Snow White: Provably Secure Proofs of Stake

Iddo Bentov Rafael Pass Elaine Shi

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

Abstract

Decentralized cryptocurrencies have pushed deployments of distributed consensus to more stringent environments than ever before. Most existing protocols rely on proofs-of-work which require expensive computational puzzles to enforce, imprecisely speaking, "one vote per unit of computation". The enormous amount of energy wasted by these protocols has been a topic of central debate, and well-known cryptocurrencies have announced it a top priority to alternative paradigms. Among the proposed alternative solutions, proofs-of-stake protocols have been of particular interest, where roughly speaking, the idea is to enforce "one vote per unit of stake". Although the community have rushed to propose numerous candidates for proofs-of-stake, no existing protocol has offered formal proofs of security, which we believe to be a critical, indispensible ingredient of a distributed consensus protocol, particularly one that is to underly a high-value cryptocurrency system.

In this work, we seek to address the following basic questions:

- What kind of functionalities and robustness requirements should a consensus candidate offer to be suitable in a proof-of-stake application?
- Can we design a *provably* secure protocol that satisfies these requirements?

To the best of our knowledge, we are the first to formally articulate a set of requirements for consensus candidates for proofs-of-stake. We argue that any consensus protocol satisfying these properties can be used for proofs-of-stake, *as long as money does not switch hands too quickly.* Moreover, we provide the first consensus candidate that *provably* satisfies the desired robustness properties.

1 Introduction

Consensus protocols are at the core of distributed systems — an important and exciting area that has thrived for the past 30 years. Traditionally, the deployment of consensus protocols has been largely restricted to relatively controlled environments, e.g., imagine a (hypothetical) deployment within Google to replicate a mission-critical service 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. Further, the set of consensus nodes is typically quite stable and reconfigurations are infrequent.

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

*http://www.initc3.org/

amazingly, it is possible to achieve robust consensus in a decentralized environment that is much more "hostile" than the traditional comfort zones for consensus deployment. Partly for this reason, Bitcoin is often referred to as the "honeybadger of money".

Most existing cryptocurrencies [44, 52] adopt proof-of-work blockchains, originally proposed by Nakamoto [44]. The Nakamoto blockchain works in a *permissionless* model, where any node can freely join and leave the protocol, and there is no a-priori knowledge of the set of consensus nodes. Until recently, the permissionless setting somewhat escaped the academic community's attention, partly because we understand that strong impossibility results exist in the permissionless model [4]. Nakamoto's ingenious blockchain protocol circumvented this lower bound by introducing proofs-of-work which, imprecisely speaking, seeks to enforce "one vote per unit of computation". Notably several recent works have formally shown the security of the Nakamoto blockchain [26, 45],

Although many would consider blockchain protocols a breakthrough for distributed consensus, today's blockchain protocols are plagued by several well-known painpoints among which performance is perhaps the most visible and most often debated. In particular, today's blockchains adopt expensive proofs-of-work, resulting in enormous wastes of energy [18, 27, 49] — it is estimated that Bitcoin mining today consumes more than 1000 MW of power [1], comparable to the power generated by a nuclear power plant such as Vogtle 3 (Georgia, USA, 1100 MW).

The search for a paradigm shift. The cryptocurrency community has been soul-searching for a paradigm shift. There is a growing interest in alternative consensus candidates that do not rely on expensive proofs-of-work, and yet are robust enough to withstand a wide-area deployment. Of particular interest is a new class of protocols called "proofs-of-stake" where, at 30,000 feet, the idea is to enforce "one vote per unit of stake in the system", where stake can be measured by the amount of currency units owned by a specific node. Since the initial suggestion of this idea in online Bitcoin forums [48], the community rushed to propose a series of protocol candidates [7, 8, 12, 19, 33, 35, 41, 48, 51], and major cryptocurrencies such as Ethereum have declared it a pressing priority to switch to a proof-of-stake paradigm [2, 12].

Despite the abundant and ever-growing excitement, unfortunately, the community continues to lack a clear and precise articulation of the security requirements of a proof-of-stake consensus protocol — for this reason, perhaps unsurprisingly, known protocol candidates [7, 8, 12, 19, 33, 35, 41, 48, 51] all appear to be lacking in one crucial ingredient, that is, formal modeling and proofs of security.

1.1 Our Contributions

In this paper, we embark on this quest: to search for an "ideal" consensus protocol candidate for proof-of-stake, that is *robust* enough to survive the adversarial nature of decentralized deployments, and more importantly, one that is *provably secure*.

It is a truth universally acknowledged by the cryptocurrency community, that consensus protocols for decentralized cryptocurrencies must be "highly robust". Many believe that classical consensus candidates are unsuitable, and that "blockchains are more robust than classical consensus". For quite a while, we were not able to formally justify such intuitions, or articulate exactly what robustness properties we look for. In our prior Sleepy work (Bentov, Pass, and Shi [9]), we make a first step forward at articulating new robustness properties that are desirable in emerging decentralized applications. Specifically, we show how to apply core ideas behind modern blockchains to construct consensus protocols in the classical sense, and meanwhile avoid expensive proofs-of-work.

The resulting protocol, called Sleepy [9], achieves new robustness properties that are not attained by any classical consensus protocol. To precisely articulate such new robustness properties, our earlier work [9] demonstrates that traditional network models considered by the distributed systems literature (including synchronous, asynchronous, and partially synchronous models) are too coarse-grained to express even very simple and natural robustness properties that one might care about. Therefore, we proposed a new formal model for consensus referred to as the *sleepy model* [9] — which roughly speaking, allows them to simultaneously achieve security in the presence of up to 50% corruption (among online nodes), and meanwhile be able to capture a broad class of failures and network jitters that are unavoidable in a wide-area setting. Note that this was not possible with classical models — since to capture failures and network jitters one would have to adopt the partially synchronous or asynchronous models; and yet a well-known lower bound by Dwork et al. [21] suggests that in these models, it is impossible to construct a consensus protocol that tolerates a $\frac{1}{3}$ corruption!

While Sleepy may seem like a good starting point, we are still quite far from constructing a secure proof-of-stake candidate. In this paper, we identify new functionalities and new robustness properties that are needed for proof-of-stake, and we show that, to satisfy these new functionality and robustness requirements would require non-trivial challenges both in terms of modeling and protocol construction. In the remainder of the paper, we make the following contributions.

- First, we articulate, in an intuitive language, a set of robustness properties are desirable for consensus protocols in the decentralized setting. Particularly, we use proof-of-stake as a driving application to identify these requirements.

- Second, we make an endeavor to formalize the set of intuitive requirements we have identified. It turns out that translating the set of intuitive requirements into a formal language is surprisingly challenging. Echoing findings of our recent Sleepy work [9], here we show that, to an even greater extent, traditional modeling techniques in classical distributed systems are too coarse-grained — as a result they are insufficient for articulating new robustness properties that are desired in emerging decentralized applications, and for articulating the robustness properties of a new class of protocols.

- Finally, we construct a provably secure consensus candidate called Snow White that achieves our desired functional and security requirements. More specifically, Snow White relies on the Sleepy consensus protocol as a starting point. We then describe a sequence of non-trivial enhancements that make the protocol progressively more versatile and more robust.

  In this process, we revisit our modeling choices several times, allowing us to achieve the following: 1) we make sure that our modeling choices are almost tight, in the sense that any additional trust assumption we introduce is necessary — since without them we would be treading on theoretical infeasibility; 2) we carefully articulate adversarial capabilities, such that we can precisely articulate the capabilities of our protocol, and prove it secure under the most powerful adversary we can defend against.

We point out that while we use proof-of-stake as a motivating application, our modeling techniques and construction may be of independent interest. Specifically, our protocol is generally applicable as a candidate for robust consensus in decentralized networks, where outages and network jitters are unavoidable, and node churns are frequent.

1.2 Desiderata and Challenges

In this paper we focus on *permissioned* consensus protocols that do not require proofs-of-work, and where there is an a-priori known initial committee of nodes responsible for consensus. In a decentralized setting, previous work has formally demonstrated that we can bootstrap common knowledge of this initial committee from a proof-of-work blockchain [47]. Further, in the proof-of-stake setting, it is often suggested that this initial committee be bootstrapped from an existing cryptocurrency system such as Bitcoin. Specifically, we can take the set of public keys associated with monetary value as the initial committee.

Robustness in a sleepy network. Despite common knowledge of an initial committee, not all nodes in the committee may be online at all times. For example, only a subset of users that own Bitcoins are miners, many remaining users only run light-weight clients that do not perform mining, or simply keep their coins in cloud storage [3, 18]. Further, in a wide-area network, node outages and network jitters are inevitable. In general, we shall classify all offline nodes and nodes with temporary bad network connections as "sleepy" nodes.

Obviously, the existence of sleepy nodes is unavoidable, and we cannot hope for sleepy nodes to participate in the consensus. A natural desideratum is that sleepy nodes should not hamper the progress of the consensus. Additionally, since a decentralized network contains mutually distrustful nodes, a higher level of resilience is typically desired. Intuitively, the following desideratum is natural to desire:

Desideratum 1 (sleepy consensus):

We would like a protocol that tolerates sleepy nodes, and specifically, achieves consistency and liveness when the *majority* of *awake* nodes are behaving honestly.

Surprisingly, our own recent work (Bentov, Pass, and Shi [9]) argues that the aforementioned robustness property is not attained by any existing classical permissioned consensus protocol. We then define a new formal model called the sleepy model [9], and show that one can apply ideas behind proof-of-work blockchains to a non-proof-of-work setting, and design new protocols that achieve the goal of "sleepy consensus". We believe that such robustness against sleepy nodes is important for a decentralized proof-of-stake application. It would now appear that the Sleepy protocol [9] is an attractive starting point for constructing proof-of-stake protocols.

Robust committee reconfigurations. Although the Sleepy protocol [9] achieves new robustness properties that are not attained by any classical consensus protocol, we soon realize that even Sleepy may not be robust enough. Below we describe two additional challenges that a proof-of-stake protocol must address.

Recall that in a proof-of-stake protocol, we would like to give nodes voting power proportional to their stake in the cryptocurrency. Therefore, committees are determined by the amount of currency units each public key carries.

However, invariably money can switch hands over time. As a result, the consensus committee should evolve to include new users that have stake in the system, and prune out old committee members who no longer hold stake. Since membership churns are a frequent operation, they must be handled in lightweight manners ideally without having to invoke special execution paths in the consensus program. We therefore propose the second intuitive desideratum.

Desideratum 2 (committee reconfigurations):

The protocol should support frequent committee reconfigurations in a light-weight manner.

Robustness against posterior corruptions. The fact money can switch hands over time raises an additional challenge. In a decentralized network, nodes typically do not have long-term mutual trust. When the majority of the money in the system has switched hands, old committee members (possibly more than majority) who no longer hold stake may now be incentivized to attack the system, e.g., by using their voting power in the past to overwrite history and thus break consistency. Therefore, we now have the following natural desideratum:

Desideratum 3 (tolerate posterior corruption):

Corrupt members of sufficiently old committees should not be able to overwrite history.

Articulating the posterior corruption requirement formally turns out to be very subtle and challenging! Following the terminology adopted by Sleepy, henceforth we refer to nodes that are honest and awake as *alert* nodes. Ideally, what one would have liked is that the scheme retains security as long as at any time, there are more alert committee members (of the present committee) than corrupt ones, or more formally, at any t, there is a positive constant ϕ such that

$$\frac{\mathsf{alert}^t(\mathsf{cmt}^t)}{\mathsf{corrupt}^t(\mathsf{cmt}^t)} > 1 + \phi \tag{1}$$

where $\mathsf{alert}^t(\mathsf{cmt}^t)$ and $\mathsf{corrupt}^t(\mathsf{cmt}^t)$ output those that are alert and corrupt at time t respectively among the committee at time t. However, it turns out that this notion is unattainable. Specifically, for a node i that newly joins or one that wakes up from sleep, a committee in the past (say, at past time t^*) might already have become majority corrupt, and the coalition may now fork off an alternate history starting from around t^*. At this moment, node i has no means of discerning the real and the alternate histories. It turns out that we can formalize this intuition and prove a lower bound that indeed, absent any additional trust assumptions, such a notion of security would have been unattainable (Section 6).

Our approach is the following.

1. First, we introduce the minimal assumptions necessary to circumvent the theoretical impossibilities. Specifically, we assume that newly joining nodes and nodes that wake up from a long sleep has a way to determine which is the correct history to believe in. For example, this can be achieved if the (re)joining node can contact a list of nodes the majority of whom are alert.

2. Second, we quantify the condition specified in Equation (1) more carefully. Let W denote an appropriate posterior corruption window parameter. Roughly speaking, we require that the scheme retains security as long as for any t,

$$\frac{\mathsf{alert}^t(\mathsf{cmt}^t) \cap \mathsf{honest}^{t+W}(\mathsf{cmt}^t)}{\mathsf{corrupt}^{t+W}(\mathsf{cmt}^t)} > 1 + \phi \tag{2}$$

   More intuitively, the parameter W qualifies the window over which a committee member has influence. If a node serves as committee member at time t and never serves on any committee in the future, then its influence is limited to the window $[t, t+W]$. After $t+W$, even if the

majority of the committee at t become corrupt, they cannot overwrite the past — as long as (imprecisely speaking) the corrupt coalition is not also serving on future committees.

Looking forward, the introduction of the parameter W allows our proof-of-stake system to be secure even when the majority of stake in old committees become corrupt later in time. Since the committee rotation can lag behind the real-time stake distribution, we would additionally require that money in the system does not switch hands too quickly, to ensure that honest nodes are roughly aligned with honest stake — in this way, we can translate the assumptions expected by our consensus protocol into assumptions on the distribution of stake.

3. Third, we would have to carefully adjust our formal model to distinguish between two types of sleepers, *light sleepers* who sleep for a short duration bounded by $\widetilde{\Delta}$, and *deep sleepers* who may sleep indefinitely before waking up. Light sleepers receive all pending messages upon waking (e.g., network jitters), whereas deep sleepers are treated essentially the same way as respawning (e.g., longer node outages). Such a distinction is necessary, since it is not hard to see that if $W < \widetilde{\Delta}$, agreement would have been impossible for nodes that wake up from light sleep — since if the committee $\widetilde{\Delta}$ steps ago are already corrupt when a node wakes up, the corrupt coalition can fork off an alternate history starting from roughly $\widetilde{\Delta}$ ago and the waking node would have no means to discern the real and alternate histories.

1.3 Snow White: Consensus Candidate for Proofs-of-Stake

Finally, we show that when we carefully articulate our model and assumptions to navigate around theoretical impossibilities while imposing the least amount of constraints, we can indeed construct a robust candidate consensus protocol, called Snow White, that satisfies all of these requirements.

Further, we argue that a consensus candidate that satisfies these constraints is suitable in a proof-of-stake setting, *as long as money in the system does not change hands too quickly.* Under such assumptions, we can translate the assumptions expected by Snow White for security to assumptions on the distribution of stake. Our Snow White protocol offers a committee reconfiguration opportunity regularly per epoch. A desirable approach is to allow nodes to have proportional number of seats in each new committee relative to their current stake in the system. However, since money in the system can continuously switch hands, it is possible that elected committee members can sell their money during the time they are serving on the committee at which point they no longer have stake, and might now be incentivized to attack the system or perform a double-spending attack. Fundamentally, this problem stems from the fact that our committee election lags slightly behind the real-time distribution of stake.

To thwart such an attack, we can have the cryptocurrency layer to limit the liquidity in the monetary system. For example, imagine that at any point of time, $a = 30\%$ of the stake is alert and will remain honest sufficiently long, $c = 20\%$ is corrupt, and the rest are sleepy. We can have the cryptocurrency layer enforce the following rule: only $\frac{a-c}{2} - \epsilon = 5\% - \epsilon$ of the stake can switch hands during every epoch (technically, this window needs to be longer than an epoch as mentioned in Section 7). In other words, if in any appropriately long window, only l fraction of money in the system can switch hands, it holds that as long as at any time, $2l + \epsilon$ more stake is alert and remain honest sufficiently long than the stake that is corrupt, we can guarantee that the conditions expected by the consensus protocol, that is, at any time, more committee members are alert and remain honest sufficiently long, than the committee members that are corrupt.

Independent work. An independent paper by Kiayias et al. [31], which was posted on eprint a week before ours, also claims to provide a provable proof of stake protocol. We posted our paper as soon as possible after becoming aware of this posting, and have not yet looked at the details of their paper. We will provide a more detailed comparison once we have read it.

2 Technical Roadmap

2.1 Background: Sleepy Consensus

As mentioned earlier, in the presence of (possibly many) sleepy nodes, it would be desirable for a protocol to be secure as long as the majority of *awake* nodes are honest.

Surprisingly, as our own prior work points out [9], no known classical consensus protocol can achieve this! Specifically, classical *synchronous* protocols would fail since the possibility of nodes going to sleep and later waking up violates synchrony assumption outright. While the *asynchronous* (or *partially synchronous*) models were designed exactly to capture adversarial environments with potentially unbounded message delays, all known asynchronous protocols crucially rely on more than $\frac{2}{3}$ fraction of nodes being honest to achieve security — indeed, it was well-understood that security against a $\frac{1}{3}$ coalition is impossible in the asynchronous setting [21]. As a result, known asynchronous protocols fail to tolerate up to 50% corruption (among awake nodes) — something that we have asked for. In particular, imagine if all nodes were awake, then the requirement we have phrased simply translates to security against $< 50\%$ corruption in the classical sense.

It would seem at this point that we are stuck and that we might have asked for a robustness property that is impossible to achieve. Perhaps surprisingly, our prior work [9] shows that the aforementioned goal of sleepy consensus is not inherently unattainable.

The sleepy model of consensus. Our earlier work defines a new formal model called the "sleepy" model [9]. In the sleepy model, honest nodes are either *alert* or *sleepy*. An alert node is one that is online and has predictable network connections to all other alert nodes. Messages delivered by an alert node is guaranteed to arrive at all other alert nodes within a maximum delay of Δ, where Δ is an input parameter to the protocol. A sleepy node captures any node that is either offline or suffering a slower than Δ network connection. A sleepy node can later wake up, and upon waking at time t, all pending messages sent by alert nodes before $t - \Delta$ will be immediately delivered to the waking node.

Therefore, on the surface, the sleepy model takes after both the traditional synchronous model (in that alert nodes have a network with a known delay parameter), and the traditional asynchronous model (in that nodes allowed to sleep and wake up later at an indefinite time, at which point it receives all pending messages). As it turns out, such a hybrid model allows us to avoid the well known $\frac{1}{3}$-lower bound for asynchronous networks, and meanwhile retain the ability to capture a wide variety of adverse network situations including network jitters and node outages!

The Sleepy protocol as a starting point. In our earlier work [9], we propose a new consensus protocol called Sleepy; and prove that Sleepy retains security as long as the majority of awake nodes are honest.

Intriguingly, the Sleepy protocol borrows core ideas behind proofs-of-work blockchains, and apply them to a classical setting without proofs-of-work. The resulting protocol achieves non-

trivial robustness guarantees that are not attained by any known classical consensus candidates — and such new robustness properties are particularly desirable in application scenarios like proofs-of-stake! As a result, we will use Sleepy as a starting point in constructing protocols for proofs-of-stake.

We now quickly review the Sleepy protocol. Sleepy relies on a random oracle (which can be replaced with a common reference string and a pseudo-random function if static corruption is assumed) to elect a leader in every time step. The elected leader is allowed to extend a blockchain with a new block, by signing a tuple that includes its own identity, the transactions to be confirm, the current time, and the previous block's hash. Like in the Nakamoto consensus, nodes always choose the longest chain if they receive multiple different ones. To make this protocol fully work, Sleepy [9] proposes new techniques to timestamp blocks to constrain the possible behaviors of an adversary. Specifically, there are two important blockchain timestamp rules:

1. a valid blockchain must have strictly increasing timestamps; and
2. honest nodes always reject a chain with timestamps in the future.

Note that all aforementioned timestamps can easily be adjusted to account for possible clock offsets among nodes by applying a generic protocol transformation [9].

Terminology. Henceforth in this paper, we adopt the same terminology as the Sleepy work [9], where honest nodes are classified into *alert* ones and *sleepy* ones. Alert nodes are honest and awake, and network delay is bounded by a known Δ between alert nodes. We assume that all corrupt nodes are awake.

2.2 Handling Committee Reconfiguration

In general, since nodes come and go in a decentralized network, the consensus protocol ought to reconfigure the committee periodically to assimilate new nodes that have joined and prune old ones who are no longer present. For example, in a proof-of-stake system, monetary units may switch hands, and the committee should evolve over time in sync with evolution of the stake-holders. Old committee members who no longer hold stake should also cease to have voting power. By contrast, newly joining nodes should be given seats in the committee proportional to the stake they have. Intuitively, this raises two requirements:

- We should allow consensus nodes to be spawned dynamically, after the start of protocol execution. For example, new users interested in a cryptocurrency may purchase money in the system and join the consensus protocol.
- The protocol should offer committee reconfiguration opportunities frequently and punctually to evolve the committee over time.

Unfortunately, the Sleepy protocol fails to handle these requirements. Specifically, the Sleepy protocol does not permit dynamic spawning of new consensus nodes and all committee members have to be spawned upfront and provided as a-priori common knowledge to all nodes — as we explain later, there will be an adaptive key selection attack if one wish to support dynamic spawning of consensus nodes.

