}, t, \mathcal{P}, \sigma)$,
  let $B := (\mathit{chain}[-1].h, \mathsf{B}, t, \mathcal{P}, \sigma, h')$, let $\mathit{chain}' := \mathit{chain}||B$.

  Now, the simulator $\mathcal{S}$ sends chain' to $\mathcal{A}$.

- Whenever $\mathcal{A}$ sends a *chain* to an honest node i, $\mathcal{S}$ intercepts the message. $\mathcal{S}$ ignores the message if *chain* is not simulation valid. Otherwise, let $\mathsf{chain} := \mathsf{extract}(\mathit{chain})$, and let $\mathsf{chain}[:\ell] \prec \mathsf{chain}$ be the longest prefix such that $\mathcal{F}_{\text{tree}}.\texttt{verify}(\mathsf{chain}[:\ell]) = 1$. The simulator checks to see if there exists a block in $\mathit{chain}[\ell+1:]$ signed by an honest $\mathcal{P}$. If so, abort outputting $\mathsf{sig\text{-}failure}$. Else, for each $k \in [\ell+1, |\mathsf{chain}|]$,

  1. let $\mathcal{P}^* := \mathit{chain}[k].\mathcal{P}$, let $t^* := \mathit{chain}[k].\mathsf{time}$.
  2. $\mathcal{S}$ then calls $\mathcal{F}_{\text{tree}}.\texttt{extend}(\mathsf{chain}[:k-1], \mathsf{chain}[k], t^*)$ on behalf of corrupt party $\mathcal{P}^*$.

  Notice that if the current *chain* is simulation valid, then the new chain' must be simulation valid as well. Finally, $\mathcal{S}$ forwards chain to honest node i.

- At any point of time, if $\mathcal{S}$ observes two different simulation valid (real-world) chains that contain identical (real-world) blocks, abort outputting $\mathsf{duplicate\text{-}block\text{-}failure}$.

Definition 4 (Simulation valid chains)**.** We say that a *chain* is simulation valid if it passes the real-world validity checks, but using the H and the $\mathcal{F}_{\text{CA}}$ implemented by the simulator $\mathcal{S}$.

Fact 6. The simulated execution never aborts with $\mathsf{duplicate\text{-}block\text{-}failure}$ except with negligible probability.

Proof. For this bad event to happen, it must be the case that two distinct queries to the hash function d returns the same result. Since there can be only polynomially many such queries, this happens with negligible probability. □

Fact 7. The simulated execution never aborts with $\mathsf{sig\text{-}failure}$ except with negligible probability.

Proof. We ignore all views where the bad event duplicate-block-failure happens.

Suppose some block B is signed by the simulator $\mathcal{S}$. Then, some honest node i must have sent $\mathsf{chain}||\mathsf{extract}(B)$ to $\mathcal{S}$ earlier, and this means that chain must be in $\mathcal{F}_{\text{tree}}$. Therefore, if sig-failure ever happens, it means that the adversary $\mathcal{A}$ has produced a signature on a different message that $\mathcal{S}$ never signed (due to no duplicate-block-failure). We can now easily construct a reduction that breaks signature security if sig-failure happens with non-negligible probability. □

Lemma 7 (Indistinguishability)**.** Conditioned on the fact that all of the aforementioned bad events do not happen, then the simulated execution is identically distributed as the real-world execution from the perspective of $\mathcal{Z}$.

Proof. Observe that the simulator's H coins are always consistent with $\mathcal{F}_{\text{tree}}$'s `leader` coins. Further, as long as there is no sig-failure, if the simulator receives any simulation valid *chain* from $\mathcal{A}$, either $\mathsf{chain} := \mathsf{extract}(\mathsf{chain})$ already exists in $\mathcal{F}_{\text{tree}}$, or else $\mathcal{S}$ must succeed in adding chain to $\mathcal{F}_{\text{tree}}$.