We now consider how to allow dynamic node spawning and committee reconfiguration. To be broadly application to a wide range of applications, our Snow White protocol does not stipulate

how applications should select the committee over time. Roughly speaking, we wish to guarantee security as long as the application-specific committee selection algorithm respects the constraint that there is honest majority among all awake nodes — note that the precise condition needed for security will change later when we start handling (mildly) adaptive corruptions and posterior corruptions. Therefore, we assume that there is some application-specific function $\mathsf{extractpks}(\mathit{chain})$ that examines the state of the blockchain and outputs a new committee over time. In a proof-of-stake context, for example, this function can roughly speaking, output one public key for each currency unit owned by the user. In Section 7, we discuss in a proof-of-stake context, how one might possibly translate assumptions on the distribution of stake to the the formal requirements expected by the consensus protocol.

Strawman scheme: epoch-based committee selection. Snow White provides an epoch-based protocol for committee reconfiguration. Each T_{epoch} time, a new epoch starts, and the beginning of each epoch provides a committee reconfiguration opportunity. Let $\mathsf{start}(e)$ and $\mathsf{end}(e)$ denote the beginning and ending times of the e-th committee. Every block in a valid blockchain whose time stamp is between $[\mathsf{start}(e), \mathsf{end}(e))$ is associated with the e-th committee.

It is important that all honest nodes agree on what the committee is for each epoch. To achieve this, our idea is for honest nodes to determine the new committee by looking at a *stabilized* part of the chain. Therefore, a straightforward idea is to make the following modifications to the basic Sleepy consensus protocol:

- Let 2ω be a look-back parameter.
- At any time $t \in [\mathsf{start}(e), \mathsf{end}(e))$ that is in the e-th epoch, an alert node determines the e-th committee in the following manner: find the latest block its local *chain* whose timestamp is no greater than $\mathsf{start}(e) - 2\omega$, and suppose this block resides at index ℓ.
- Now, output $\mathsf{extractpks}(\mathit{chain}[:\ell])$ as the new committee.

In general, the look-back parameter 2ω must be sufficiently large such that all alert nodes have the same prefix $\mathit{chain}[:\ell]$ in their local chains by time $\mathsf{start}(e)$. On the other hand, from an application's perspective, 2ω should also be recent enough such that the committee composition does not lag significantly behind.

Preventing an adaptive key selection attack. Unfortunately, the above scheme is prone to an adaptive key selection attack where an adversary can break consistency with constant probability. Specifically, as the random oracle H is chosen prior to protocol start, the adversary can make arbitrary queries to H. Therefore, the adversary can spawn corrupt nodes and seed them with public keys that causes them to be elected leader at desirable points of time. For example, since the adversary can query H, it is able to infer exactly in which time steps honest nodes are elected leader. Now, the adversary can pick corrupt nodes' public keys, such that every time an honest node is leader, a corrupt node is leader too — and he can sustain this attack till he runs out of corrupt nodes. Since the adversary may control up to $\Theta(n)$ nodes, he can thus break consistency for $\Theta(n)$ number of blocks.

We note that the Sleepy consensus protocol avoids such an adaptive key selection attack by adopting a rather restricted model, where 1) all nodes have to be spawned upfront; 2) all `corrupt`

and `sleep` instructions have to be declared upfront; and 3) then the random oracle H is chosen, and the protocol execution starts. Specifically, Sleepy does not allow dynamic spawning of nodes.

We describe how to defend against such an adaptive key selection attack. For simplicity, we assume, for the time being, we shall still stick to a *static* corruption model, but in contrast with Sleepy, we will allow spawning of new nodes after protocol execution starts. Here *static* security means that the adversary must declare whether a node is corrupt and which times it is asleep immediately before the node is spawned.

Our idea is to have nodes determine the next epoch's committee first, and then select the next epoch's hash — in this way, the adversary will be unaware of next epoch's hash until well after the next committee is determined. More specifically, we can make the following changes to the Sleepy protocol:

- Let 2ω and ω be two look-back parameters, for determining the next committee and next hash respectively.
- At any time $t \in [\mathsf{start}(e), \mathsf{end}(e))$ that is in the e-th epoch, an alert node determines the e-th committee in the following manner: find the latest block its local *chain* whose timestamp is no greater than $\mathsf{start}(e) - 2\omega$, and suppose this block resides at index ℓ_0. Now, output $\mathsf{extractpks}(\mathit{chain}[:\ell_0])$ as the new committee.
- At any time $t \in [\mathsf{start}(e), \mathsf{end}(e))$ an alert node determines the e-th hash in the following manner: find the latest block its local *chain* whose timestamp is no greater than $\mathsf{start}(e) - \omega$, and suppose this block resides at index ℓ_1. Now, output $\mathsf{extractnonce}(\mathit{chain}[:\ell_1])$ as a nonce to seed the new hash.
- We augment the protocol such that alert nodes always embed a random seed in any block they mine, and $\mathsf{extractnonce}(\mathit{chain}[:\ell_1])$ can simply concatenate all seeds in the prefix of the chain (in practice, further optimizations are possible), and use the resulting string as a nonce to seed the random oracle H.

For security, we require that

1. The two look-back parameters 2ω and ω are both sufficiently long ago, such that all alert nodes will have agreement on $\mathit{chain}[:\ell_0]$ and $\mathit{chain}[:\ell_1]$ by the time $\mathsf{start}(e)$; and
2. The two look-back parameters 2ω and ω must be spaced out sufficiently in time, such that the adversary cannot predict $\mathsf{extractnonce}(\mathit{chain}[:\ell_1])$ until well after the next committee is determined.

Remark 1 (On the use of the random oracle)**.** Note that while the Sleepy work can rely on a pseudorandom function and a common reference string to remove the random oracle, here we cannot rely on the same approach, and we thus assume a random oracle. In the Sleepy work, assuming bad events such as signature forgery and hash collisions do not happen, then the adversary's best strategy depends only on whether in each time step each node is elected. In Snow White, however, once the adversary learns the secret key of the pseudorandom function, he can pick the nonces and the public keys of newly spawned nodes in a way that is dependent on the pseudorandom function's secret key.

Achieving security under adversarially biased hashes. It is not hard to see that the adversary can bias the nonce used to seed the hash, since the adversary can place arbitrary seeds in the blocks it contributes. In particular, suppose that the nonce is extracted from the prefix $chain[:\ell_1]$. Obviously, with at least constant probability, the adversary may control the ending block in this prefix. By querying H polynomially many times, the adversary can influence the seed in the last block $chain[\ell_1]$ of the prefix, until it finds one that it likes.

Indeed, if each nonce is used only to select the leader in a small number of time steps (say, $O(1)$ time steps), such adversarial bias would indeed have been detrimental — in particular, by enumerating polynomially many possibilities, the adversary can cause itself to be elected with probability almost 1 (assuming that the adversary controls the last block of the prefix).

However, we observe that as long as the same nonce is used sufficiently many times, the adversary cannot consistently cause corrupt nodes to be elected in many time steps. More specifically, suppose each nonce is used to elect at least $\Omega(\kappa)$ leaders, then except with $\mathsf{negl}(\kappa)$ probability, the adversary cannot increase its share by more than an ϵ fraction — for an arbitrarily small constant $\epsilon > 0$. Therefore, to prove our scheme secure, it is important that each epoch's length (henceforth denoted T_{epoch}) be sufficiently long, such that once a new nonce is determined, it is used to elect sufficiently many leaders.

Reasoning about security under adversarially biased hashes. Formalizing this above intuition is somewhat more involved. In particular, our proof (Section 9.1) relies on the following strategy:

- First, we prove security under an ideal protocol denoted Π_{ideal} (Section 8) where we pretend that an imaginary trusted entity selects the next epoch's hash after the next committee is determined. In this way, the adversary has no bias over the randomness chosen, and the adversary cannot query the hash either before the next committee is determined.

- Next, we consider another protocol Π_{bias} that captures the adversary's ability to bias the hash. We then argue that if a bad event 1) happens with small probability in Π_{ideal}, and 2) depends only on a constant number of hashes in Π_{bias}; then this bad event must happen with small probability in the in Π_{bias} as well, adjusting for the adversary's ability to bias the hashes.

- We now inspect all bad events we would like to bound regarding Π_{bias}, including failures related to chain growth, chain quality, and consistency. We argue that essentially, all these bad events that we care about can be over-approximated by the union of polynomially bad events that depend only on a small number of hashes in Π_{bias}. More specifically, we divide the execution of Π_{bias} into possibly overlapping, medium-sized windows such that we can localize dependence. We argue that 1) the bad events in each window depend only on the hash functions involved in each window; 2) each window spans a constant number of epochs; and 3) if nothing bad happens in each (possibly overlapping) medium-sized window, then nothing bad happens across all time.

  In particular, each window cannot be too small since otherwise the bad events we care about would depend on too many hashes (and thus lead to possibly exponential blowup in a union bound); and as mentioned earlier, from the application's perspective, it is desirable that each window not be too large.

2.3 Security against a Mildly Adaptive Adversary

So far, we have assumed a static corruption model, where all `corrupt` and `sleep` instructions have to be declared upfront at either protocol start or prior to node spawning — and this appears to be a rather constraining adversarial model. Can we defend against an adversary that can dynamically corrupt nodes after they are spawned?

Unfortunately, one soon realizes that this type of overall approach (based on leader-election) cannot defend against a fully adaptive adversary — since such an adversary can simply observe who is leader in each time step, and make that node sleep (or corrupt it) precisely in the time step it is leader. We stress that nonetheless, Sleepy [9] achieves new robustness properties that are not attained by any classical protocol; and Snow White achieves additional robustness properties on top of Sleepy [9].

On the bright side, we observe that it is possible to handle a much more powerful adversary without requiring static corruption. Specifically, we can tolerate an adversary who can dynamically corrupt nodes or make nodes sleep, but such `corrupt` or `sleep` instructions take a while to be effective. For example, in practice, it may take some time to infect a machine with malware. Such a "mildly adaptive" corruption model has been formally defined by Pass and Shi [47], where they call it the τ-agile corruption model, where τ denotes the delay parameter till `corrupt` or `sleep` instructions take effect. Intuitively, as long as τ is sufficiently large, it will be too late for an adversary to corrupt a node or make the node sleep upon seeing the next epoch's hash. By the time the `corrupt` or `sleep` instruction takes effect, it will already be well past the epoch.

It turns out, however, that articulating the protocol's security under such an agile corruption model turns out to be rather tricky, as we now explain.

A history rewriting attack. The need to support agile corruptions opens up the following possible scenario: a committee member may be alert at the time that the committee is serving; however, it can become corrupt later in time.

When an old committee member becomes corrupt, it can now use its signing key to sign arbitrary blocks in the past; and this leads to a history rewriting attack. Specifically, suppose a specific committee serves in the e-th epoch. At some point of time much later, the majority of the e-committee now become corrupt. At this moment, these corrupt nodes can collude and cause a divergent chain where all blocks fork off starting at roughly the e-th epoch. Specifically, this attack can succeed because the set of corrupt nodes can fully control the alternative history (as long as it respects possible application-specific validity rules). He can control the all following epoch's hash functions and adaptively select the keys for corrupt nodes in subsequent committees. At this moment, if a new, alert node gets spawned, it is unable to distinguish the real log from the revised one, and thus consistency is broken. We point out that such an attack not only applies to a newly joining node, but also existing nodes as well — if the adversary succeeds in constructing a longer chain for the alternate history. Interestingly, we point out that such an attack is not possible in a proof-of-work blockchain. Nodes in a proof-of-work blockchain do not hold any secrets such as signing keys, and therefore if a node that mines an honest block becomes corrupt later in time, it cannot overwrite history "for free" without expending additional computation.

Therefore, to be able to prove security, it appears that we will need to require that for any committee across all time, *the majority of nodes in that committee must remain honest forever*. Note that it is okay for the majority of old committee members to go to sleep later, as long as the majority of any committee does not get corrupt. To make the above intuition more precise, we can

phrase the following condition: there exists a constant $\phi > 0$, such that for any execution trace view and any $t < |\mathsf{view}|$,

$$\frac{\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})}{\mathsf{corrupt}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})} \geq 1 + \phi \tag{3}$$

In the above[1], let cmt^t denote the committee serving at time t, then $\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$ outputs those among cmt^t that are alert at time t; $\mathsf{honest}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$ those among cmt^t that remain honest forever; and $\mathsf{corrupt}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$ output those among cmt^t that ever become corrupt. In other words, at any point of time t, the alert nodes among the present committee that remain honest forever (but possibly can go to sleep) must outnumber the nodes that ever become corrupt in the present committee.

2.4 Handling Posterior Corruption

Majority posterior corruption. Although the condition specified in Equation (3) allows us to prove security under an agile adversary, we would like an even stronger robustness guarantee in proof-of-stake-like applications. In particular, when old committee members have sold their currency units, they no longer hold stake in the system, and may attempt to subvert the system by rewriting history. Informally speaking, we desire the following robustness guarantee:

> [Security in the presence of majority posterior corruption]: Even when the *majority* of old committee members become corrupt, as long as the corruption takes place sufficiently late in time, it will be too late for the corrupt coalition to go back in time and corrupt history.

The "punctual" world. Let us first focus on an imaginary world, where all nodes are spawned upfront prior to protocol start and although nodes are allowed to sleep, sleepy nodes never have to wake up and rejoin the protocol — henceforth we refer to this world as the "punctual" world. We show that such a history revision attack is easy to address in such a "punctual" world, without introducing any additional assumptions. To see this, let us reflect on what the consistency requirement means for a blockchain protocol. Consistency requires that at any point of time, if an alert node removes the trailing $O(\kappa)$ blocks from its chain, the prefix of the chain should have stabilized and should always persist into its own future.

This gives rise to the following idea: suppose that we can already prove the consistency property as long as when there is no majority posterior corruption. Now, to additionally handle majority posterior corruption, we can have alert nodes always reject any chain that diverges from its current longest chain at a point sufficiently far back in the past (say, at least W time steps ago). In this way, old committee members that have since become corrupt cannot convince alert nodes to revise history that is too far back — in other words, the confirmed transaction log stabilizes and becomes immutable after a while.

Care must be taken when parametrizing the posterior corruption window W: although a smaller W indicates stronger security, it is necessary for W to be sufficiently large to not hamper liveness. If W is too small, the adversary can send alert nodes an adversarial fork, causing alert nodes may reject honest chains prematurely. This will cause the protocol to get stuck — more specifically, chain growth lower bound will not hold if W is too small.

[1] Without risk of ambiguity, we abuse set notation to also denote the set cardinality.

Indeed, if we were in such a "punctual" world, with some additional work, we can prove that the protocol (with the above modification) would indeed be secure as long as the adversary respects the following condition [2] — below for simplicity we assume that nodes do agree on the committee at any time whereas in actual technical definition is more precise such that we do not have to even make this assumption. We require that there exists a constant $\phi > 0$ such that for every possible execution trace view, for every $t \leq |\mathsf{view}|$, let $r = \min(t + W, |\mathsf{view}|)$,

$$\frac{\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})}{\mathsf{corrupt}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})} \geq 1 + \phi \tag{4}$$

where $\mathsf{alert}^t(\mathsf{cmt}^s(\mathsf{view}), \mathsf{view})$, $\mathsf{honest}^t(\mathsf{cmt}^s(\mathsf{view}), \mathsf{view})$, and $\mathsf{corrupt}^t(\mathsf{cmt}^s(\mathsf{view}), \mathsf{view})$ output the number of nodes in the committee of time s that are alert (or honest, corrupt, resp.) at time t.

Note that the condition specified in Equation (4) is weaker than that in Equation (3) which implies a higher degree of robustness. Informally, it requires the following: at any time t, there must be more alert committee members that remain honest (but possibly can fall asleep) for W more times steps, than those that become corrupt within a W window; In other words, we no longer require the majority of any committee to remain honest (possibly asleep) forever; we now only require that the majority of any committee to remain honest for a duration of W — and it is okay even if all the committee become corrupt sufficiently late in time. Another way to think of the constraint in Equation (4) is the following: we would like to guarantee that if a node becomes corrupt after time $t + W$, it should no longer have voting power over any history that is prior to t. Therefore, each committee does not care if their members become corrupt as long as corruption happens sufficiently late in time.

Joining safely. Earlier, we have describe an approach that defends against posterior corruption in an "punctual" world — assuming that node joins late in time, and no node ever needs to wake up and rejoin the protocol. The remaining challenge is to design a secure bootstrapping mechanism for nodes that join or rejoin late.

Recall that we described a history revision attack earlier, where if the majority of an old committee become corrupt at a later point of time, they can simulate an alternate past, and convince a newly joining node believe in the alternate past. Therefore, it seems that the crux is the following question:

> How can a node joining the protocol correctly identify the true version of history?

Unfortunately, it turns out that this is impossible without additional trust — in fact, we can formalize the aforementioned attack and prove a lower bound (Section 6) which essentially shows that in the presence of majority posterior corruption, a newly joining node has no means of discerning a real history from a simulated one:

> [Lower bound for posterior corruption]: Absent any additional trust, it is impossible achieve consensus in the sleepy model of execution, if the majority of an old committee can become corrupt later in time.

[2] In fact, the protocol is secure even under more relaxed assumptions: where all nodes are spawned upfront prior to protocol start, and we allow sleepy nodes to wake up as long as they sleep only for a short amount of time (where short is a parameter that will be specified later).

We therefore ask the following question: what minimal, additional trust assumptions can we make such that we can defend against majority posterior corruption? Informally speaking, we show that all we need is a secure bootstrapping process for newly joining nodes as described below. We assume that a newly joining node is provided with a list of nodes L the majority of whom must be alert — if so, the new node can ask the list of nodes in L to vote on the current state of the system, and thus it will not be mislead to choose a "simulated" version of the history.

Rejoining safely. Besides newly joining nodes, a related issue is when sleepy nodes wake up, they must securely rejoin the system. In the sleepy network model [9], we assume that a sleepy node, upon waking, would receive all pending messages plus possibly adversarial inserted ones. With posterior corruption, however, if nodes sleep for too long and then wake up, it is effectively not much different than a newly spawning node — for example, in the extreme case, the node can sleep from the start and only wake up much later. As a result, absent additional trust assumptions, an adversary can confuse the waking node with a simulated history — by sending the waking node protocol messages from both the real and simulated executions. In fact, we can formalize this intuition and prove an impossibility result, which works almost identically as the previous impossibility for newly joining nodes (Section 6).

In light of this lower bound, we make the following changes to our model and protocol: we will distinguish between a *light sleeper* and a *deep sleeper*: a light sleeper is one that wakes up relatively quickly after falling asleep; whereas a deep sleeper may be asleep for a long time. Intuitively, a deeper sleeper captures a longer outage — possibly longer than the posterior corruption window. Therefore for a deep sleeper to rejoin the protocol, it must perform a secure initialization procedure in the same manner as a newly joining node; otherwise the adversary can convince the rejoining node to accept a simulated history instead of the real one. On the other hand, a light sleeper captures transient network jitters, typically much shorter than the posterior corruption window. We assume that a light sleeper receives all pending messages (including possibly adversarial ones) upon waking, and a light sleeper need not perform protocol reinitialization[3].

We point out that the ability to capture light sleepers is important: it fundamentally separates our model from traditional synchronous models. Traditional synchronous protocols that rely on strong synchrony to reach common knowledge would generally be insecure in our model, where nodes can sleep transiently and receive messages off-sync when they wake up.

2.5 Putting it Altogether

In summary, our protocol, roughly speaking, works as follows. A formal description of the full protocol is presented in Section 5.

1. First, following Sleepy, there is a random oracle H that determines if a member of the present committee is a leader in each time step. If a node is leader in a time step t, he can extend the blockchain with a block of the format $(h_{-1}, \mathsf{txs}, \mathsf{time}, \mathsf{nonce}, \mathsf{pk}, \sigma)$, where h_{-1} is the previous block's hash, txs is a set of transactions to be confirmed, nonce is a random seed that will be useful later, pk is the node's public key, and σ is a signature under pk on the entire contents of the block. A node can verify the validity of the block by checking that 1) $\mathsf{H}^{\mathsf{nonce}_e}(\mathsf{pk}, \mathsf{time}) < D_p$ where D_p is a difficulty parameter such that the hash outcome is smaller than D_p with probability

[3]When Snow White awakens from her sleep, she sends a message to the prince, "Good heavens, where am I?" "You are with me", the prince sends a message in reply, "I love you more than anything else in the world" [28].

p, and nonce_e is a nonce that is reselected every epoch (we will describe how the nonce is selected later); and 2) the signature σ verifies under pk.

2. Also following Sleepy, a valid blockchain's timestamps must respect two constraints: 1) all timestamps must strictly increase; and 2) any timestamp in the future will cause a chain to be rejected.

3. Next, to defend against old committees that have since become corrupt from rewriting history, whenever an alert node receives a valid chain that is longer than his own, he only accepts the incoming chain if the incoming chain does not modify blocks too far in the past, where "too far back" is defined by the parameter κ_0.

4. Next, a newly joining node or a node waking up from deep sleep must invoke a secure bootstrapping mechanism such that it can identify the correct version of the history to believe in. One mechanism to achieve this is for the (re)spawning node to contact a list of nodes the majority of whom are alert.

5. Next, our protocol defines each contiguous T_{epoch} time steps to be an epoch. At the beginning of each epoch, committee reconfiguration is performed in the following manner. First, nodes find the latest prefix (henceforth denoted $chain_{-2\omega}$) in their local chain whose timestamp is at least 2ω steps ago. This prefix $chain_{-2\omega}$ will be used to determine the next committee — and Snow White defers to the application-layer to define how specifically to extract the next committee from the state defined by $chain_{-2\omega}$. Next, nodes find the latest prefix (henceforth denoted $chain_{-\omega}$) in their local chain whose timestamp is at least ω steps ago. Given this prefix $chain_{-\omega}$, we extract the nonces contained in all blocks, the resulting concatenated nonce will be used to seed the hash function H for the next epoch.

2.6 Related Work

We briefly review the rich body of literature on consensus protocols including permissioned and permissionless consensus. Much of this section borrows from our earlier work [9].

Models for permissioned consensus protocols. Consensus in the permissioned setting [6, 10, 11, 13, 17, 20, 21, 23–25, 29, 30, 36–40, 50] 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 [39] where nodes are interconnected with authenticated channels[4]; and the "*authenticated Byzantine*" model [20], where a public-key infrastructure exists, such that nodes can sign messages and such digital signatures can then be transferred.

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

Permissioned, synchronous protocols. Many feasibility and infeasibility results have been shown. Notably, Lamport et al. [39] 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 [20], 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 [20] — however, if randomness is allowed, existing works have demonstrated expected constant round protocols that can tolerate up to a half corruptions [23, 30].

Permissioned, asynchronous protocols. A well-known lower bound by Fischer, Lynch, and Paterson [24] 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 [13, 25, 43], nodes' local randomness [6, 50], or randomness in network delivery [11]; 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 [17] and FaB [40] 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. [21], 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 [4]. 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 [44]. Upon closer examination, these protocols circumvent known lower bounds such as those by Canetti et al. [4] and Lamport et al. [39] 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 [26, 45–47]. Notably, Garay et al. [26] formally analyze the Nakamoto blockchain protocol in synchronous networks. Pass et al. [45] extend their analysis to asynchronous networks. More recently, Pass and Shi [47] 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 [22], where a coalition wielding $\frac{1}{3}$ of the computation power can reap up to a half of the rewards. Pass and Shi [46] recently show how to design a fair blockchain (called Fruitchains) from any blockchain protocol with positive chain quality. Since our Snow White 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 [46] to build a fair blockchain from Snow White.

Dynamic reconfiguration for consensus protocols. Dynamic reconfiguration has been studied in the classical setting for permissioned consensus. For example, Vertical Paxos [38] and BFT-SMART [10] allow nodes to be reconfigured in a dynamic fashion. The more recent Hybrid Consensus protocol by Pass and Shi [47] also performs committee reconfiguration over time, however, their protocol requires that in some transient windows, multiple instances of the permissioned consensus protocol are run concurrently.

In this paper, we also consider dynamic reconfiguration, but we consider it for blockchain-style protocols and rely on new techniques. One compelling advantage of our approach is that group reconfiguration is seamless in our protocol and does not need to introduce special execution paths. We also do not need to invoke multiple concurrent consensus instances.

Proofs-of-stake. The idea of having a cryptocurrency that relies on proofs-of-stake was discussed first in an online forum [7, 19], and subsequently considered in various works [7, 8, 12, 33, 35, 41, 51]. Various types of protocols have been considered, for example, those that combine proofs-of-work and proofs-of-stake [8, 19], those that rely on variants of classical consensus [35, 42], and those that more closely resemble a blockchain where elected leaders sign blocks [5, 7, 34]. Superficially, the latter category of protocols [5, 7, 34] is closest to our work. However, at the moment we are not able to prove any of these existing candidates secure. Many subtle choices made in our protocol are crucial to proving security, e.g., the precise timestamp rules, and the correct parametrization of the difficulty parameter — existing candidates lack one or more of these crucial ingredients, and therefore we know of no proof to any existing protocol. We stress that these details are important, and even small changes to our protocol can cause our proofs to break in the best case, or open up to possible attacks in the worst.

Recently, well-known cryptocurrencies such as Ethereum has declared it a pressing priority to switch to a proof-of-stake protocol, and they are currently developing a candidate called Casper [12]. Thus far, to the best of our knowledge, no protocol has offered formal security, which we believe is of critical importance for a consensus protocol, especially one that underlies a cryptocurrency system that carries high value.

3 Definitions

3.1 Protocol Execution Model

We first describe our basic protocol execution model. Our protocol execution model extends and enriches the sleepy model [9]. To capture a more powerful adversary, we make the following notable changes in modeling in comparison with sleepy [9]:

- We allow dynamic node spawning whereas the basic sleepy model [9] requires that all nodes (including honest and corrupt ones) be spawned prior to protocol start.
- We allow the adversary to issue `corrupt` and `sleep` instructions after protocol execution starts, whereas in the basic sleepy model [9], all `corrupt` and `sleep` instructions must be declared upfront prior to protocol start. Later when we describe our protocol, we will specify further

constraints on the adversary, in particular, under what conditions the adversary can corrupt nodes or make them sleep.

- Our model distinguishes between a *light sleeper* and a *deep sleeper* and treats them differently. As explained in Sections 2.1 and 5, such a distinction is necessary so as not to tread on theoretical impossibility.

Below we describe our execution model. We note that this section focuses on describe the *basic* execution model. We defer it to later sections to specify precise constraints (e.g., how long it takes for corruption to take effect, what parameters are admissible, etc.) that must be placed on the adversary to prove our protocol secure.

We assume a standard Interactive Turing Machine (ITM) model [14–16] 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. At any point of time, the environment $\mathcal{Z}$ can communicate with corrupt nodes in arbitrary manners. This also implies that the environment can see the internal state of corrupt nodes. Corrupt nodes can deviate from the prescribed protocol arbitrarily, i.e., exhibit byzantine faults. All corrupt nodes are controlled by a probabilistic polynomial-time adversary denoted $\mathcal{A}$, and the adversary can see the internal states of corrupt nodes. For honest nodes, the

environment cannot observe their internal state, but can observe any information honest nodes output to the environment by the protocol definition. Specifically, we assume the following corruption model.

- *Spawn.* At any time, $\mathcal{Z}$ can spawn fresh nodes, either alert or corrupt ones.

  We assume that upon spawning an alert node i at (the beginning of) time t, $(\mathcal{A}, \mathcal{Z})$ must deliver an initialization message to node i. Our protocols later will impose further constraints on this initialization message, and these constraints may imply additional trust assumptions necessary for a node to securely join the protocol. Therefore we defer requirements for the initialization message to protocol-specific compliance rules.

  We allow the adversary $\mathcal{A}$ to spawn *corrupt* nodes on its own without informing $\mathcal{Z}$.

- *Corrupt.* At any time t, $\mathcal{A}$ can issue to $\mathcal{Z}$ a corruption instruction of the form:

  $$(\texttt{corrupt}, i, t') \text{ where } t' \geq t$$

  A $(\texttt{corrupt}, i, t')$ instruction causes node i to become corrupt at time $t' \geq t$ (if it did not already become corrupt earlier).

- *Sleep.* At any time t, $\mathcal{A}$ can issue to $\mathcal{Z}$ a sleep instruction of the form:

  $$(\texttt{sleep}, i, t_0, t_1) \text{ where } t_0 \leq t \leq t_1$$

  A $(\texttt{sleep}, i, t_0, t_1)$ instruction causes node i to be asleep (or sleeping/sleepy) between time $[t_0, t_1]$ — as long as it did not already become corrupt earlier. A sleeping honest node (also called a sleeper) stops receiving or sending messages. If a sleeper does not become corrupt during the time it is asleep, it may wake up later again.

  Our model distinguishes between a deep sleeper and a light sleeper. A sleeper that sleeps for a long time before waking up is called a deep sleeper and one that wakes up soon is called a light sleeper. The definition of long and short depends on the protocol, and therefore we defer its specification to protocol-specific compliance rules.

  When a light sleeper wakes up, $(\mathcal{A}, \mathcal{Z})$ is required to deliver a wakeup message that is an unordered set 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.

  By contrast, a deep sleeper waking up is treated the same way as node respawning. Specifically, $(\mathcal{A}, \mathcal{Z})$ is required to resend the node an initialization message which must satisfy the same requirement of an initialization message for a newly spawning node.

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*.

3.2 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 $\kappa \geq \kappa_0$.

Convention for parameters. 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.*

Compliant executions. In this paper, for each protocol we introduce (including intermediate ones used in the proofs), we will define compliant executions by specifying a set of constraints on the p.p.t. pair $(\mathcal{A}, \mathcal{Z})$. Roughly speaking, our theorems will state that desirable security properties are respected except with negligible probability in any compliant execution. Since compliance is defined per protocol, we will often use the notation Π-compliant $(\mathcal{A}, \mathcal{Z})$ to mean that $(\mathcal{A}, \mathcal{Z})$ must respect the constraints expected by the Π protocol.

4 Preliminaries: Blockchain Formal Abstraction

In this section, we define the formal abstraction and security properties of a blockchain. Our definitions follow the approach of Pass et al. [45], which in turn are based on earlier definitions from Garay et al. [26], and Kiayias and Panagiotakos [32].

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 sleep 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.

Later in the text, if the context is clear, we sometimes omit writing a subset of the sub- or super-scripts and/or the view — for example, sometimes we simply write chain if the context is clear.

4.1 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 Π-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:

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

4.2 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 Π-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:

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

Global $\mathcal{G}_{\text{sign}}^{\Sigma}$ functionality (possibly shared with other protocols)

On initialization: $\Gamma = \emptyset$

On receive $\texttt{gen}$ from $\mathcal{P}$:

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \Sigma.\mathsf{gen}(1^{\kappa})$, and add the tuple $(\mathcal{P}, \mathsf{pk}, \mathsf{sk})$ to table Γ

Notify $\mathcal{A}$ of $(\mathcal{P}, \mathsf{pk})$, and return pk

On receive $\texttt{sign}(\mathsf{pk}, \mathsf{msg})$ from $\mathcal{P}$ in protocol sid:

assert that a tuple of the form $(\mathcal{P}, \mathsf{pk}, \mathsf{sk}) \in \Gamma$ exists for some sk

return $\Sigma.\mathsf{Sign}_{\mathsf{sk},sid}(\mathsf{msg})$

On receive $\texttt{getkey}(\mathcal{P})$ from $\mathcal{A}$: if $\mathcal{P}$ is corrupt, return all tuples in Γ of the form $(\mathcal{P}, \_, \_)$ to $\mathcal{A}$

Figure 1: Global signing functionality, parametrized by a signature scheme denoted $\Sigma = (\mathsf{Sign}, \mathsf{Ver})$. We use the shorthand $\mathsf{Sign}_{\mathsf{sk},sid}$ and $\mathsf{Ver}_{\mathsf{pk},sid}$ to denote that the message is prefixed with the protocol's session identifier sid.

4.3 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 Π-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).

5 The Snow White Protocol

5.1 Modeling Digital Signatures

Our protocol makes use of digital signatures. We model digital signatures in a way such that the signature keys can be shared between our consensus protocol and any application-level protocol. For

example, imagine that the cryptocurrency layer uses the same signing keys to sign transactions. Our modeling approach guarantees that the security of our Π_{sleepy} protocol is retained when composed with arbitrary application-level protocols, as long as the application-level protocols respect the compliance rules expected by Π_{sleepy}.

Specifically, we follow the GUC paradigm [15] and model the signature as a signing functionality shared across protocols. Figure 1 illustrates this signature functionality denoted $\mathcal{G}_{\text{sign}}^{\Sigma}$, which is parametrized by a signature scheme denoted Σ. We often omit writing the superscript Σ without risk of ambiguity. We now explain the $\mathcal{G}_{\text{sign}}$ functionality. $\mathcal{G}_{\text{sign}}$ generates and remembers a new signature key pair for a party upon the `gen` call; and signs messages for parties upon the `sign` call using any of the party's signing keys. Finally upon a `getkey` query, $\mathcal{G}_{\text{sign}}$ discloses the secret signing keys of corrupt parties to the adversary $\mathcal{A}$.

Note that in practice, such a functionality is actually realized in the following way: every honest node implements a trusted signing wrapper that is shared across all protocols instances executed by the honest node. This trusted signing wrapper is in charge of generating signature keys and perform signing operations. Following the GUC modeling paradigm [15], the union of the trusted signing wrappers across all honest nodes is considered as the trusted computing base (TCB), and therefore conceptually grouped into this single functionality $\mathcal{G}_{\text{sign}}$. When a node becomes corrupt, its signing wrapper is then controlled by the adversary, therefore the secret signing keys get disclosed to the adversary.

Like in the standard GUC paradigm, we assume that the environment $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$ in the following ways:

- $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$ *acting as an honest party* executing other (possibly rogue) protocols. Since other protocols have different session identifiers, $\mathcal{Z}$ cannot ask $\mathcal{G}_{\text{sign}}$ to sign messages pertaining to the challenge session identifier, which is the protocol instance that we are proving security for.
- $\mathcal{Z}$ can interact with $\mathcal{G}_{\text{sign}}$ *acting as a corrupt party or* $\mathcal{A}$ by routing messages through the adversary $\mathcal{A}$.

Mapping from public keys to nodes. In addition to defining honest, alert, and corrupt for nodes, it will be convenient later for us to refer to public keys as being honest, alert, or corrupt. This is defined in the most natural manner.

Given an execution trace denoted view, a public key pk is said to be honest (or alert resp.) at time $t \leq |\mathsf{view}|$ in view, if some tuple of the form $(\mathcal{P}, \mathsf{pk}, \_) \in \mathcal{G}_{\text{sign}}.\Gamma$ at time t in view, and further, $\mathcal{P}$ is honest (or alert resp.) at time t in view. If a public key pk is not honest at t, we say that it is corrupt at t. Note that a corrupt pk may not exist in $\mathcal{G}_{\text{sign}}$.

5.2 Format of Real-World Blocks

We use the notation *chain* to denote a real-world blockchain. Our protocol also defines an $\mathsf{extract}$ function that outputs an ordered list of transactions from a blockchain. A real-world blockchain is a chain of real-world blocks. We now define a valid block and a valid blockchain.

Valid blocks. We say that a tuple

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

is a valid block iff

1. $\Sigma.\mathsf{Ver}_{\mathsf{pk},sid}((\mathsf{h}_{-1}, \mathsf{txs}, \mathsf{time}, \mathsf{nonce}); \sigma) = 1$ where sid is the session identifier of the proof-of-stake protocol — as mentioned earlier, we use the notation $\mathsf{Ver}_{\mathsf{pk},sid}$ to indicate that the message is prefixed with the protocol's session identifier sid; and
2. $h = \mathsf{d}(h_{-1}, \mathsf{txs}, \mathsf{time}, \mathsf{nonce}, \mathsf{pk}, \sigma)$, 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 $\mathsf{eligible}^t(\mathit{chain}, \mathsf{pk})$ be a function that given the current state of chain, determines whether pk is elected as a leader in time step t, by making calls to a random oracle H. We defer the concrete specification of $\mathsf{eligible}^t(\mathit{chain}, \mathsf{pk})$ to Figure 2.

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

- $\mathit{chain}[0] = \mathit{genesis} := (\bot, \bot, \mathsf{time} = 0, \mathsf{nonce}_0, \bot, \bot, h = \vec{0})$ where nonce_0 is a nonce randomly generated prior to protocol start;
- $\mathit{chain}[-1].\mathsf{time} \leq t$; 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 $t := \mathit{chain}[i].\mathsf{time}$, $\mathsf{pk} := \mathit{chain}[i].\mathsf{pk}$, it holds that $\mathsf{eligible}^t(\mathit{chain}[:i-1], \mathsf{pk}) = 1$.

5.3 Epoch-Based Committee Election

Epochs. Our protocol proceeds in epochs, where in each epoch, a different committee will be elected and will be eligible to mine blocks. Let T_{epoch} be a protocol parameter that denotes the length of each epoch. We define a round-down function

$$\mathsf{rnddown}(t) := \lfloor \frac{t}{T_{\text{epoch}}} \rfloor \cdot T_{\text{epoch}}$$

to denote the starting time of the epoch that time t belongs to.

Per-epoch committee election. Let view be an execution trace of non-zero support where the current time $t := |\mathsf{view}|$. Let i denote a node that is honest at time t in view. Let $\mathit{chain} := \mathit{chain}_t^i(\mathsf{view})$ be node's i's chain at time t in view. At this point of time, node i will apply an election function to decide the set of public keys eligible in the current time step t. To this end, node i will look at its local chain, and select a block that is sufficiently far back — the set of public keys contained in the prefix up to this block will be eligible to mine in time t.

We have yet to define what it means to be "sufficiently far back". To this end, we define a look-back parameter denoted ω. An honest node i will select the largest index j such that $\mathit{chain}[j].\mathsf{time} + 2\omega \leq \mathsf{rnddown}(t)$. Then the public keys extracted by calling $\mathsf{extractpks}(\mathit{chain}[:j])$

will be the committee for time t. Later, we will choose the parameter ω to be reasonably large (and yet not too long ago) such that all honest nodes will agree on committee at time t with overwhelming probability.

$\mathsf{elect\_cmt}^t(chain) :=$
$\quad \mathsf{extractpks}(chain[:j])$ where j is the largest index s.t. $chain[j].\mathsf{time} + 2\omega \leq \mathsf{rnddown}(t)$

Per-epoch hash reseeding. Given a set of eligible committee members, a hash function will be applied to choose a leader for each time step. We now define a rule for selecting this hash function. Let H denote a globally known hash function modeled as a random oracle. We henceforth use the notation

$$\mathsf{H}^{\mathsf{nonce}}(x) := \mathsf{H}(\mathsf{nonce}||x)$$

In other words, we elect a hash by choosing a nonce.

$\mathsf{elect\_h}^t(chain) := chain[j].\mathsf{nonce}$ where j is the largest index s.t. $chain[j].\mathsf{time} + \omega \leq \mathsf{rnddown}(t)$

Later, we will choose an appropriate parameter ω such that

1. ω is reasonably large (and yet not too long ago) such that all honest nodes will agree on the hash function elected at time t with overwhelming probability; and
2. ω is smaller than 2ω by a reasonable margin, such that the hash will be selected sufficiently long after the committee is determined by the blockchain.

5.4 Protocol Overview

We describe our Snow White protocol in Figure 2. The protocol proceeds in epochs whose length is determined by the parameter T_{epoch}. At the start of each epoch, the protocol switches to a new committee that can be determined by examining the current state of the blockchain. Further, a new hash is used for each different epoch, and the hash is selected by computing a new nonce from the current state of the blockchain.

Once a committee and a hash is determined for an epoch, we can now describe the "mining" process. Let pks_e denote the e-th committee. In every time step during the e-th epoch, if a node $i$is in the e-th committee, it will compute $\mathsf{H}(\mathsf{pk}, t)$ and if the outcome is smaller than D_p, then node i is a leader in time step t. In this case, node i will extend its current chain by signing a new block containing the following: 1) the previous block's hash, 2) a set of transactions to be confirmed, 3) the current time, 4) a freshly generated nonce, and 5) its own public key. The node then announces the new chain to the network.

In each time step, regardless of whether a node is in the present committee, a node receives chains from the network and verify their validity. If a received chain is valid but deviates from a node's current chain too far in the past, such a chain is not punctual and will be rejected. Nodes always choose the longest chain among all chains it did not reject.

Finally, when a node spawns or wakes up from deep sleep (henceforth referred to as respawning), an initialization procedure is invoked. At this moment, $\mathcal{A}$ must deliver to the node an initialization message containing a list of chains denoted $\{chain_i\}_{i \in L}$ such that the majority of these chains reflect the true state of an alert node at time $t - 1$ (see protocol compliance rules defined in Section 5.5).

Protocol $\Pi_{\text{snowwhite}}(\kappa_0, p, \omega, T_{\text{epoch}}, \mathsf{extractpks})$

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

let $\mathsf{pk} := \mathcal{G}_{\text{sign}}.\mathtt{gen}()$, output pk to $\mathcal{Z}$, wait to receive $(\mathsf{pks}_0, \{chain_i\}_{i \in L})$

find the longest valid $chain_0$ that is a prefix of the majority of chains in $\{chain_i\}_{i \in L}$

find the longest valid $chain \in \{chain_i\}_{i \in L}$ such that $chain_0 \prec chain$

record $chain$ and pk

On receive $chain'$:

assert $|chain'| > |chain|$ and $chain'$ is valid w.r.t. the current time t

assert $chain[: -\kappa_0] \prec chain'$

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

Every time step:

- receive input $\mathtt{transactions}(\mathsf{txs})$ from $\mathcal{Z}$, and pick $\mathsf{nonce} \leftarrow_{\$} \{0,1\}^{\kappa}$
- let t be the current time, if $\mathsf{eligible}^t(chain, \mathsf{pk})$:

  let $\sigma := \mathcal{G}_{\text{sign}}.\mathtt{sign}(\mathsf{pk}, chain[-1].h, \mathsf{txs}, t, \mathsf{nonce})$, $h' := \mathsf{d}(chain[-1].h, \mathsf{txs}, t, \mathsf{nonce}, \mathsf{pk}, \sigma)$,

  let $B := (chain[-1].h, \mathsf{txs}, t, \mathsf{nonce}, \mathsf{pk}, \sigma, h')$, 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 $\mathsf{eligible}^t(chain, \mathsf{pk})$

Assume: $chain[0].\mathsf{nonce} = \mathsf{nonce}_0$, $\mathsf{extractpks}(chain[: 0]) = \mathsf{pks}_0$

Let $\mathsf{elect\_cmt}^t(chain)$ be a function that returns $\mathsf{extractpks}(chain[: j])$ s.t. j is the largest index satisfying $chain[j].\mathsf{time} + 2\omega \leq \mathsf{rnddown}(t)$

Let $\mathsf{elect\_h}^t(chain)$ be a function that returns $\mathsf{extractnonce}(chain[: j])$ s.t. j is the largest index satisfying $chain[j].\mathsf{time} + \omega \leq \mathsf{rnddown}(t)$

Let $\mathsf{pks}^* := \mathsf{elect\_cmt}^t(chain)$, let $\mathsf{nonce}^* := \mathsf{elect\_h}^t(chain)$

Return 1 if $\mathsf{H}^{\mathsf{nonce}^*}(\mathsf{pk}, t) < D_p$ and $\mathsf{pk} \in \mathsf{pks}^*$; else return 0

$\mathsf{extractnonce}(chain)$: output the concatenation of the nonces in all blocks in $chain$

Figure 2: The Snow White consensus protocol. The difficulty parameter D_p is set such that a committee member is elected leader with probability p in a single time step. pks_0 denotes the initial committee. Chain validity is stated w.r.t. $\mathsf{eligible}$ although we omit writing w.r.t. $\mathsf{eligible}$ for simplicity.