The rest of the proof works through a standard repartitioning argument. □

Fact 8. If $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{sleepy}}(p)$-compliant, then $(\mathcal{S}^{\mathcal{A}}, \mathcal{Z})$ is $\Pi_{\text{ideal}}(p)$-compliant.

Proof. $\Pi_{\text{sleepy}}(p)$ and $\Pi_{\text{ideal}}(p)$ have identical compliance rules. The only rule to verify is the Δ-compliance and the requirement for forwarding all pending messages when a sleepy node wakes up — every other rule is straightforward to verify. Observe that whenever an honest node sends $\mathcal{S}$ an ideal-world chain, $\mathcal{S}$ will transform it to a real-world *chain* and forward it to $\mathcal{A}$. Since $(\mathcal{A}, \mathcal{Z})$ is compliant, for each alert node j, within Δ steps $\mathcal{A}$ will ask $\mathcal{S}$ to forward *chain* to j. Similarly, for any sleepy node j that wakes up after Δ time, at the time it wakes up, $\mathcal{A}$ will ask $\mathcal{S}$ to forward *chain* to j. Note that $\mathcal{S}$ will never drop such a request since all *chain* sent from $\mathcal{S}$ to $\mathcal{A}$ are simulation valid. Therefore $\mathcal{S}$ respects the Δ-delay rule as well, and further $\mathcal{S}$ respects the rule to forward waking nodes all pending messages. □

Finally, since the simulated execution is compliant, it respects all the desired properties as Theorem 4 states. Now, since real-world execution and the simulated execution are indistiguishable, it holds that all the desired properties hold in the same way for the real-world execution. We therefore complete the proof of our main theorem, that is, Theorem 2 of Section 4.

Acknowledgments

We thank Rachit Agarwal, Kai-Min Chung, Naomi Ephraim, Ittay Eyal, and Andrew Morgan for helpful and supportive discussions. This work is supported in part by NSF grants CNS-1217821, CNS-1314857, CNS-1514261, CNS-1544613, CNS-1561209, CNS-1601879, CNS-1617676, AFOSR Award FA9550-15-1-0262, an Office of Naval Research Young Investigator Program Award, a Microsoft Faculty Fellowship, a Packard Fellowship, a Sloan Fellowship, Google Faculty Research Awards, and a VMWare Research Award.

References

[1] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In *CRYPTO*, pages 361–377, 2005.

[2] User "BCNext". NXT. http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt, 2014.

[3] Michael Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 27–30, New York, NY, USA, 1983. ACM.

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

[5] 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.

[6] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 355–362, 2014.

[7] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, October 1985.

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

[9] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 524–541, 2001.

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

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

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

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

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

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

[18] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *SIAM Journal of Computing*, 1997.

[19] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[20] Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Trans. Dependable Secur. Comput.*, 2(1):46–56, January 2005.

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

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

[23] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. *IACR Cryptology ePrint Archive*, 2015:1019, 2015.

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

[25] Sunny King and Scott Nadal. PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, August 2012.

[26] Jae Kwon. Tendermint: Consensus without mining. `http://tendermint.com/docs/tendermint.pdf`, 2014.

[27] Leslie Lamport. The weak byzantine generals problem. *J. ACM*, 30(3):668–676, 1983.

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

[29] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC '09, pages 312–313, 2009.

[30] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[31] Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3), 2006.

[32] Gregory Maxwell and Andrew Poelstra. Distributed consensus from proof of stake is impossible, 2014. `https://download.wpsoftware.net/bitcoin/pos.pdf`.

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

[34] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. Cryptology ePrint Archive, Report 2016/199, 2016. `http://eprint.iacr.org/`.

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

[36] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of blockchain protocol in asynchronous networks. https://eprint.iacr.org/2016/454.

[37] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. Manuscript, 2016.

[38] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. Manuscript, 2016.

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

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

[41] User "tacotime". Netcoin proof-of-work and proof-of-stake hybrid design, 2013. https://web.archive.org/web/20131213085759/http://www.netcoin.io/wiki/Netcoin_Proof-of-Work_and_Proof-of-Stake_Hybrid_Design.