Table 1: Notations

κ	security parameter
chain_i^t	extracted ideal-world chain for node i honest at time t
$chain_i^t$	real-world formatted chain for node i honest at time t
W	posterior corruption window
ϕ	ϕ fraction more must be alert and remain honest for W more steps than those corrupt within W steps
τ	agility parameter, time till corruption/sleep operations take effect
Δ	maximum network delay for alert nodes
$\widetilde{\Delta}$	maximum duration of a light sleep
2ω, ω	time to look back to decide committee/hash respectively
$\kappa_0 := \frac{\kappa}{2}$	incoming chain must agree with all but the last κ_0 blocks
p	probability that a node gets elected leader in any time step
T_{epoch}	length of an epoch

If this is the protocol start, this list can simply be the genesis block. As mentioned earlier, this reflects the fact that a spawning (or respawning) node can contact a list of nodes in the network the majority of whom must be alert. As we argue in Sections 2.1 and 6, this process allows a spawning or respawning node to determine the correct version of history to believe in. Without this additional trust assumption, consensus would have been impossible in the presence of majority posterior corruption. Now the spawning/respawning node computes its state as follows: First, it computes the longest valid $chain_0$ that is a prefix of the majority of chains in the list. Next, it finds the longest *chain* in the list that contains $chain_0$. This *chain* now becomes the internal state of the spawning/respawning node.

Remark: committee members and non-members. We remark that in each epoch, there are two types of nodes in the system, the current committee members and committee non-members. Although only committee members are contributing blocks, our consistency and liveness guarantees extend to all nodes, including members and non-members.

5.5 Compliant Execution

We now articulate a set of constraints that $(\mathcal{A}, \mathcal{Z})$ must respect for our protocol to guarantee security.

Summary of notations. For convenience, we summarize our notations and parameters in Table 1.

Additional useful notations. It will also be useful to define some derived variables. 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.

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. Without loss of generality due to rescaling of κ, we shall henceforth assume that

$$\kappa_0 = \frac{\kappa}{2}$$

We say that the parameters $(p, \kappa_0, T_{\text{epoch}}, \omega;\ n, \phi, \Delta, \tau, \widetilde{\Delta}, W)$ are $\Pi_{\text{snowwhite}}$-admissible iff the following constraints hold:

- $np\Delta < 1$ and moreover, there exists a constant $\psi > 0$ such that

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

- $W > \omega \geq \frac{2\kappa}{\gamma} + \widetilde{\Delta}$;
- $T_{\text{epoch}} \geq 3\omega$;
- $\tau > W + T_{\text{epoch}} + 2\omega$;

Intuitions for admissible parameters. We now given an intuitive explanation for these parameters. All these intuitions will later arise as technicalities in our proof.

- First, the requirement $(1-2\alpha(\Delta+1))\alpha > (1+\psi)\beta$ roughly says that the alert committee members that remain honest till the near future, even when discounted by a parameter related to the network delay, must outnumber the committee members that are corrupt or to become corrupt in the near future — where "near future" is characterized by the posterior corruption window W. Specifically, the discount factor $(1-2\alpha(\Delta+1))$ arise due to technicalities that arise in the consistency proof.
- Second, the look-back parameters 2ω and ω must be reasonably large, such that
  1. The prefix of the chain that is used to decide the next epoch's committee and hash has stabilized, such that all nodes will agree on the next epoch's committee and hash;
  2. The two look-back parameters are spaced out far enough such that when the committee is determined, the adversary cannot predict the nonce that determines the next hash; and
  3. Further, the parameters ω and 2ω are related to the light sleep bound $\widetilde{\Delta}$ and the punctuality parameter $\kappa_0 = \frac{\kappa}{2}$, ensuring that even when a light sleeper wakes up, it suffices to use its old chain (before going to sleep) with the last κ_0 blocks removed — henceforth denoted $\mathit{chain}^s[:-\kappa_0]$ where s is the time the node last went to sleep — to decide the next committee and hash. Specifically, this requires that $\mathit{chain}^s[:-\kappa_0]$ must have a block with a recent enough timestamp relative to ω and 2ω.

- Next, nodes reject blocks that modify κ_0 blocks back into their past, and for light sleepers this is adjusted by another $\widetilde{\Delta}$ parameter — therefore the posterior corruption window W has to be reasonably large to be commensurate with these two parameters.

- Next, the epoch length T_{epoch} has to be reasonably large, since as we mention in Section 2.2 and Section 9.1, once a random oracle is chosen, it must be used sufficiently many times to prove security. Also in our current parameterization, we do not treat the first epoch specially, so T_{epoch} must also be large enough for the protocol to warm up — roughly speaking, the blockchain must be at least 2ω time long for a committee (that is not the initial committee) to be determined.

- Finally, the agility parameter τ, which stipulates how long it takes for `corrupt` and `sleep` instructions to take effect, must be reasonably large to eliminate possible "adaptive" corruption behaviors, where the adversary first sees the next committee and hash, and then decides who to corrupt or make sleep. If τ is sufficiently large, such an attack will not succeed. Specifically, if an adversary attempts to corrupt a node (or make it sleep) after seeing the next hash, then when the `corrupt` or `sleep` instruction takes effect, it will already be well after this epoch for such "adaptivity" to be effective, where the notion of "well after" is related to the posterior corruption parameter W. Roughly speaking, from the time the adversary sees the next hash till "well after" the next epoch takes a total of $W + T_{\text{epoch}} + c \cdot \omega$ time; therefore, the requirement that $\tau > W + T_{\text{epoch}} + 2\omega$ is easy to understand.

Compliant executions. We say that the pair $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{snowwhite}}$-compliant if the following holds for any view with non-zero support:

- **Initialization.** At the start of the execution, the following happens. First, $\mathcal{Z}$ can spawn a set of either honest or corrupt nodes. $\mathcal{Z}$ learns the honest nodes' public keys after calling their `init`() procedure. Next, $\mathcal{Z}$ provides the inputs (pks_0, $\{genesis\}$) to all honest nodes.

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

- **τ-agility.** Whenever $\mathcal{A}$ issues a $(\texttt{corrupt}, t)$ or a $(\texttt{sleep}, t, t')$ instruction at time r, it must hold that $t - r \geq \tau$.

- **Sleeping.** If a sleeper wakes up within $\widetilde{\Delta}$ time since it last went to sleep, it is considered a light sleeper. Otherwise, if it sleeps for more than $\widetilde{\Delta}$ time before waking up it is considered a deep sleeper and must reinitialize as if it is re-spawning (see below).

- **Spawning.** When a new node spawns or a deep sleeper wakes up, $(\mathcal{A}, \mathcal{Z})$ must deliver the same pks_0 to this node, and further $(\mathcal{A}, \mathcal{Z})$ must deliver to this node a message $\{chain_i^{t-1}\}_{i \in L}$ that contains the internal chains of a set of nodes (denoted L) the majority of whom are alert at $t - 1$. If a node $i \in L$ is corrupt[5], it can provide an arbitrarily $chain_i^{t-1}$.

  Intuitively, this captures the requirement that a newly spawning node must be able to connect to a subset of nodes the majority of which are alert.

[5]Considering that in practice, it may take Δ time for messages to be transmitted to the newly spawned node, it is possible to relax this condition where $(\mathcal{A}, \mathcal{Z})$ is only required to deliver to a spawning node a message $\{chain_i^{t_i}\}_{i \in L}$ where for more than majority of L, it must hold that i is alert at $t_i \in [t - \Delta, t]$. It is not hard to adjust the proofs of our theorems to this relaxed case.

- **Resilience.** Let $t \leq |\mathsf{view}|$, and let i be a node that is honest at time t in view. Let $chain_i^t(\mathsf{view})$ denote node i's protocol internal state at time t in view, and define

$$\mathsf{cmt}_i^t(\mathsf{view}) := \mathsf{elect\_cmt}^t(chain_i^t(\mathsf{view}))$$

We require that for every $t \leq |\mathsf{view}|$, for every honest node i that is honest at t in view, let $r = \min(t + W, |\mathsf{view}|)$,

$$\frac{\mathsf{alert}^t(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}^r(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view})}{\mathsf{corrupt}^r(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view})} \geq 1 + \phi \tag{5}$$

where alert^t, honest^r, $\mathsf{corrupt}^r$ are defined as below:

  - $\mathsf{alert}^t(S, \mathsf{view})$ outputs those in S that are alert at time t in view.
  - $\mathsf{honest}^r(S, \mathsf{view})$ outputs those in S that are honest at time r.
  - $\mathsf{corrupt}^r(S, \mathsf{view})$ outputs those in S that are corrupt at time r.

  Informally, we require that among committee of time t (as perceived by any node honest at time t), more are alert at time t and remain honest till r (but possibly can go to sleep), than those corrupt at time r.

- **Number of awake nodes.** For every honest node i that is honest at time t in view, let $r = \min(t + W, |\mathsf{view}|)$, we have that

$$(\mathsf{alert}^t(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}^r(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view})) + \mathsf{corrupt}^r(\mathsf{cmt}_i^t(\mathsf{view}), \mathsf{view}) = n$$

- **Admissible parameters.** The parameters $(p, \kappa_0, T_{\text{epoch}}, \omega; n, \phi, \Delta, \tau, \widetilde{\Delta}, W)$ are $\Pi_{\text{snowwhite}}$-admissible, where $p, \kappa_0, T_{\text{epoch}}, \omega$ are input parameters to the $\Pi_{\text{snowwhite}}$ protocol, and $(n, \phi, \Delta, \tau, \widetilde{\Delta}, W)$ are parameters related to $(\mathcal{A}, \mathcal{Z})$.

5.6 Theorem Statement

Theorem 1 (Security of $\Pi_{\text{snowwhite}}$)**.** For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \kappa$, $\Pi_{\text{snowwhite}}$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0-consistency against any $\Pi_{\text{snowwhite}}$-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 5.5.

The proof of this theorem will be provided in Sections 8, 9, and 10.

6 Lower Bounds

Recall that in our protocol, when a node first spawns or after a deep sleeper wakes up, the node must perform an initialization procedure where it contacts a list of nodes the majority of whom are alert. We show that this additional trust assumption is necessary if one wishes to tolerate majority posterior corruption.

We state our lower bound for a blockchain protocol, but it is not hard to see that the same lower bound proof holds for any consensus protocol (often referred to as state machine replication in the classical distributed systems literature) as defined by Pass and Shi [47]. Note that Pass and Shi also show that a formal blockchain abstraction implies a classical consensus (i.e., state machine replication) abstraction.

Theorem 2 (Access to a majority honest set for (re)spawning nodes is necessary)**.** Assuming common knowledge of the initial committee cmt_0, and absent any additional trust assumptions, it is impossible to realize a secure blockchain protocol in our execution model if there exists $\frac{1}{\mathsf{poly}(\kappa)}$ fraction of views such that no node ever sleeps, and at some time T,

$$\mathsf{corrupt}^T(\mathsf{cmt}_0, \mathsf{view}) > \mathsf{alert}^1(\mathsf{cmt}_0, \mathsf{view}) \cap \mathsf{honest}^T(\mathsf{cmt}_0, \mathsf{view})$$

The lower bound holds even if at any time, there are more alert present committee members than corrupt ones, even if all corruptions are declared statically upfront, and even if assuming a PKI.

Proof. Consider the following $(\mathcal{A}, \mathcal{Z})$ pair:

- $(\mathcal{A}, \mathcal{Z})$ first provides an initial committee cmt_0 consisting of $n = 2f + 1$ nodes, where f of them are corrupt, and the remaining are alert. Then at time T, one additional node among cmt_0 becomes corrupt — at this moment, $f + 1$ among cmt_0 are corrupt, and f are still alert.
- $(\mathcal{A}, \mathcal{Z})$ constructs appropriate transactional inputs such that this will cause the committee to switch completely at some time $t^* < T$, such that the new committee, denoted cmt_1, does not intersect with cmt_0. Further, $(\mathcal{A}, \mathcal{Z})$ makes sure that all nodes in cmt_1 are alert all the time.
- At time T, when the majority of cmt_0 become corrupt, $(\mathcal{A}, \mathcal{Z})$ creates a simulated execution in its head with the $f + 1$ corrupt cmt_0 members that he has: in the simulated execution, a different set of transactions are provided to the initial committee cmt_0, such that at time t^* the simulated execution switches to a new committee cmt'_1 consisting only of corrupt nodes. In this way, $(\mathcal{A}, \mathcal{Z})$ can continue with the simulation after the committee switch. Further, S'_1 also does not intersect with cmt_0 just like the real execution.
- $(\mathcal{A}, \mathcal{Z})$ spawns a new alert node i after time T, and delivers messages from both the simulated and the real executions to node i.

Since the simulated execution and the real one are identically distributed, the newly joining node i cannot output the correct log with probability more than $\frac{1}{2}$. □

We note that the same lower bound proof holds for a deep sleeper that sleeps for a long time and then wakes up. In other words, if we changed our model to prevent dynamic spawning of nodes, but still allow sleeping, the same lower bound would still hold if majority posterior corruption can happen.

7 Extensions and Application to Proofs of Stake

In this section, we discuss how to apply the Snow White protocol in a proof-of-stake application.

Assumptions on distribution of stake. Roughly speaking, our Snow White protocol expects the following assumption for security: at any point of time, there are more alert committee members that will remain honest sufficiently long than there are corrupt committee members. In a proof-of-stake setting, we would like to articulate assumptions regarding the distribution of stake among stake-holders, and state the protocol's security in terms of such assumptions.

Since our Snow White protocol allows a committee reelection opportunity once every epoch, it is possible that the distribution of the stake in the system lags behind the committee election. However, suppose that this is not the case, e.g., pretend for now that there is no money transfer, then it is simple to translate Snow White's assumptions to distribution on stake. Imagine that the application-defined extractpks($chain$) function will output one public key for each unit of currency as expressed by the state of $chain$ — we do not care about the implementation details of extractpks($chain$), and in fact that is an advantage of our modular composition approach. In this way, our Snow White protocol retains security as long as the at any point of time, more stake is alert and will remain honest sufficiently long than the stake that is corrupt. Here when we say "a unit of stake is alert (or honest, corrupt, resp.)", we mean that the node that owns this unit of stake is alert (or honest, corrupt, resp.).

In the real world, however, there is money transfer — after all that is the entire point of having cryptocurrencies — therefore the committee election lags behind the redistribution of stake. This may give rise to the following attack: once a next committee is elected, the majority of the stake in the committee can now sell their currency units and perform an attack on the cryptocurrency (since they now no longer have stake). For example, the corrupt coalition can perform a double-spending attack where they spend their stake but attempt to fork a history where they did not spend the money.

One approach to thwart such an attack is to limit the liquidity in the system. For example, imagine that at any point of time, $a = 30\%$ of the stake is alert and will remain honest sufficiently long, $c = 20\%$ is corrupt, and the rest are sleepy. We can have the cryptocurrency layer enforce the following rule: only $\frac{a-c}{2} - \epsilon = 5\% - \epsilon$ of the stake can switch hands during every window of size $2\omega + T_{\text{epoch}} + W$. In other words, if in any appropriately long window, only l fraction of money in the system can switch hands, it holds that as long as at any time, $2l + \epsilon$ more stake is alert and remain honest sufficiently long than the stake that is corrupt, we can guarantee that the conditions expected by the consensus protocol, that is, at any time, more committee members are alert and remain honest sufficiently long, than the committee members that are corrupt.

Incentive compatibility. In a practical deployment, a very important desideratum is incentive compatibility. Roughly speaking, we hope that each node will earn a "fair share" of rewards and transaction fees — and in a proof-of-stake system, fairness is defined as being proportional to the amount of stake a node has. In particular, any minority coalition of nodes should not be able to obtain an unfair share of the rewards by deviating from the protocol — in this way, rational nodes should not be incentivized to deviate from the protocol.

Since Snow White relies on Sleepy which is a blockchain-style protocol, we also inherit the drawback of Nakamoto blockchain — it is well-known that there exists a selfish mining attack [22,45]

such that a minority coalition can increase its rewards by a factor of nearly 2 in the worst case.

In a recent work called Fruitchains, Pass and Shi [46] describes a novel idea that bootstraps a ϵ-fair blockchain from a standard Nakamoto blockchain [26,44,45]. In essence, Fruitchains proposes to use two mining processes piggybacked on top of each other, one for the underlying Nakamoto consensus for mining "blocks", and another one for mining "fruits". Fruitchains then relies on the first mining process's liveness to ensure that all honest nodes' work mining fruits will surely be incorporated and cannot be erased by an adversary. Pass and Shi [46] argue that Fruitchains can not only be used to fairly distribute fruit rewards[6], but also transaction fees — as long as transaction fees are distributed equally among a recent segment of roughly $\Omega(\frac{\kappa}{\epsilon})$ fruits, where κ is the security parameter and $\epsilon > 0$ is a constant that denotes the fairness parameter. If such a mechanism is adopted to distribute rewards and transaction fees, Fruitchains guarantees that no minority coalition (in terms of computation power) can gain ϵ-fraction more rewards and fees than its fair share.

We stress that the Fruitchains transformation also applies to our Snow White protocol, and we can rely on the same idea to achieve fairness and incentivize honest behavior. In particular, we can introduce a second mining process by having two hashes per epoch. The first hash is still used for mining blocks just like in the present Snow White protocol, and the second hash is used for mining "fruits". Just like in Fruitchains, mining rewards can be distributed to fruits rather than blocks, and transaction fees can be distributed equally among a recent segment of roughly $\Omega(\frac{\kappa}{\epsilon})$ fruits. Just like in Fruitchains, this guarantees that as long as at any time, there are more alert committee members that remain honest sufficiently long than corrupt committee members, the corrupt coalition cannot increase its share by more than ϵ no matter how it deviates from the prescribed protocol.

8 Proofs: Analyzing A Simplified Ideal Protocol

Proof roadmap. Instead of directly analyzing the real-world protocol which is rather complex, we first describe some ideal protocols where nodes interact with each other, and an ideal functionality will act as as a trusted third party and keep track of all legitimate chains. The ideal protocols are much simpler to analyze in comparison with the real-world protocol. Further, the ideal protocols are meant to capture of the essence of the real-world protocol in some way, such that analyzing possible attacks in the ideal protocols will be indicative of the possible attacks in the real-world protocol.

In this section, we start by analyzing a very simple ideal protocol denoted Π_{ideal}, and then through a sequence of hybrid steps. In the next section, we gradually augment the ideal protocol such that it becomes increasingly closer to the real-world protocol. At the end of this section, we will arrive at a hybrid protocol called Π_{hyb}, which captures ideal-world attack behavior but sends messages that contain real-world formatted chains. Finally, in Section 10, we will show that the real-world protocol $\Pi_{\text{snowwhite}}$ is as secure as the hybrid protocol Π_{hyb}.

8.1 Simplified Ideal Protocol Π_{ideal}

We first define a simplified protocol Π_{ideal} parametrized with an ideal functionality $\mathcal{F}_{\text{tree}}$ — see Figures 3 and 4. The ideal functionality $\mathcal{F}_{\text{tree}}$ allows the adversary $\mathcal{A}$ to choose the committee for

[6] "Orange is the new block" in Fruitchains.

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

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

On receive $\texttt{setpids}(t, \mathsf{pids}_t)$ from $\mathcal{A}$:
assert no tuple of the form $(t, \_)$ has been recorded
assert pids_t contain only parties that have been spawned
record (t, pids_t)

On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{A}$ or internally:
assert (t, pids_t) has been recorded, and $\mathcal{P} \in \mathsf{pids}_t$
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 3: Ideal functionality $\mathcal{F}_{\mathbf{tree}}$. The ideal functionality allows $\mathcal{A}$ to choose a committee on a time-based granularity. $\mathcal{A}$ is not able to query the `leader` entry point for time t until it has chosen a committee for time t.

Protocol Π_{ideal}

On receive $\texttt{init}(\mathsf{chain}_0)$: record $\mathsf{chain} := \mathsf{chain}_0$

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 4: Ideal protocol Π_{ideal}

every time step separately, specifically, by calling $\mathcal{F}_{\text{tree}}.\mathsf{setpids}(t, \mathsf{pids}_t)$. $\mathcal{F}_{\text{tree}}$ flips random coins to decide whether a committee member is the elected leader for every time step. Once the $\mathcal{A}$ commits to a committee for a specific time step t, it is now allowed to query a function called $\mathcal{F}_{\text{tree}}.\texttt{leader}$ that tells $\mathcal{A}$ which committee member is elected as the leader in time t (if any at all). However, $\mathcal{A}$ cannot query $\mathcal{F}_{\text{tree}}.\texttt{leader}(\_, t)$ for time t function before committing to the t-th committee (since otherwise $\mathcal{A}$ could adaptively choose a committee such that honest nodes never get elected as leaders). Further, we require that the adversary $\mathcal{A}$ follow a somewhat static corruption model: once it chooses a node i as a member of any committee, it is not allowed to corrupt node i any more — since otherwise $\mathcal{A}$ could simply query $\mathcal{F}_{\text{tree}}.\texttt{leader}$ and adaptively corrupt those that have been elected as leaders. 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.

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 of t if $\mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{time}(\mathsf{chain}) = t$.

Compliant $(\mathcal{A}, \mathcal{Z})$. A compliant $(\mathcal{A}, \mathcal{Z})$ pair for protocol Π_{ideal} is defined as a pair of p.p.t. algorithms such that every view of non-zero support satisfies the following constraints:

- **Sleeping.** No matter how long a node sleeps till it wakes up, it is treated as a light sleeper (as long as the node has not become corrupt during its sleep).
- **Spawning.** When a new, alert node spawns at time t, $(\mathcal{A}, \mathcal{Z})$ must deliver to it an initialization message chain_0 such that $\mathsf{chain}_0 \in \mathcal{F}_{\text{tree}}$ and chain_0 is no shorter than the shortest chain of any alert node at time $t-1$. If this is the protocol start, then chain_0 is simply $\mathsf{genesis}$. All spawned nodes must have distinct party identifiers.
- **A-priori commitment of future committee.** $\mathcal{A}$ must have called $\mathcal{F}_{\text{tree}}.\texttt{setpids}(t, \mathsf{pids}_t)$ before t. In other words, $\mathcal{A}$ must choose the committee pids_t before time t.
- **Epoch-wise somewhat static corruption.** Instead of delayed corruption/sleep, we consider a more permissive but easier to analyze corruption model. Roughly speaking, we require that $\mathcal{A}$ cannot adaptively corrupt a node after examining whether it is elected a leader in any time step. Further, $\mathcal{A}$ cannot adaptively make a node sleep for the duration $[t_0, t_1]$ after observing whether the node is elected leader during $[t_0, t_1]$. We formalize this intuition below.

  At any time $t \le t'$, $\mathcal{A}$ is allowed to issue $(\texttt{corrupt}, i, t')$ iff

  – $\mathcal{A}$ has not called $\mathcal{F}_{\text{tree}}.\texttt{setpids}(r, \mathsf{pids}_r)$ for any r such that $i \in \mathsf{pids}_r$;

  At time $t \le t_0 \le t_1$, $\mathcal{A}$ is allowed to issue $(\texttt{sleep}, i, t_0, t_1)$ iff

  – for every $r \in [t_0, t_1]$, $\mathcal{A}$ has not called $\mathcal{F}_{\text{tree}}.\texttt{setpids}(r, \mathsf{pids}_r)$ such that $i \in \mathsf{pids}_r$.

  In other words, after a node i has been selected for any committee, $\mathcal{A}$ can no longer corrupt it — this also means that if a node is honest when it is chosen into the committee, it will remain honest forever. Further, before choosing the e-th committee, $\mathcal{A}$ must commit to which nodes will be asleep and exactly when during epoch e.

- **Resilience.** At any time step t, let $\mathsf{cmt}^t(\mathsf{view})$ be the (t, pids_t) committee set that $\mathcal{A}$ sends to $\mathcal{F}_{\text{tree}}$ in view. It must hold that

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

  where $\mathsf{alert}^t(S, \mathsf{view})$ denotes those among S are alert at time t; $\mathsf{honest}(S, \mathsf{view})$ denotes those among S remain honest forever; and and $\mathsf{corrupt}(S, \mathsf{view})$ denotes those among S are ever corrupt in view.

- **Number of awake nodes.** Let $\mathsf{cmt}^t(\mathsf{view})$ be the (t, pids_t) committee set that $\mathcal{A}$ sends to $\mathcal{F}_{\text{tree}}$ in view. It must hold that for every $t \leq |\mathsf{view}|$,

$$(\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})) + \mathsf{corrupt}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) = n$$

  In other words, at every time step t, the number of alert committee members and the number of corrupt committee members must sum up to n.

- **Admissible parameters.** The parameters (p, n, ϕ, Δ) satisfy the following constraints: $np\Delta < 1$ and moreover, there exists a constant $\psi > 0$ such that

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

  where α and β are derived variables whose definitions were presented in Section 5.5.

Theorem 3 (Security of Π_{ideal})**.** For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \kappa$, $\Pi_{\text{snowwhite}}$ 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:

- 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 5.5.

Proof. Although our ideal protocol Π_{ideal} is different from the ideal protocol for Sleepy [9], we stress that the differences are inconsequential to the induced stochastic process. We claim that the proof follows in the same manner as that of Sleepy [9], by pointing out the differences between our ideal protocol and that of Sleepy [9].

Recall that Sleepy [9] defines an ideal protocol where they assume that there is a fixed committee known upfront. All nodes are spawned upfront, and all `corrupt` and `sleep` instructions are declared upfront. Our Π_{ideal} is more fine-grained: First, each time step can have a different committee. Second, nodes can get spawned dynamically, and `corrupt` and `sleep` instructions need not be declared at the time of spawning. However, it is important to observe that the compliance rule for our Π_{ideal} basically stipulates that from the perspective of every committee: 1) if the adversary wants a committee member to ever be corrupt, he must commit to this decision before seeing random coins that decide if the committee member gets elected as leader; and 2) if the adversary wants a committee member at t to be asleep at t, he also must commit to this decision before seeing the random coins that decide if this committee member is elected leader at t. □

9 Proofs: Intermediate Hybrid Protocols

9.1 Ideal Protocol with Adversarially Biased Hashes

We now consider a hybrid protocol denoted Π_{bias} that effectively allows the adversary to bias the hash functions. The definition of Π_{bias} is almost identical to Π_{ideal}, except that all honest nodes' interactions with $\mathcal{F}_{\text{tree}}$ are now replaced with $\mathcal{F}_{\text{bias}}$.

Compliant executions. A compliant $(\mathcal{A}, \mathcal{Z})$ pair for Π_{bias} is defined in almost the same way as a compliant environment for Π_{ideal}, except now we additionally require that

- $\mathcal{A}$ must have $\mathcal{F}_{\text{bias}}.\texttt{sethash}(e, \mathsf{nonce}_e)$ before epoch e starts. In other words, $\mathcal{A}$ must choose the next hash function before the next epoch begins.
- We additionally require that $T_{\text{epoch}} \geq \frac{\kappa}{\gamma}$.

A useful lemma. Henceforth we will sometimes use the terminology "hash for an epoch" to refer to the randomness used by $\mathcal{F}_{\text{bias}}$ (or $\mathcal{F}_{\text{tree}}$) for the epoch. Recall that the only difference between Π_{bias} and Π_{ideal} is that in Π_{bias}, when $\mathcal{F}_{\text{bias}}$ picks the hash for an epoch, the adversary is allowed to look at polynomially many choices for each epoch's hashes, and then instruct $\mathcal{F}_{\text{bias}}$ which hash to use. In particular, the adversary can choose the worst-case combination of different epochs' hashes to maximize its own advantage.

Given a view, we say that $\mathcal{A}$ looks at q hashes for an epoch in view, if all of its queries to $\mathcal{F}_{\text{bias}}.\texttt{leader}$ for a given epoch has q distinct nonces. Further, let $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa)$, and let $\mathsf{bad\text{-}event}(\mathsf{view})$ be a random variable defined over view. We now define the same $\mathsf{bad\text{-}event}(\mathsf{view}')$ over an execution trace $\mathsf{view}' \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa)$ in the most natural manner. In particular, view' can be thought of as a superset of the bits in view. We can define a function $\mathsf{compress}(\mathsf{view}')$ which removes all additional bits that are in view' but not in view, such as the $\mathcal{F}_{\text{bias}}.\texttt{sethash}$ calls made by $\mathcal{A}$; and moreover only the hashes chosen by $\mathcal{A}$ are preserved in $\mathsf{compress}(\mathsf{view}')$, the remaining hashes are thrown away in $\mathsf{compress}(\mathsf{view}')$. In this way we can define $\mathsf{bad\text{-}event}(\mathsf{view}') := \mathsf{bad\text{-}event}(\mathsf{compress}(\mathsf{view}'))$.

Lemma 1 (Union bound over small number of hashes)**.** Let $\mathsf{bad\text{-}event}(\mathsf{view}) \in \{0, 1\}$ be a random variable that depends only on the randomness for c epochs, i.e., there exists $E \subset \mathbb{N}$ where $|E| = c$, such that for any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, the following holds where $\vec{v}(\mathsf{view})$ returns the all the randomness $\mathcal{F}_{\text{tree}}$ generated for all epochs in view, and $v_e(\mathsf{view})$ denotes the randomness generated by $\mathcal{F}_{\text{tree}}$ corresponding to the e-th epoch:

$$\begin{aligned} & \Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{bad\text{-}event}(\mathsf{view}) = 1 \;\middle|\; \vec{v}(\mathsf{view})\right] \\ = \; & \Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{bad\text{-}event}(\mathsf{view}) = 1 \;\middle|\; \{v_e(\mathsf{view})\}_{e \in E}\right] \end{aligned}$$

We have that any p.p.t. pair Π_{bias}-compliant p.p.t. pair $(\mathcal{A}', \mathcal{Z}')$ such that $\mathcal{A}'$ looks at no more than q hashes for each epoch in any $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}', \mathcal{Z}', \kappa)$ of non-zero support, there exists a Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, the following holds:

$$\begin{aligned} & \Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}', \mathcal{Z}', \kappa) : \mathsf{bad\text{-}event}(\mathsf{view}) = 1\right] \\ \leq \; & \Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{ideal}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{bad\text{-}event}(\mathsf{view}) = 1\right] \cdot q^c \end{aligned}$$

$\mathcal{F}_{\text{bias}}(T_{\text{epoch}}, p)$

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

On receive $\texttt{setpids}(t, \mathsf{pids}_t)$ from $\mathcal{A}$:
assert no tuple of the form $(t, \_)$ has been recorded,
assert pids_t contain only parties that have been spawned
record (t, pids_t)

On receive $\texttt{leader}(\mathsf{nonce}, \mathcal{P}, t)$ from $\mathcal{A}$ or internally:
assert (t, pids_t) has been recorded, and $\mathcal{P} \in \mathsf{pids}_t$
if $\Gamma[\mathsf{nonce}, \mathcal{P}, t]$ has not been set, let $\Gamma[\mathsf{nonce}, \mathcal{P}, t] := \begin{cases} 1 & \text{with probability } p \\ 0 & \text{o.w.} \end{cases}$
return $\Gamma[\mathsf{nonce}, \mathcal{P}, t]$

On receive $\texttt{sethash}(e, \mathsf{nonce}_e)$ from $\mathcal{A}$: record (e, nonce_e)

On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B})$ from $\mathcal{P}$: let $e = \mathsf{epoch}(t)$ where t denotes the current time
assert $\mathsf{chain} \in \mathsf{tree}$, $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$, and a pair (e, nonce_e) was recorded
assert $\texttt{leader}(\mathsf{nonce}_e, \mathcal{P}, t)$ 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 $e = \mathsf{epoch}(t')$:
assert $\mathsf{chain} \in \mathsf{tree}$, and $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$,
assert a pair (e, nonce_e) was recorded, and $\texttt{leader}(\mathsf{nonce}_e, \mathcal{P}^*, t')$ outputs 1
assert $\mathsf{time}(\mathsf{chain}) < t' \leq t$ where t is current time
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 5: Ideal functionality $\mathcal{F}_{\mathbf{bias}}$, allowing adversarially biased hash functions.

Proof. By a straightforward union bound. More specifically, for any p.p.t. pair Π_{bias}-compliant p.p.t. pair $(\mathcal{A}', \mathcal{Z}')$ that attacks Π_{bias}, we can construct a Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ and an execution of Π_{ideal}, where $(\mathcal{A}, \mathcal{Z})$ is allowed to choose the random bits of $\mathcal{F}_{\text{tree}}$ for epochs not in E, i.e., epochs that bad-event does not depend on; however for epochs in E, $\mathcal{F}_{\text{tree}}$ gets to choose the randomness.

$(\mathcal{A}, \mathcal{Z})$ calls $(\mathcal{A}', \mathcal{Z}')$ as a blackbox. Whenever $(\mathcal{A}', \mathcal{Z}')$ makes `leader` queries on a future time step t that is not in any of the epochs in E, $(\mathcal{A}, \mathcal{Z})$ generates the answer at random. Whenever $(\mathcal{A}', \mathcal{Z}')$ calls `sethash` for an epoch in E, $(\mathcal{A}, \mathcal{Z})$ asks $\mathcal{F}_{\text{tree}}$ to use the same random bits as what $(\mathcal{A}', \mathcal{Z}')$ has chosen for the corresponding epoch. Whenever $(\mathcal{A}', \mathcal{Z}')$ makes `leader` queries on a future time step t that is in an epoch in E, and the query contains a nonce that has not been seen, at this moment, $(\mathcal{A}, \mathcal{Z})$ flips a random coin with probability $\frac{1}{q}$ and guesses whether the nonce queried will be chosen by $(\mathcal{A}', \mathcal{Z}')$. If the coin turns up heads, $(\mathcal{A}, \mathcal{Z})$ returns answers to $(\mathcal{A}', \mathcal{Z}')$ consistent with its $\mathcal{F}_{\text{tree}}$. Otherwise, $(\mathcal{A}, \mathcal{Z})$ returns fresh random answers. If the choice later turns out to be

wrong, $(\mathcal{A}, \mathcal{Z})$ simply aborts. If eventually $(\mathcal{A}', \mathcal{Z}')$ calls `sethash` for a challenge nonce it has never queried (for an epoch in E), for any future `leader` query related to this challenge nonce, $(\mathcal{A}, \mathcal{Z})$ returns answers consistent with its own $\mathcal{F}_{\text{tree}}$. Finally, whatever other actions $(\mathcal{A}', \mathcal{Z}')$ outputs, $(\mathcal{A}, \mathcal{Z})$ replays it in the execution of Π_{ideal}. It is not hard to see that $(\mathcal{A}, \mathcal{Z})$ will not abort with $\frac{1}{q^c}$ probability. If $(\mathcal{A}, \mathcal{Z})$ does not abort, then bad-event happens in the execution of Π_{bias} iff it happens in the execution of Π_{ideal}. Due to the definition of conditional independence, the probability of bad-event happening in Π_{ideal} does not depend on the randomness of any other epoch not in E (recall that we had $(\mathcal{A}, \mathcal{Z})$ fix the randomness of $\mathcal{F}_{\text{tree}}$ for any epoch not in E). □

Theorem 4 (Security of Π_{bias})**.** Π_{bias} satisfies T_0-consistency, (T_0, μ)-chain quality, and (T_0, g_0, g_1)-chain growth against any Π_{bias}-compliant $(\mathcal{A}, \mathcal{Z})$ for the same parameters T_0, μ, g_0, g_1 as defined in Theorem 3.

Proof. We now prove the above Theorem 4. Due to Lemma 1, it suffices to show that every bad event we care about bounding is a subset of the union of $\mathsf{poly}(\kappa)$ bad events each of which depends only on a constant number of hashes. Recall that the proof of Theorem 3 essentially follows the proof in the Sleepy paper [9] — as mentioned earlier even though our ideal protocol is different from that of Sleepy [9], the differences are inconsequential and does not alter the induced stochastic process.

In the remainder of the proof, we will revisit Sleepy's proof [9]. Instead of presenting the full proof again from scratch, we focus on pointing out how to express every bad event as the union of polynomially many bad events each of which depends only on a constant number of hashes.

Chain growth lower bound. It is easy to see that the consistent length property still holds with our new Π_{bias}.

We now prove chain growth lower bound. We will show that and every window of medium length, i.e., for every $\frac{T_0}{g_0} \leq t_0 \leq \frac{2T_0}{g_0}$, the chain growth lower bound holds for the parameter t_0 over views sampled from $\mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa)$. It is not hard to see that if the chain growth lower bound holds for every window of medium length $\frac{T_0}{g_0} \leq t_0 \leq \frac{2T_0}{g_0}$, then it also holds for every $T \geq T_0$ and every $t_0 \geq \frac{T}{g_0}$, since every longer window can be broken up into disjoint windows of medium lengths, and we simply have to take a union bound over these windows.

To complete the proof, it suffices to observe the following:

- In Sleepy's chain growth lower bound proof [9] which in turn follows that of Pass et al. [45], for any window $[t, t']$, conditioned on any execution trace view^t up till time t, the minimum chain growth during the window $[t, t']$ is upper bounded by a random variable that depends only on the randomness generated by $\mathcal{F}_{\text{tree}}$ corresponding to the time window $[t, t']$, but does not depend on any other random bits generated by $\mathcal{F}_{\text{tree}}$. Our chain growth lower bound proof then goes to show that conditioned on any view^t, the minimum chain growth during the window $[t, t']$ has to be large. Note that the minimum chain growth during the window $[t, t']$ may depend on random bits before this window, but the proof lower bounds the minimum chain growth during $[t, t']$ with another (implicitly defined) random variable that does not depend on any randomness before t.

- Since $T_{\text{epoch}} \geq \frac{\kappa}{\gamma}$, it holds that every window of medium length (where medium length is as defined above) involves only $O(1)$ number of epochs.

However, as mentioned earlier, since the chain growth lower bound for a medium sized window is lower bounded by a random variable that depends only on $c = O(1)$ hashes, by Lemma 1, chain growth lower bound holds except with negligible probability over $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa)$ for any medium sized window.

Chain quality. We examine the chain quality proof of Sleepy [9]. Below we use the same notations as in Sleepy [9].

If $t \leq 2T_{\text{epoch}}$ is small, then the random variable $\mathbf{Q}(\mathsf{view})[r : r + t]$ depends on only $c = O(1)$ number of hashes. Similarly, the random variable $\mathbf{A}(\mathsf{view})[r : r + t]$ also depends only on $c = O(1)$ number of hashes for $t \leq 2T_{\text{epoch}}$.

Applying Lemma 1, for fixed $[r, r + t]$ such that $t \leq 2T_{\text{epoch}}$ is small, we have that for any $\epsilon > 0$, any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ compliant for Π_{bias},

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{Q}(\mathsf{view})[r : r + t] > (1 + \epsilon)np \cdot t\right] < \mathsf{negl}(npt) \cdot q(\kappa)^c$$

and

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathbf{A}(\mathsf{view})[r : r + t] > (1 + \epsilon)\beta t\right] < \mathsf{negl}(\beta t) \cdot q(\kappa)^c$$

where $q(\kappa)$ denotes the maximum number of hash queries made by $\mathcal{A}$.

Now, taking a union bound, we can upper bound $Q_t(\mathsf{view})$ and $A_t(\mathsf{view})$ for any $t \leq 2T_{\text{epoch}}$ — see the Sleepy work [9] for definitions of Q_t and A_t. Specifically, for any $t \leq 2T_{\text{epoch}}$, any $\epsilon > 0$, any Π_{bias}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function $\mathsf{negl}(\cdot)$ and a polynomial function $\mathsf{poly}(\cdot)$ such that for all κ,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : Q_t(\mathsf{view}) > (1 + \epsilon)np \cdot t\right] < \mathsf{negl}(npt) \cdot \mathsf{poly}(\kappa)$$

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : A_t(\mathsf{view}) > (1 + \epsilon)\beta t\right] < \mathsf{negl}(\beta t) \cdot \mathsf{poly}(\kappa)$$

The above proved bounds for $Q_t(\mathsf{view})$ and $A_t(\mathsf{view})$ for small values of t, assuming $t \leq 2T_{\text{epoch}}$. We now consider large windows. Similarly as before, we can break up large windows into medium-sized windows of lengths $[T_{\text{epoch}}, 2T_{\text{epoch}}]$. By taking a union bound over all windows, we easily get the following fact.

Fact 1. For any $t > 0$, any $\epsilon > 0$, any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ compliant for Π_{bias}, there exists a negligible function $\mathsf{negl}(\cdot)$ and a polynomial $\mathsf{poly}(\cdot)$ such that for all κ,

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : Q_t(\mathsf{view}) > (1 + \epsilon)np \cdot t\right] < \mathsf{negl}(np \cdot \min(t, T_{\text{epoch}})) \cdot \mathsf{poly}(\kappa)$$

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{bias}}}(\mathcal{A}, \mathcal{Z}, \kappa) : A_t(\mathsf{view}) > (1 + \epsilon)\beta t\right] < \mathsf{negl}(\beta \cdot \min(t, T_{\text{epoch}})) \cdot \mathsf{poly}(\kappa)$$

The remainder of the chain quality proof can then be completed following exactly the same recipe as Sleepy [9], plugging in our new Fact 1 to bound the random variables A_t and Q_t.

Consistency. In Sleepy's consistency proof [9], they define a view 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

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

They show that there are $\mathsf{negl}(\kappa)$ fraction of such bad views — note that the same holds in our case, simply plugging in our new Fact 1. Conditioned on views that are not bad, the Sleepy work [9] then argue that for every window of length $\frac{2\kappa}{\beta}$, there has to exist a pivot point except with negligible probability. Note that for any window of length $\frac{2\kappa}{\beta}$, and ignoring views that are not bad, the bad event that a pivot point does not exist within the window depends on randomness that are at most Δ far from boundaries of the window — this means that the bad event depends on $O(1)$ number of hashes given that $T_{\text{epoch}} \geq \frac{\kappa}{\gamma}$. Now by Lemma 1, the bad event that there does not exist a pivot within a window of length $\frac{2\kappa}{\beta}$ is $\mathsf{negl}(\kappa)$. The remainder of the proof follows in the same way as Sleepy [9].

Chain growth upper bound. First, given that we have already proved chain growth lower bound and a bound for the random variable A_t, we can prove a "no long block withholding" lemma in exactly the same way as Sleepy [9], where for a withholding time of ϵt, the failure probability is replaced with $\mathsf{negl}(\min(\beta \cdot \min(t, T_{\text{epoch}}))) \cdot \mathsf{poly}(\kappa)$ instead of $\mathsf{negl}(\beta t)\mathsf{poly}(\kappa)$. For completeness, we state this lemma below since it will be used later in the proof as well.

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 2 (No long block withholding)**.** For every Π_{bias}-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

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

We can now prove chain growth upper bound exactly in the same way as Sleepy [9], relying on our bound on Q_t as well as the new "no long block withholding" lemma. □

9.2 Allowing Posterior Corruption

The "no long block withholding" lemma (see Lemma 2) states that if there is a $\mathsf{chain} \in \mathcal{F}_{\text{punctual}}.\mathsf{tree}$ with a sufficiently old timestamp, then if an honest node never accepted chain as its prefix earlier, it is not going to ever accept chain as its prefix. This implies that even if the adversary $\mathcal{A}$ successfully asks $\mathcal{F}_{\text{punctual}}$ to extend a chain with a sufficiently stale timestamp, this action is useless because $\mathcal{A}$ cannot ever persuade any honest node to ever accept this chain (or any longer chain containing it). In this section, we will augment our ideal functionality to simply reject $\mathcal{A}$'s requests to extend a chain with a sufficiently stale timestamp — see $\mathcal{F}_{\text{punctual}}$ in Figure 6. It is not hard to show that this modification does not affect the security of our ideal protocol.

Protocol Π_{punctual}. We define Π_{punctual} in exactly the same manner as Π_{bias}, except that calls to $\mathcal{F}_{\text{bias}}$ are now replaced with calls to $\mathcal{F}_{\text{punctual}}$.

$\mathcal{F}_{\text{punctual}}(T_{\text{epoch}}, p, W)$

Almost the same as $\mathcal{F}_{\text{bias}}$, except with the following change highlighted in blue:

On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B}, t')$ from corrupt party $\mathcal{P}^*$: let $e = \mathsf{epoch}(t')$:

assert $\mathsf{chain} \in \mathsf{tree}$, and $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$

assert a pair (e, nonce_e) was recorded, and $\texttt{leader}(\mathsf{nonce}_e, \mathcal{P}^*, t')$ outputs 1

assert $\mathsf{time}(\mathsf{chain}) < t' \leq t$ where t is current time

assert $t' \geq t - W$

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

Figure 6: Ideal functionality $\mathcal{F}_{\text{punctual}}$. $\mathcal{F}_{\text{punctual}}$ enforces punctuality, and rejects stale blocks that arrive too late. Blue denotes the difference from $\mathcal{F}_{\text{bias}}$.

Compliant $(\mathcal{A}, \mathcal{Z})$. We now consider a model where $\mathcal{Z}$ can corrupt committee members sufficiently ancient in the past, as long as $\mathcal{Z}$ has not committed these nodes to serve on committees in recent, present, or future epochs. We show that because $\mathcal{F}_{\text{punctual}}$ rejects blocks with stale timestamps anyway, corruption into the past does not allow the adversary to do anything interesting additionally. As a result, we prove that Π_{punctual} is actually secure in this stronger corruption model.

More formally, we say that a p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ is compliant for Π_{punctual} iff the following holds:

- **A-priori commitment of committees and hashes.** Same as Π_{bias}.
- **Spawning and sleeping.** Same as the compliance rules for Π_{bias}.
- **Corruption model.** At time $t \leq t'$, $\mathcal{A}$ is allowed to issue $(\texttt{corrupt}, i, t')$ iff
  - there does not exist $r \geq t' - W$ such that $\mathcal{A}$ has called $\mathcal{F}_{\text{punctual}}.\texttt{setpids}(r, \mathsf{pids}_r)$ where $i \in \mathsf{pids}_r$;

  At time $t \leq t_0 \leq t_1$, $\mathcal{A}$ is allowed to issue $(\texttt{sleep}, i, t_0, t_1)$ if
  - for every $r \in [t_0, t_1]$, $\mathcal{A}$ has not called $\mathcal{F}_{\text{punctual}}.\texttt{setpids}(r, \mathsf{pids}_r)$ such that $i \in \mathsf{pids}_r$.

  In other words, $\mathcal{A}$ can only ask a node i to become corrupt at time t', if $\mathcal{A}$ has not committed i to be on a committee any time at $t' - W$ or later. However, it is possible that after $\mathcal{A}$ asks a node i to become corrupt at a future time, $\mathcal{A}$ can then commit it to some committee. It is also possible for $\mathcal{A}$ to ask a node to be corrupt at a future time if the node served on some very old committee, but has not been committed to any committee since. Note that this "posterior corruption" ability was not allowed for our earlier corruption model (i.e., $(\mathcal{A}, \mathcal{Z})$ compliant for Π_{bias}).

  Further, similar as before, before $\mathcal{A}$ commits to a committee for time t, $\mathcal{A}$ must commit to which set of honest nodes will become asleep at time t.
- **Resilience.** For any time step t, let $\mathsf{cmt}^t(\mathsf{view})$ be the (t, pids_t) committee set that $\mathcal{A}$ sends to

$\mathcal{F}_{\text{punctual}}$ in view, let $r = \min(t + W, |\mathsf{view}|)$, it must hold that

$$\frac{\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})}{\mathsf{corrupt}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})} \geq 1 + \phi$$

where $\mathsf{alert}^s(S, \mathsf{view})$, $\mathsf{honest}^s(S, \mathsf{view})$, and $\mathsf{corrupt}^s(S, \mathsf{view})$ denote those among S are alert, honest, and corrupt at time s respectively.

Notice that our new resilience rule is weaker now, it only requires at any time t, alert committee members who remain honest for W more steps outnumber committee members who become corrupt by $t + W$. Before in Π_{ideal} and Π_{bias}, we essentially required W to be infinity.

- **Number of awake nodes.** Let $\mathsf{cmt}^t(\mathsf{view})$ be the (t, pids_t) committee set that $\mathcal{A}$ sends to $\mathcal{F}_{\text{punctual}}$ in view. It holds that for every $t \leq |\mathsf{view}|$, let $r = \min(t + W, |\mathsf{view}|)$,

$$(\mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) \cap \mathsf{honest}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})) + \mathsf{corrupt}^r(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view}) = n$$

- **Admissible parameters.** Same as in Π_{bias} with the additional requirement that $W \geq \frac{\kappa}{\gamma}$.

Theorem 5 (Security of Π_{punctual})**.** Π_{punctual} satisfies T_0-consistency, (T_0, μ)-chain quality, and (T_0, g_0, g_1)-chain growth against any Π_{punctual}-compliant $(\mathcal{A}, \mathcal{Z})$ for the same parameters T_0, μ, g_0, g_1 as defined in Theorem 3.

Proof. First, it is not hard to see that Π_{punctual} satisfies consistency, μ-chain quality, and (g_0, g_1)-chain growth for a weaker corruption model, i.e., against any any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ compliant w.r.t. Π_{bias} (rather than w.r.t. Π_{punctual}). Recall that a $(\mathcal{A}, \mathcal{Z})$ pair compliant w.r.t. Π_{bias} is not allowed posterior corruption. To see this, consider a compliant execution of Π_{bias}. Due to the "no long block withholding" lemma, a block with an old timestamp will never be first accepted by honest nodes with $1 - \mathsf{negl}(\beta W) = 1 - \mathsf{negl}(\frac{\beta\kappa}{\gamma}) = 1 - \mathsf{negl}(\kappa)$ probability, where W denotes how old the block is.

Below we simply ignore the negligible fraction of bad views where the "no long block withholding" lemma fails. This means that in any good view, if $\mathcal{A}$ tries to call $\mathcal{F}_{\text{bias}}.\mathtt{extend}(\mathsf{chain}, \mathsf{B}, t')$ at time t, where $t' < t - W$ and suppose that $\mathsf{chain}||\mathsf{B}$ is not already in $\mathcal{F}_{\text{bias}}$, then no honest will later ever call $\mathcal{F}_{\text{bias}}.\mathtt{verify}(\mathsf{chain}')$ where $\mathsf{chain}||\mathsf{B} \prec \mathsf{chain}'$. For this reason, it is equivalent if $\mathcal{F}_{\text{bias}}$ simply ignored such adversarial requests to $\mathcal{F}_{\text{bias}}.\mathtt{extend}(\_, \_, t')$ at time t, where $t' < t - W$. And the only difference between Π_{bias} and Π_{punctual} is precisely this: in Π_{punctual}, $\mathcal{F}_{\text{punctual}}$ ignores such adversarial requests to extend a chain with very old timestamps.

Let $\mathsf{allchains}^t(\mathsf{view})$ denote the set that includes an ordered list of the output chains of all nodes alert at time t. To complete the proof, it suffices to show the following lemma.

Lemma 3 (Posterior corruption does not matter)**.** For any Π_{punctual}-compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$, there exists Π_{bias}-compliant p.p.t. $(\mathcal{A}', \mathcal{Z}')$, and a function $\mathsf{somechains}^t(\mathsf{view})$ that selects an appropriate subset of alert nodes' output chains in view and at time t, such that the following distributions are identical:

$$\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \{\mathsf{allchains}^t(\mathsf{view})\}_{t \in [|\mathsf{view}|]} \text{ and}$$
$$\mathsf{view}' \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}', \mathcal{Z}', \kappa) : \{\mathsf{somechains}^t(\mathsf{view}')\}_{t \in [|\mathsf{view}'|]}$$

Proof. $(\mathcal{A}', \mathcal{Z}')$ runs $(\mathcal{A}, \mathcal{Z})$ in a sandbox and intercepts $(\mathcal{A}, \mathcal{Z})$'s communications with outside. At a high level, whenever $(\mathcal{A}, \mathcal{Z})$ wants to corrupt a node, $(\mathcal{A}', \mathcal{Z}')$ will spawn a sybil of the node and corrupt the sybil instead. This will allow $(\mathcal{A}', \mathcal{Z}')$ to respect Π_{bias}'s compliance rules and yet be able to emulate $(\mathcal{A}, \mathcal{Z})$'s attack. As pointed out earlier, if Π_{punctual} is run with such a weaker, Π_{bias}-compliant attacker, then the execution respects all the desired properties including chain growth, chain quality, and consistency. Since $(\mathcal{A}', \mathcal{Z}')$ can emulate attacks by $(\mathcal{A}, \mathcal{Z})$, we can then infer that Π_{punctual} retains these properties in the presence of a stronger, Π_{punctual}-compliant attacker too.

- If at some time $t \le t'$, $\mathcal{A}$ issues $(\texttt{corrupt}, i, t')$: if $\mathcal{A}$ has issued $(\texttt{corrupt}, i, r)$ for $r \le t'$ earlier, this request is ignored. Otherwise, i must not been chosen as a committee member for $[t'-W, \infty]$ since $(\mathcal{A}, \mathcal{Z})$ is Π_{punctual}-compliant. Now $\mathcal{Z}'$ spawns a sybil i^* immediately (if no sybil of i has been spawned earlier) and provides it with i's internal state as input. Let i^* be the sybil of i spawned either at the current time t or earlier. Further, $\mathcal{A}'$ issues $(\texttt{corrupt}, i^*, t')$. $(\mathcal{A}', \mathcal{Z}')$ remembers (or updates) the mapping $\mathsf{sybil}[i] = (i^*, t' - W)$. We also say that $t' - W$ is sybil node i^*'s *effective* time. Intuitively, i^* will act as a defunct copy of i before its effective time; and afterwards i^* will act on behalf of i and then i will effectively become the defunct copy.

- Whenever $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{setpids}(t, \mathsf{pids}_t)$ and this is the first time $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{setpids}(t, \_)$, $(\mathcal{A}', \mathcal{Z}')$ will inspect pids_t. If $i \in \mathsf{pids}_t$ and some tuple $\mathsf{sybil}[i] = (i^*, s)$ has been stored for some $s \le t$, replace node i's occurrence in pids_t with i^*.

  Whenever $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{extend}(\_, \_, \_)$ or $\mathcal{F}_{\text{punctual}}.\texttt{verify}(\_)$ acting as node i at time t, $(\mathcal{A}', \mathcal{Z}')$ finds the stored sybil identity i^* for i — note that such a sybil identity i^* has to exist and i^* has to be already corrupt at t if $\mathcal{A}$ is acting as i at t. Now $\mathcal{A}'$ rewrites the call acting as i^* instead.

  Similarly, whenever $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{leader}(\_, i, t)$, $(\mathcal{A}', \mathcal{Z}')$ makes the following check: if i has a stored sybil identity i^* and moreover i^* is effective at time t, then $\mathcal{A}'$ rewrites the call replacing i with i^*.

- Whenever $\mathcal{A}$ or $\mathcal{Z}$ sends a message to an honest node i, $\mathcal{A}'$ or $\mathcal{Z}'$ sends a duplicate of this message to i's sybil i^* if one exists.

- Whenever a sybil node i^* sends a message to $\mathcal{A}$ or $\mathcal{Z}$ at time t (this means that the sybil node i^* has not become corrupt yet, and is still honest), simply drop the message.

- For every other message sent by $(\mathcal{A}, \mathcal{Z})$, $(\mathcal{A}', \mathcal{Z}')$ directly passes through them.

If $(\mathcal{A}, \mathcal{Z})$ is Π_{punctual}-compliant, and let $(\mathcal{A}', \mathcal{Z}')$ be defined as above, then the following facts must hold: in any $\mathsf{view}' \leftarrow \mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}', \mathcal{Z}', \kappa)$ of non-zero support, for any pair of nodes (i, i^*) where i^* is i's sybil whose effective time is r, i^* is never on any committee for any $t \le r$; and i is never on any committee for any $t > r$. Further, for every sybil i^* in view', $\mathcal{A}'$ has to issue $(\texttt{corrupt}, i^*, \_)$ instructions prior to any $\mathcal{F}_{\text{punctual}}.\texttt{setpids}$ calls that commit i^* to being a committee member.

Claim 1. If $(\mathcal{A}, \mathcal{Z})$ is Π_{punctual}-compliant, then $(\mathcal{A}', \mathcal{Z}')$ must be Π_{bias}-compliant.

Proof. • *Corruption model.* Observe that a Π_{punctual}-compliant $\mathcal{A}$ will never issue $(\texttt{corrupt}, i, t')$ at time $t \le t'$, if node i has already been committed to as a committee member for time $t' - W$ of after. If $\mathcal{A}$ issues $(\texttt{corrupt}, i, t')$ at time $t \le t'$ for some node i that was on a committee before $t' - W$ but has not been committed to as a committee member since, then $(\mathcal{A}', \mathcal{Z}')$ captures this

request and rewrites it with a spawn and a corrupt request for a different node i^*. In this way, it is not hard to see that $\mathcal{A}'$ will never issue $(\texttt{corrupt}, i, t')$ at time $t \le t'$ if node i has ever been committed to as any (past or future) committee member by time t.

- *Resilience and correct parametrization.* For any fixed sequence of random bits $\vec{v}$ consumed by all ITMs in the execution, we consider the pair of execution traces defined by $\vec{v}$, denoted $\mathsf{view}(\vec{v})$ and $\mathsf{view}'(\vec{v})$ in the support of $\mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}, \mathcal{Z}, \kappa)$ and $\mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}', \mathcal{Z}', \kappa)$ respectively.

  It is not hard to see that for every pair $\mathsf{view}(\vec{v})$ and $\mathsf{view}'(\vec{v})$ defined by randomness $\vec{v}$,

  $$\forall t : \mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}'), \mathsf{view}') = \mathsf{alert}^t(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$$

  We now show that

  $$\forall t : \mathsf{corrupt}(\mathsf{cmt}^t(\mathsf{view}'), \mathsf{view}') = \mathsf{corrupt}^{\min(t+W, |\mathsf{view}|)}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$$

  which would also imply

  $$\forall t : \mathsf{honest}(\mathsf{cmt}^t(\mathsf{view}'), \mathsf{view}') = \mathsf{honest}^{\min(t+W, |\mathsf{view}|)}(\mathsf{cmt}^t(\mathsf{view}), \mathsf{view})$$

  $\le$: For every node j that is on the committee of time t in view' and is ever corrupt, j must be the sybil of some node henceforth denoted i. Clearly i must be on the committee for time t in view, therefore it suffices to show that i is corrupt by time $\min(t + W, |\mathsf{view}|)$ in view. Notice that if j on the committee at time t and is ever corrupt in view', let r denote the time that j becomes corrupt. It must be the case that $(\texttt{corrupt}, j, r)$ is issued and afterwards $\mathcal{A}'$ commits j to being a committee member at t. We now show $r \le t + W$. Notice that if $\mathcal{A}'$ issues $(\texttt{corrupt}, j, r)$, then j must be a sybil node whose effective time starts at $r - W$ — before $r - W$ even though j has been spawned, it does not do anything interesting such as being added to committees. Therefore $t \ge r - W$.

  To complete the proof, it is not hard to observe that $(\texttt{corrupt}, i, r)$ must be in view and further in view $\mathcal{A}$ must commit i to being a committee member at t.

  $\ge$: For every i on the committee at t in view and is ever corrupt in $[t, \min(t+W, |\mathsf{view}|)$, it must be the case that in view, $\mathcal{A}$ issues $(\texttt{corrupt}, i, r)$ for some $r \le \min(t + W, |\mathsf{view}|)$ first, and then $\mathcal{A}$ commits i to the t-th committee. Therefore, in the corresponding view', $\mathcal{A}'$ will issue $(\texttt{corrupt}, i^*, r)$ where i^* is the sybil of i in view'. Further, $\mathcal{A}'$ will commit i^* to the t-th committee.

  It is easy to verify that $(\mathcal{A}', \mathcal{Z}')$ satisfies the remaining compliance rules. □

We now consider the most natural selection function $\mathsf{somechains}$ that selects a subset of alert nodes' output chains at every time t given $\mathsf{view}' \leftarrow \mathsf{EXEC}^{\Pi_{\text{punctual}}}(\mathcal{A}', \mathcal{Z}', \kappa)$. Specifically, for each pair (i, i^*) such that both are alert at time t, $\mathsf{somechains}^t$ would select i^* at time t if t is at or after i^*'s effective time. Otherwise, $\mathsf{somechains}^t$ would select i.

Claim 2. Under the aforementioned selection function $\mathsf{somechains}^t$, for every pair $\mathsf{view}(\vec{v})$ and $\mathsf{view}'(\vec{v})$ defined by randomness $\vec{v}$,

$$\{\mathsf{allchains}^t(\mathsf{view})\}_{t \in [|\mathsf{view}|]} = \{\mathsf{somechains}^t(\mathsf{view}')\}_{t \in [|\mathsf{view}'|]}$$

Protocol Π_{hyb}

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

let $\mathsf{pk} := \mathcal{G}_{\text{sign}}.\texttt{gen}()$, output pk to $\mathcal{Z}$, wait to receive *chain*, record *chain* and pk

On receive *chain*′:

assert $|chain'| > |chain|$ and $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{verify}(chain') = 1$

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

Every time step:

- receive input $\texttt{transactions}(\mathsf{txs})$ from $\mathcal{Z}$
- pick random fresh nonce, and let t be the current time
- let $\sigma := \mathcal{G}_{\text{sign}}.\texttt{sign}(\mathsf{pk}, chain[-1].h, \mathsf{txs}, t, \mathsf{nonce})$, $h' := \mathsf{d}(chain[-1].h, \mathsf{txs}, t, \mathsf{nonce}, \mathsf{pk}, \sigma)$
- let $B := (chain[-1].h, \mathsf{txs}, t, \mathsf{nonce}, \mathsf{pk}, \sigma, h')$
- if $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}(chain, B)$ outputs "succ": let $chain := chain||B$ and gossip *chain*
- output $\mathsf{extract}(chain)$ to $\mathcal{Z}$

$\widetilde{\mathcal{F}}_{\text{punctual}}$: Same as $\mathcal{F}_{\text{punctual}}$ except that the $\texttt{extend}(\_, B, t')$ entry point now additionally asserts that $t' = B.\mathsf{time}$

Figure 7: A hybrid protocol carrying real-world blocks.

Proof. To see this, notice that all $(\mathcal{A}', \mathcal{Z}')$ does is renaming nodes — therefore it is not hard to see that the only way that the two can differ is if at some time $t \geq t'$, $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{extend}(\_, \_, t')$ acting as some corrupt node i, such that i is leader for t' in view, but i is no longer effective at time t in view' because its sybil node i^* has taken over.

Now suppose that at $t \geq t'$, $\mathcal{A}$ calls $\mathcal{F}_{\text{punctual}}.\texttt{extend}(\_, \_, t')$ acting as some corrupt node i. There are two cases:

- $t' < t - W$: In this case, $\mathcal{F}_{\text{punctual}}$ would have ignored the request in view since the timestamp t' is too old.
- $t' \geq t - W$: Since we know that in view, i is corrupt at time t, it has to be the case that i^* is already effective at time t'.

□

□

□

9.3 Hybrid Protocol: Ideal Protocol with Real-World Blocks

We are almost ready to show that our real-world protocol emulates the ideal-world one which we know how to analyze. But before that, we have to go through one more intermediate step that

correct the protocol's interfaces to the environment $\mathcal{Z}$ such that the interfaces will type check by the real-world protocol's type definitions.

In this section, we will define a hybrid protocol Π_{hyb} that carries real-world interfaces to the environment $\mathcal{Z}$ — see Figure 7. We will show that all the properties we care about (including consistency, chain growth, and chain quality) hold for Π_{hyb} in exactly the same way they hold for Π_{punctual}. Since Π_{hyb} carries real-world interfaces, we can later show that our real-world protocol $\Pi_{\text{snowwhite}}$ emulates Π_{hyb}; and therefore the real-world protocol satisfies all these properties as well.

Compliant $(\mathcal{A}, \mathcal{Z})$. We say that $(\mathcal{A}, \mathcal{Z})$ Π_{hyb}-compliant if the following holds for any view with non-zero support:

- **Initialization.** At the start of the execution, the following happens. First, $\mathcal{Z}$ can spawn a set of either honest or corrupt nodes. $\mathcal{Z}$ learns the honest nodes' public keys after calling their $\texttt{init}()$ procedure. Next, $\mathcal{A}$ provides the inputs $\{\mathit{genesis}\}$ to all honest nodes. At this point, protocol execution starts.
- **Sleeping.** All sleepers are treated as light sleepers. Upon waking, all pending messages it should have received but did not receive are delivered, plus adversarially inserted messages.
- **Spawning.** When a new, alert node spawns at time t, $(\mathcal{A}, \mathcal{Z})$ must deliver to it an initialization message chain_0 such that $\mathit{chain}_0 \in \mathcal{F}_{\text{punctual}}$ and chain_0 is no shorter than the shortest chain of any alert node at time $t-1$.
- **A-prior commitment.** $\mathcal{A}$ must have called $\mathcal{F}_{\text{punctual}}.\texttt{setpids}(t, \mathsf{pids}_t)$ before t. Similarly, $\mathcal{A}$ must have called $\mathcal{F}_{\text{punctual}}.\texttt{sethash}(e, \mathsf{pids}_e)$ before $\mathsf{start}(e)$.
- **Corruption model, resilience, number of awake nodes, admissible parameters.** Same as in Π_{punctual}.

Theorem 6 (Security of Π_{hyb})**.** Π_{hyb} satisfies T_0-consistency, (T_0, μ)-chain quality, and (T_0, g_0, g_1)-chain growth against any Π_{hyb}-compliant $(\mathcal{A}, \mathcal{Z})$ for the same parameters T_0, μ, g_0, g_1 as defined in Theorem 3.

Proof. Follows in a straightforward manner from the security of Π_{punctual}. □

Remark: Agreement of $\widetilde{\mathcal{F}}_{\text{punctual}}$ timestamp and blockchain timestamp. We note that $\widetilde{\mathcal{F}}_{\text{punctual}}$ checks that the claimed timestamp agrees with the timestamp in the block B when an adversary calls $\texttt{extend}$. Observe also that alert nodes always use truthful timestamps when calling $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}$. Due to this reason, henceforth, for any $\mathit{chain} \in \widetilde{\mathcal{F}}_{\text{punctual}}$, we may use its $\widetilde{\mathcal{F}}_{\text{punctual}}$-timestamp and $\mathit{chain}[-1].\mathsf{time}$ interchangeably.

9.4 Timestamp Freshness Lemma

We prove a useful property about timestamp freshness in any alert node's chain. This will be useful in the next section when we prove that the real-world protocol emulates the hybrid-world. Roughly speaking, the timestamp freshness property says that in any alert node's chain, the timestamp of any block cannot be too early relative to the position of the block in the chain. This will later be

useful in Section 10 in proving that in a simulated execution, certain good events (whose occurrence depends on the existence of large timestamps in alert nodes' chains) happen early enough.

Formally, we define a property $\mathsf{freshtime}^{\ell,r}(\mathsf{view}) = 1$ iff the following holds for view: at any time t, for any node i alert at t and suppose $|chain_i^t(\mathsf{view})| \geq \ell$, then $chain_i^t(\mathsf{view})[-\ell].\mathsf{time} > t - r$.

Claim 3 (Freshness of timestamp in stablized chain)**.** For any Π_{hyb}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every κ, every $\epsilon, \epsilon_0 > 0$, every $\ell > 0$, let $r = \frac{\ell + \epsilon\kappa}{g_0}$ where $g_0 = (1 - \epsilon_0)\gamma$, it holds that

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\mathrm{hyb}}}(\mathcal{A}, \mathcal{Z}, \kappa) : \mathsf{freshtime}^{\ell,r}(\mathsf{view}) = 1\right] \geq 1 - \mathsf{negl}(\kappa)$$

Proof. Below we ignore the negligible fraction of views where bad events such as failure of chain quality or chain growth happen.

If $t < r$, the claim trivially holds. We focus on proving the case where $t \geq r$. In this case, by chain growth, every node alert at time t must have chain length at least $\ell + \epsilon\kappa$. By chain quality, in $chain_i^t[-(\ell + \epsilon\kappa) : -\ell]$ there must be a block mined by node j honest at time t' in view — by definition this means that node j has chain length at least $|chain_i^t| - (\ell + \epsilon\kappa)$ at time t'.

By chain growth, we have that at most $t - t' \leq \frac{\ell + \epsilon\kappa}{g_0} = r$ time has elapsed between t' and t. Since honest blocks contain true timestamps reflecting when the block is mined, there exists a block in $B \in chain_i^t[-\ell + \epsilon\kappa : -\ell]$ such that $B.\mathsf{time} \geq t - r$. The rest of the proof is obvious by observing that timestamps must be strictly increasing in $chain_i^t$ assuming i is alert at t.

□

Remark. Henceforth, whenever we apply Claim 3 in our proofs, we will assume that $g_0 = (1-\epsilon_0)\gamma$ for an ϵ_0 that is appropriately small — it is not hard to identify such a ϵ_0 for all proofs that rely on Claim 3 henceforth — we therefore often omit spelling out g_0 as $g_0 = (1 - \epsilon_0)\gamma$ for simplity.

10 Proofs: Real World is as Secure as the Hybrid World

10.1 Theorem Statement

Theorem 7 ($\Pi_{\mathrm{snowwhite}}$ emulates Π_{hyb} while preserving compliance)**.** For any real-world p.p.t. adversary $\mathcal{A}$ for $\Pi_{\mathrm{snowwhite}}$, there exists a p.p.t. adversary (also called the simulator) $\mathcal{S}$ for Π_{hyb}, such that for any p.p.t. $\mathcal{Z}$ satisfying the condition that $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\mathrm{snowwhite}}$-compliant, we have that

- $(\mathcal{S}, \mathcal{Z})$ is Π_{hyb}-compliant; and
- $\mathsf{EXEC}^{\Pi_{\mathrm{snowwhite}}}(\mathcal{A}, \mathcal{Z}, \kappa) \stackrel{c}{\equiv} \mathsf{EXEC}^{\Pi_{\mathrm{hyb}}}(\mathcal{S}, \mathcal{Z}, \kappa)$

In the above, both $\Pi_{\mathrm{snowwhite}}$ and Π_{hyb} (and specifically $\widetilde{\mathcal{F}}_{\mathrm{punctual}}$) are instantiated with "matching" parameters. More specifically, the following must be hold:

- Both $\Pi_{\mathrm{snowwhite}}$ and Π_{hyb} (or more specifically $\widetilde{\mathcal{F}}_{\mathrm{punctual}}$) are instantiated with the same (p, T_{epoch}).
- Suppose W is the posterior corruption parameter respected by the real-world adversary $\mathcal{A}$, it holds that in protocol Π_{hyb}, $\widetilde{\mathcal{F}}_{\mathrm{punctual}}$ is instantiated with the parameter W.

In this section, we prove the above Theorem 7.

AddtoTree($chain$)

- Computes ℓ such that $chain[:\ell-1]$ is the longest prefix of $chain$ such that $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{verify}(chain[:\ell-1]) = 1$.
- If any block in $chain[\ell:]$ is signed by a public key that does not correspond to a corrupt node, abort outputting signature-failure.
- Else, for each $\ell' \in [\ell, |chain|]$: call $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}(chain[:\ell'-1], chain[\ell'], chain[\ell'].\mathsf{time})$ acting as the corrupt node that corresponds to $chain[\ell'].\mathsf{pk}$ at time $chain[\ell'].\mathsf{time}$.
- If $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{verify}(chain)$ does not return true at this point, abort outputting extend-failure.

Figure 8: AddtoTree subroutine internally called by $\mathcal{S}$.

10.2 Simulator Construction

We first describe the construction of $\mathcal{S}$, which interacts in a blackbox manner with $\mathcal{A}$.

Intended invariants by construction. By construction, the simulator is meant to maintain the following invariants:

1. $\mathcal{S}$ keeps performing internal checks in every time step, and aborts whenever $(\mathcal{S}, \mathcal{Z})$ is about to violate Π_{hyb}'s compliance rules. In this way, as long as the simulation has not aborted, $(\mathcal{S}, \mathcal{Z})$ is by construction Π_{hyb}-compliant, and therefore we can use the security properties of Π_{hyb} to reason about the simulated execution. Later, we will also show aborts will not happen except with negligible probability.
2. $\mathcal{S}$ always makes sure that any *chain* sent to alert nodes that would have been accepted by alert nodes in the real world must be in $\widetilde{\mathcal{F}}_{\text{punctual}}$. In other words, if $\mathcal{A}$ tries to send a *chain* to an alert node i, $\mathcal{S}$ will first emulate node i's $\Pi_{\text{snowwhite}}$-behavior to see if node i might have accepted *chain* in the real-world protocol $\Pi_{\text{snowwhite}}$. If so, then $\mathcal{S}$ will make sure that *chain* is indeed in $\widetilde{\mathcal{F}}_{\text{punctual}}$ before forwarding *chain* to i. This may mean that $\mathcal{S}$ will need to call $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}$ to insert new chains before forwarding *chain* to i.

   In this way, an essential step in showing the indistinguishability of the real-world and simulated executions is to argue that $\mathcal{S}$ can always succeed in adding a *chain* to $\widetilde{\mathcal{F}}_{\text{punctual}}$ if *chain* would have been accepted in the real-world by alert nodes (see Section 10.4).

Simulator description. For convenience, we introduce the syntactic sugar

$$\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(e, \mathsf{pids}_e)$$

to allow $\mathcal{S}$ to set the committee for each time step t that is in epoch e, all with the same committee pids_e. Clearly $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(e, \mathsf{pids}_e)$ can be implemented by multiple calls to $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(t, \mathsf{pids}_t)$.

Our simulator $\mathcal{S}$ relies on a parameter $\epsilon^* > 0$. Our proof will hold as long as ϵ^* is sufficiently small, e.g., less than $\frac{1}{4}$. Henceforth without loss of generality, the reader can assume that $\epsilon^* = \frac{1}{4}$. The security failure probability will be related to ϵ^*, e.g., in the form of $\mathsf{negl}(\mathsf{poly}(\epsilon^*, \kappa))$.

- Whenever a party calls $\mathcal{G}_{\text{sign}}.\texttt{gen}$, the adversary is notified of the pair $(\mathcal{P}, \mathsf{pk})$. It stores this party identifier and public key mapping. If an honest node's public key ever collides with a key that is already stored in the mapping, abort outputting duplicate-key-failure.

- Since $\mathcal{S}$ sees all messages sent to and from honest nodes, $\mathcal{S}$ can simulate the internal longest *chain* kept by all honest nodes in the most natural manner. Henceforth we assume that for any t, any i honest at time t, $\mathcal{S}$ knows $chain_i^t$.

- Whenever $\mathcal{A}$ asks hash queries: if this query has been seen before, $\mathcal{S}$ returns the same answer as before.

  Else, suppose that the query is of the form $\mathsf{H}^{\mathsf{nonce}}(\mathsf{pk}, t)$, let $e = \mathsf{epoch}(t)$: if $\mathcal{S}$ has not yet committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$ the e-th committee, then $\mathcal{S}$ simply generates a random number of appropriate length, and returns it to $\mathcal{A}$. Else if $\mathcal{S}$ has committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$ the e-th committee, then $\mathcal{S}$ generates a random number that agrees with $\widetilde{\mathcal{F}}_{\text{punctual}}$. Note that due to our simulator description later, if $\mathcal{S}$ has committed to the e-th committee, then there is a spawned party (either honest or corrupt) for each public keys in the e-th committee, and the simulator $\mathcal{S}$ must have this mapping stored.

  More specifically, $\mathcal{S}$ first looks up the party identifier $\mathcal{P}$ that corresponds to pk. If such a party identifier is not found, $\mathcal{S}$ samples a random number of appropriate length and returns it to $\mathcal{A}$. Else, $\mathcal{S}$ calls $b = \widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{leader}(\mathsf{nonce}, \mathcal{P}, t)$. If $b = 1$, it rejection samples an h until $h < D_p$, and then returns h. Else, it it rejection samples an h until $h \geq D_p$, and then returns h.

- Whenever $\mathcal{A}$ sends a protocol message *chain* to an honest party i, the simulator $\mathcal{S}$ checks the validity of *chain* simulating node i running the real-world protocol's checks — here the hash function H is implemented with $\mathcal{S}$'s own table. Specifically, $\mathcal{S}$ answers its own H queries in the same way that it answers $\mathcal{A}$'s H queries. If these real-world checks pass, the simulator $\mathcal{S}$ calls $\mathsf{AddtoTree}(chain)$ as described in Figure 8. If the call does not abort outputting signature-failure or extend-failure, $\mathcal{S}$ forwards *chain* to node i.

  If the real-world checks fail, $\mathcal{S}$ drops the message *chain* and does not forward *chain* to node i — note that this may cause $\mathcal{S}$ to violate the Δ network delivery requirement. Therefore, as we describe later, $\mathcal{S}$ will perform internal consistency checks, and if it ever violates the Δ network delivery requirement, it simply aborts outputting Δ-failure.

- Whenever $\mathcal{A}$ sends an initialization message $\{chain_i\}_{i \in L}$ to an honest party i that has just spawned or waken up from deep sleep: the simulator $\mathcal{S}$ runs the real-world algorithm to compute an initial *chain* — as before, here the hash H is implemented by $\mathcal{S}$ itself. $\mathcal{S}$ then calls $\mathsf{AddtoTree}(chain)$, and if the call did not abort with either signature-failure or extend-failure, $\mathcal{S}$ sends *chain* to node i.

- At the beginning of every time step t, the simulator $\mathcal{S}$ performs the following verification. First, if $\mathcal{S}$ has not called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(e, \_)$ or $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{sethash}(e, \_)$ where $e := \mathsf{epoch}(t)$, abort outputting late-failure. The simulator additionally checks the resilience, number of awake nodes, and admissible parameter conditions for time step t, and if the checks fail, abort outputting param-failure.

  For each honest node i, the simulator finds $chain_i^t$ henceforth denoted $chain_i$ for short. Recall that the simulator keeps track of the longest chain each honest node has.

– *Consistency checks.* For every node i alert at time t, $\mathcal{S}$ computes $\mathsf{pks}_i := \mathsf{elect\_cmt}^t(\mathit{chain}_i)$. If both i and j are alert at t but $\mathsf{pks}_i \neq \mathsf{pks}_j$ abort outputting consistency-failure. If pks_i does not agree with the committee at t which $\mathcal{S}$ previously committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$, abort outputting consistency-failure. Here consistency is defined by $\mathcal{S}$'s internal public key to party identifier mapping.

For every node i alert at time t, $\mathcal{S}$ computes $\mathsf{nonce}_i := \mathsf{elect\_h}^t(\mathit{chain}_i)$. If nonce_i does not agree with what $\mathcal{S}$ previously committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$, abort outputting consistency-failure.

– *Choose next committee or hash when necessary.* Henceforth let $\mathit{chain} := \mathit{chain}_i^t$, note that $\mathcal{S}$ checks all alert nodes' chains to see if it needs to choose the next committee or hash.

If $|\mathit{chain}| \geq \epsilon^*\kappa + 2$, $\mathcal{S}$ will check to see if it needs to commit to the next epoch's committee and/or hash. Let $\mathsf{start}(e) := e \cdot T_{\text{epoch}}$ denote the start of epoch $e \in \mathbb{N}$.

For any $e \in \mathbb{N}$, if there exists consecutive blocks $(B_0, B_1) \in \mathit{chain}[: -\epsilon^*\kappa]$ such that $B_1.\mathsf{time} + 2\omega > \mathsf{start}(e)$ but $B_0.\mathsf{time} + 2\omega \leq \mathsf{start}(e)$, $\mathcal{S}$ calls

$$\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{setpids}(e, \mathsf{extractpids}(\mathit{chain}_0)) \text{ where } \mathit{chain}_0 := \mathit{chain}[: \mathsf{index}(B_0)]$$

if it has not already called $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{setpids}(e, \_)$ earlier. Here $\mathsf{extractpids}(\mathit{chain}_0)$ first calls $\mathsf{pks}^* := \mathsf{extractpks}(\mathit{chain}_0)$ and then maps the public keys to their party identifiers in the following way:

1. If $\mathcal{S}$ has recorded $(\mathcal{P}, \mathsf{pk})$ then map pk to $\mathcal{P}$. Recall that $\mathcal{S}$ should have recorded such a mapping for all honest nodes' public keys.
2. If $\mathcal{S}$ has not recorded a mapping for pk, spawn a corrupt node with party identifier j, and map pk to j. Recall that our execution model allows $\mathcal{S}$ to spawn corrupt nodes without $\mathcal{Z}$'s knowledge. Further $\mathcal{S}$ stores the mapping from the party identifier j to the public key pk.

Similarly, for any $e \in \mathbb{N}$, if there exists consecutive blocks $(B_0, B_1) \prec \mathit{chain}[: -\epsilon^*\kappa]$ such that $B_1.\mathsf{time}+\omega > \mathsf{start}(e)$ but $B_0.\mathsf{time}+\omega \leq \mathsf{start}(e)$, $\mathcal{S}$ calls $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{sethash}(e, \mathsf{extractnonce}(\mathit{chain}_0))$ where $\mathit{chain}_0 := \mathit{chain}[: \mathsf{index}(B_0)]$ if it has not already called $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{sethash}(e, \_)$ earlier.

- At the end of every time step t, $\mathcal{S}$ performs a network delivery check. If it has ever received a message from some alert node by $t - \Delta$, but the message did not get delivered to any node alert at t, then $\mathcal{S}$ aborts outputting Δ-failure.

- Whenever $\mathcal{S}$ calls $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{sethash}(e, \mathsf{nonce}_e)$, if $\mathcal{A}$ (or any internal call) has made a hash query of the form $\mathsf{H}^{\mathsf{nonce}_e}(\_, \_)$ before $\mathcal{S}$ called $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathtt{setpids}(e, \_)$, abort outputting predict-failure.

- If $\mathcal{A}$ ever issues a $(\mathtt{corrupt}, i, t')$ instruction to $\mathcal{Z}$ at time $t \leq t'$, the simulator $\mathcal{S}$ checks that it has not committed node i to any committee at time $t' - W$ or later. If the check fails, $\mathcal{S}$ aborts outputting corruption-failure; else, pass through the instruction to $\mathcal{Z}$.

  If $\mathcal{A}$ ever issues a $(\mathtt{sleep}, i, t_0, t_1)$ instruction to $\mathcal{Z}$ at time $t \leq t_0 \leq t_1$, the simulator $\mathcal{S}$ checks that it has not committed node i to any committee between $[t_0, t_1]$. If the check fails, $\mathcal{S}$ aborts outputting corruption-failure; else, pass through the instruction to $\mathcal{Z}$.

- $\mathcal{S}$ directly passes through all other messages between $\mathcal{A}$ and $\mathcal{G}_{\text{sign}}$. Similarly, $\mathcal{S}$ directly passes through all other messages between $\mathcal{A}$ and $\mathcal{Z}$.

- At the protocol start, whenever $\mathcal{A}$ sends pks_0 to a spawning node as part of the initialization message, $\mathcal{S}$ registers the first epoch's committee with $\widetilde{\mathcal{F}}_{\text{punctual}}$. Before doing so, if there is any public key in pks_0 that does not have a party identifier mapping, $\mathcal{S}$ spawns a corrupt node that corresponds to this public key, and stores the mapping. $\mathcal{S}$ also registers nonce_0 with $\widetilde{\mathcal{F}}_{\text{punctual}}$ as the nonce for the first epoch at protocol start.
- If $\mathcal{S}$ ever observes that two honest nodes have different chains with the same block hash, abort outputting hash-failure.

We can now immediately state a few simple facts.

Fact 2 (Compliant execution)**.** In the above simulation, for any p.p.t. $(\mathcal{A}, \mathcal{Z})$ $\Pi_{\text{snowwhite}}$-compliant, the pair $(\mathcal{S}^{\mathcal{A}}, \mathcal{Z})$ is Π_{hyb}-compliant.

Proof. By definition, notice that the construction of the simulator $\mathcal{S}$ performed internal checks and always aborts outputting failure before it ever has a chance of being non-compliant. □

Fact 3 (No hash collision)**.** $\mathcal{S}$ does not abort with hash-failure except with negligible probability.

Proof. Straightforward due to the collision resistance of the digest function d. □

Fact 4 (No honest key collision)**.** $\mathcal{S}$ does not abort with duplicate-key-failure except with negligible probability.

Proof. Straightforward due to the security of the signature scheme. □

Due to the above facts, henceforth we will ignore the negligible fraction of views that have hash or honest key collisions.

10.3 Consistency and Compliance of the Simulated Execution

In this section, we focus on showing that if $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{snowwhite}}$-compliant, then the simulated execution has nice properties regarding consistency and the relative timing of events.

Lemma 4 (Consistency and a-priori commitment)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{late\text{-}failure} \text{ or } \mathsf{consistency\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. We consider any view for $\mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}, \mathcal{Z}, \kappa)$ where none of the bad events related to chain quality, growth, and consistency happen.

We prove the lemma for hashes, and the argument for the committee goes in the same way. It suffices to prove the following: For any $e \in \mathbb{N}$ in view, let $t = \mathsf{start}(e)$, it holds that for every node i honest at time $t-1$, there exists

1. an honest block $B \in \mathit{chain}_i^{t-1}[: -\epsilon^*\kappa]$ with $B.\mathsf{time} > t - \omega$; and
2. an honest block $B' \in \mathit{chain}_i^{t-1}[: -\epsilon^*\kappa]$ with $B'.\mathsf{time} \leq t - \omega$.

where $chain_i^{t-1}(\mathsf{view})$ denotes the internal chain maintained by node i at time $t-1$ in view. We often write $chain_i^{t-1}$ in place of $chain_i^{t-1}(\mathsf{view})$ without risk of ambiguity.

Notice that due to the definition of $\mathcal{S}$ which runs the real-world checks on any *chain* received from $\mathcal{A}$ before forwarding them onto honest nodes, and due to the definition of Π_{hyb}, $chain_i^{t-1}$ must have strictly increasing timestamps. Therefore if the above conditions regarding the existence of B and B' in $chain_i^{t-1}$ are satisfied then

- $\mathcal{S}$ will have called $\widetilde{\mathcal{F}}_{\mathrm{punctual}}.\mathtt{sethash}(e, \_)$ before time t;
- Due to consistency and definition of $\mathcal{S}$, $\mathcal{S}$ will not abort with $\mathsf{consistency\text{-}failure}$ due to disagreement on the hash for epoch e in view.

Notice that the existence of the block B' trivially holds since the genesis block is defined to have a timestamp of 0. The existence of a block B is proved in Claim 3 given that $\omega \geq \frac{2\kappa}{\gamma} + \widetilde{\Delta}$. This completes our proof. □

Lemma 5 (Unpredictability of future hashes)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\mathrm{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{predict\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Given a $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\mathrm{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z})$ where none of the bad events related to chain quality, growth, consistency happen. Let $e \in \mathbb{N}$ denote an epoch in view such that $\mathcal{S}$ has called $\widetilde{\mathcal{F}}_{\mathrm{punctual}}.\mathtt{sethash}(e, \_)$. Note that by definition, $\mathcal{S}$ must have called $\widetilde{\mathcal{F}}_{\mathrm{punctual}}.\mathtt{setpids}(e, \_)$ too. Let $T = \mathsf{start}(e)$.

- Henceforth in this proof we shall adopt small enough constants ϵ_0 and ϵ'. The proof holds for any constant small enough. For example, one may assume $\epsilon_0 = \epsilon' = 1/8$.
- Let $chain_i^t(\mathsf{view})$ be the chain that triggered $\mathcal{S}$ to call $\widetilde{\mathcal{F}}_{\mathrm{punctual}}.\mathtt{sethash}(e, \_)$, where i is a node honest at time t. Let (B^*_{-1}, B^*) be two consecutive blocks in $chain_i^t[: -\epsilon^*\kappa]$, where $B^*.\mathsf{time} > T - \omega$ and $B^*_{-1} \leq T - \omega$.

  Let B_l denote the last honest block to the left of B^* in $chain_i^t$ that is not genesis. We will first pretend that such a block B_l exists, and later we will prove that indeed it does. Let $B_r \in chain_i^t[\mathsf{index}(B^*) :]$ be the first honest block to the right of B^* — such a block must exist due to chain quality, and that there are at least $\epsilon^*\kappa$ blocks to the right of B^*. By chain quality, $\mathsf{index}(B_r) - \mathsf{index}(B^*) \leq \epsilon_0\kappa$ in $chain_i^t$, and $\mathsf{index}(B^*) - \mathsf{index}(B_l) \leq \epsilon_0\kappa$ in $chain_i^t$. Therefore, $\mathsf{index}(B_r) - \mathsf{index}(B_l) \leq 2\epsilon_0\kappa$. Further, since B_l is honest, $T_1 := B_l.\mathsf{time}$ is the time when an honest node first mines B_l. Similarly, $T_2 := B_r.\mathsf{time}$ is the time when an honest node first mines B_r. By chain growth, $T_2 - T_1 \leq \frac{2\epsilon_0\kappa}{g_0}$. By definition, we also know that $T_1 \leq B^*.\mathsf{time} \leq T_2$, and therefore $B^*.\mathsf{time} - T_1 \leq \frac{2\epsilon_0\kappa}{g_0}$, i.e., $T_1 \geq B^*.\mathsf{time} - \frac{2\epsilon_0\kappa}{g_0} > T - \omega - \frac{2\epsilon_0\kappa}{g_0}$.

  For the above bound on T_1 to hold, it remains to show that such a B_l exists. To show this, we prove that there is a constant $\epsilon' > 0$ such that $\mathsf{index}(B^*)$ (w.r.t. $chain_i^t$) is greater than $\epsilon'\kappa$, since then B_l must exist by chain quality. Suppose for the sake of contradiction that $\mathsf{index}(B^*) < \epsilon'\kappa$. As before, let B_r be the first honest block to the right of B^* in $chain_i^t$. Such a B_r must exist within at most $\epsilon_0\kappa$ blocks from B^*. Therefore $\mathsf{index}(B_r) < (\epsilon' + \epsilon_0)\kappa$. Since B_r is an honest block, $B_r.\mathsf{time}$ denotes the time B_r was first mined by an honest node. By chain growth,

$B_r.\mathsf{time} \leq \frac{(\epsilon'+\epsilon_0)\kappa}{g_0}$. Since $T \geq T_{\text{epoch}} \geq 3\omega \geq \frac{3\kappa}{g_0}$, it holds that for sufficiently small constants $\epsilon', \epsilon_0 > 0$,

$$B^*.\mathsf{time} \leq B_r.\mathsf{time} \leq \frac{(\epsilon' + \epsilon_0)\kappa}{g_0} \leq T - \omega$$

Therefore we reach a contradiction.

- Let T_0 be the time in view that $\mathcal{S}$ called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(e, \_)$. We now show that $T_0 \leq T - 1.5\omega$. It is not hard to see that $\mathcal{S}$ will definitely have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(e, \_)$ by time t, if there is a node i honest at $t-1$, such that $\mathit{chain}_i^{t-1}[: -\epsilon^*\kappa]$ contains a block whose timestamp is greater than $T - 2\omega$ where $T = \mathsf{start}(e)$.

  By Claim 3, let $r = T - 1.5\omega \geq 1.5\omega$, for every node i honest at $r-1$, $\mathit{chain}_i^{r-1}[: -\epsilon^*\kappa]$ must contain a block whose timestamp is greater than $r - \frac{\omega}{2} = T - 2\omega$ — notice that this relies on our choice of ϵ^* being sufficiently small. This suffices for showing that $T_0 \leq T - 1.5\omega$.

Given our $\mathsf{extractnonce}$ definition, it suffices to prove that $T_1 > T_0$. Recall that T_1 is the time the honest block B_l is mined, and the block B_l contains a random string that is unpredictable any time before T_1. We now show that indeed $T_1 > T_0$. Observe that since $T \geq T_{\text{epoch}}$ and $T_{\text{epoch}} \geq 3\omega$, and that for small enough ϵ_0, $\frac{2\epsilon_0\kappa}{g_0} < 0.5\omega$, it clearly holds that $T_1 > T_0$.

□

Lemma 6 (Simulator respects Δ-network delivery)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \Delta\text{-failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Fix some $\mathsf{view} \leftarrow \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa)$ where none of the bad events related to chain growth, chain quality, and consistency happen. Suppose that a node i alert at t sends a chain_i^t at time t in view. Due to the $\Pi_{\text{snowwhite}}$-compliance of $(\mathcal{A}, \mathcal{Z})$, for every node that is alert at $t + \Delta$, $\mathcal{A}$ will ask $\mathcal{S}$ to deliver chain_i^t to node j at some $t_j \in [t, t + \Delta]$.

Clearly, $\mathit{chain}_i^t \in \widetilde{\mathcal{F}}_{\text{punctual}}$ starting at time t in view. Due to consistency, for any node j that is alert at sometime $s \in [t, t + \Delta]$, including ones that might have just waken up from a light sleep, it must hold that $\mathit{chain}_j^s[: -\kappa_0] \prec \mathit{chain}_i^t$. Therefore, j's real-world checks will not cause j to reject chain_i^t had j received chain_i^t at any $s \in [t, t + \Delta]$. As a result, $\mathcal{S}$ will not drop this message chain_i^t. □

Lemma 7 (Compliant corruptions)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{corruption\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Suppose that $\mathcal{A}$ issues $(\texttt{corrupt}, i, t')$ at some time t. Since $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{snowwhite}}$-compliant, it must hold that $t' - t > \tau$. We now show that by time t, $\mathcal{S}$ cannot have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}$ for any time during $[t' - W, \infty]$.

Notice that the only way $\mathcal{S}$ could have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(s, \_)$ by time t is if there is a node i honest at t such that $\mathit{chain}_i^t[: -\epsilon^*\kappa]$ contains a block B such that $B.\mathsf{time} > \mathsf{rnddown}(s) - 2\omega > s - T_{\text{epoch}} - 2\omega$. This means that $\mathcal{S}$ can only have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(s, \_)$ by time t if $t \geq s - T_{\text{epoch}} - 2\omega$. If $s \geq t' - W$, this means that $t \geq t' - W - T_{\text{epoch}} - 2\omega$. However, we also know

that $t < t' - \tau$. Since $\tau > W + T_{\text{epoch}} + 2\omega$, it cannot be the case that $t \geq t' - W - T_{\text{epoch}} - 2\omega$; and therefore $\mathcal{S}$ cannot have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}(s, \_)$ for any time $s \in [t' - W, \infty]$.

Similarly, suppose that $\mathcal{A}$ issues $(\texttt{sleep}, i, t')$ at some time t. Since $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{snowwhite}}$-compliant, it must hold that $t' - t > \tau$. we can similarly show that $\mathcal{S}$ cannot have called $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{setpids}$ for any time during $[t', \infty]$. □

Lemma 8 (Parameter preservation)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{param\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Straightforward to verify. □

10.4 All Real-World Valid Chains are in $\widetilde{\mathcal{F}}_{\text{punctual}}$

In this section, we show that for any chain that would have been accepted by an honest node by the real-world verification algorithm, $\mathcal{S}$ must succeed in adding it to $\widetilde{\mathcal{F}}_{\text{punctual}}$ if the chain does not exist in $\widetilde{\mathcal{F}}_{\text{punctual}}$ already.

Lemma 9 (Unforgeability of signatures)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{signature\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Straightforward reduction to the security of the signature scheme. Conditioned on no hash collision, if there is ever $\mathsf{signature\text{-}failure}$, the adversary $\mathcal{A}$ must have forged a signature on a new message that the simulator $\mathcal{S}$ has not sent $\mathcal{A}$. We can easily leverage such an adversary $\mathcal{A}$ to build a reduction to break signature security. Note also that $\mathcal{G}_{\text{sign}}$ is a global functionality, however, the environment $\mathcal{Z}$ cannot query $\mathcal{G}_{\text{sign}}$ for signatures pertaining to the challenge session identifier — this is important for the reduction to work. □

Simulation valid chains. Given a view of the simulated execution, we say that a *chain* is *simulation valid* w.r.t. time t in view if it is valid as defined in the real-world protocol, but where the hash function H is replaced by hash queries to the simulator $\mathcal{S}$. The simulator $\mathcal{S}$ answers these hash queries in the same way it answers $\mathcal{A}$'s hash queries.

Sufficiently long honest prefix. Given a view of the simulated execution, we say that a simulation valid *chain* (w.r.t. t) has a sufficiently long honest prefix at time t in view, iff

> There exists a prefix $chain_0 \prec chain$ such that $chain_0[-1].\mathsf{time} > t - \omega$, and moreover, there exists $s \leq t$ and a node i alert at time s, such that $chain_0 \prec chain_i^s[: -\epsilon^* \kappa]$.

Claim 4. Let *chain* be simulation valid in view at time t, and suppose that *chain* has a sufficiently long honest prefix at t. It holds that the following two ways for determining whether a public key pk is a leader in any time $r \leq t$ are equivalent in the simulated execution:

1. Using the real-world $\mathsf{eligible}^r(chain, \mathsf{pk})$ function where H is implemented by $\mathcal{S}$; and

2. Calling $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{leader}(\mathsf{nonce}^r, \mathcal{P}, r)$ where nonce^r denotes the nonce previously chosen by $\mathcal{S}$ for time step r, and $\mathcal{P}$ is the party identifier corresponding to pk (as determined by $\mathcal{S}$'s stored mapping) — if no such mapping is found, then $\mathcal{P}$ is simply $\perp$.

Proof. Recall that if the simulation does not abort, there is no duplicate-key-failure. If $r < T_{\text{epoch}}$, pks_0 will be selected as the committee by the real-world algorithm, and recall that $\mathcal{S}$ has registered with $\widetilde{\mathcal{F}}_{\text{punctual}}$ the party identifiers for pks_0 as the initial committee.

The more interesting case is when $r \geq T_{\text{epoch}} \geq 3\omega$. In this case, since $chain_0[-1].\mathsf{time} > t - \omega$, the prefix that determines the committee or hash for any $r \leq t$ must be contained in $chain_0$. Due to consistency and the definition of $\mathcal{S}$, it holds that the committee for any $r \leq t$ determined by the real-world algorithm based on *chain* must agree with what $\mathcal{S}$ committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$. Similarly, $\mathcal{S}$ must have committed to $\widetilde{\mathcal{F}}_{\text{punctual}}$ the same nonce for each $r \leq t$ as what the real-world algorithm would output as the nonce for each $r \leq t$ based on *chain*.

Further, due to no predict-failure, consistency, and the way $\mathcal{S}$ answers H queries, it holds that using H to elect leaders agrees with the random coins used by $\widetilde{\mathcal{F}}_{\text{punctual}}$ for electing leaders. □

Claim 5. Let $(\mathcal{A}, \mathcal{Z})$ be $\Pi_{\text{snowwhite}}$-compliant. Given any view of $\mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa)$, let *chain* be a simulation valid chain w.r.t. time t in view, and suppose that *chain* has a sufficiently long honest prefix at t in view. Then if $\mathcal{S}^{\mathcal{A}}$ calls $\mathsf{AddtoTree}(chain)$ at time t in view, the call must succeed.

Proof. First, since $chain_0[-1].\mathsf{time} > t-\omega$, and $W > \omega$, and since timestamps must strictly increase for a simulation valid chain, it is clear that none of the adversarial blocks at the end will be rejected by $\widetilde{\mathcal{F}}_{\text{punctual}}$ due to staleness. The rest of proof follows in a straightforward manner due to Claim 4 and no signature-failure.

□

Claim 6. In the simulated execution, if $\mathcal{S}$ sends a *chain* to an alert node, then *chain* must be simulation valid at t and have a sufficiently long prefix at t.

Proof. We now prove the above lemma. Recall that $\mathcal{S}$ simulates the real-world verification algorithm for node i, and only forwards a *chain* to alert node i if the real-world checks succeed. It suffices to prove that if the real-world checks pass, then the chain has a sufficiently long prefix.

There are three possible scenarios, and we analyze them one by one.

Case 1: $\mathcal{A}$ sends *chain* to a node i that has been alert. This case is very similar to Case 2, except that the $\widetilde{\Delta}$ in Case 2 is now replaced with Δ. By our parameter admissible rules, it is not hard to see that $\Delta \leq \frac{1}{2\gamma}$. The rest of the proof follows in the same way as Case 2.

Case 2: $\mathcal{A}$ sends *chain* to node i that has just waken up after a light sleep. Suppose that at time t, $\mathcal{A}$ wants to send *chain* to honest node i who has waken up at time t after a short sleep, and let s denote the most recent time node i went to sleep before t. By $\Pi_{\text{snowwhite}}$-compliance, we know that $t - s \leq \widetilde{\Delta}$.

We know that it must be the case $chain_i^s[:-\kappa_0] \prec chain$ for *chain* to be accepted by node i's real-world checks. Also observe that by Claim 3, $chain_i^s[-\kappa_0].\mathsf{time} > s - \frac{\kappa}{g_0} \geq t - \widetilde{\Delta} - \frac{\kappa}{g_0} \geq t - \omega$.

Case 3: $\mathcal{A}$ sends a newly spawned node i an initialization message. Fix any view of non-zero support in the execution $\mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa)$. All of the following statements are with respect to this view. Given a set $S_L := \{chain_i\}_{i \in L}$, we say that a *chain* is real-world admissible w.r.t. S_L if 1) *chain* is simulation valid; 2) $chain \prec chain_i$ for some $i \in L$; and 3) let $chain'$ be the longest common prefix of any majority subset of S, it holds that $chain' \prec chain$.

Suppose L is a node set the majority of whom are alert at time t. Suppose that *chain* is real-world admissible w.r.t. S_L. Now we take the set of honest nodes in L and compute the longest prefix of their chains; and $chain_h$ denote this longest prefix. Suppose the aforementioned $chain'$ is computed by taking a subset $S' \subseteq S_L$ that comprise majority. Since the majority are alert at time t in L, one alert node must exist in S'. Clearly $chain'$ should be at least as long as $chain_h$. Therefore we conclude that $chain_h \prec chain'$.

It suffices to argue that $chain_h[-\epsilon^*\kappa].\mathsf{time} > t - \omega$. Let $i \in L$ be a node alert at time t, we know that $chain_h \prec chain_i^t$. Due to consistency, there cannot be more than $\epsilon_1\kappa$ blocks after $chain_h$ in $chain_i^t$. Now the fact that $chain_h[-\epsilon^*\kappa].\mathsf{time} > t - \omega$ follows from Claim 3.

□

Lemma 10 (Success of AddtoTree)**.** For any compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, there exists a negligible function negl such that for every κ

$$\Pr\left[\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{hyb}}}(\mathcal{S}^{\mathcal{A}}, \mathcal{Z}, \kappa) : \mathsf{view} \text{ aborts due to } \mathsf{extend\text{-}failure}\right] \leq \mathsf{negl}(\kappa)$$

Proof. Straightforward by Claim 5, Claim 6, and the definition of $\mathcal{S}$. □

10.5 Indistinguishability of the Real-World and Simulated Executions

Lemma 11 (Indistinguishability of the real-world and simulated executions)**.** For any $\Pi_{\text{snowwhite}}$-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, conditioned on the simulated execution not aborting, then $\mathcal{Z}$'s view in the simulated execution and real execution are identically distributed.

Proof. We now prove this lemma.

Hybrid 1. Same as the simulated execution, but with the following modification: when the an honest node needs to call the ideal-world $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}(chain, B)$, a real-world algorithm is adopted for extending the chain: the honest node calls $\mathsf{eligible}^t(chain, \mathsf{pk})$ where H is implemented by $\mathcal{S}$. If the outcome is 1, add $chain||B$ to $\widetilde{\mathcal{F}}_{\text{punctual}}.\mathsf{tree}$.

Claim 7. No p.p.t. $\mathcal{Z}$ can distinguish the simulated execution and Hybrid 1 except with negligible probability.

Proof. It suffices to show that if an alert node tries to extend a *chain* at time t, then *chain* is simulation valid and has a sufficiently long prefix at t. If this is true, then by Claim 4, using the real-world algorithm to decide whether a node is leader is equivalent to what $\widetilde{\mathcal{F}}_{\text{punctual}}$ thinks. Therefore, using the real-world algorithm to extend the chain rather than calling $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{extend}$ would be equivalent. Given Claim 6 and the definition of the simulated execution, it is not hard to see that every *chain* an alert node tries to extend is simulation valid and has a sufficiently long prefix.

□

Hybrid 2. Same as Hybrid 1, but with the following modification: whenever an honest node receives a *chain*′ whose length is longer than its own *chain*, do real-world checks instead of calling $\widetilde{\mathcal{F}}_{\text{punctual}}.\texttt{verify}$.

Claim 8. No p.p.t. $\mathcal{Z}$ can distinguish Hybrid 1 and Hybrid 2 except with negligible probability.

Proof. Notice that $\mathcal{S}$ always performs real-world checks on behalf of an alert node i before forwarding any *chain* to an alert node i. Therefore, it is easy to see that the claim is true given that we have proved Lemma 10. □

Finally, to prove Lemma 11, it suffices to observe that Hybrid 2 is equivalent to the real-world execution by a standard argument of redrawing algorithm boundaries. □

Acknowledgments

We thank Rachit Agarwal, Kai-Min Chung, and Ittay Eyal 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] `http://realtimebitcoin.info/`.

[2] Private communication with Vitalik Buterin, Vlad Zamfir, and the Ethereum Research Foundation.

[3] A brief look at bitcoin wallet statistics. `http://bitcoinx.io/news/reports/a-brief-look-at-bitcoin-wallet-statistics/`.

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

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

[6] 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.

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

[8] 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.

[9] Iddo Bentov, Rafael Pass, and Elaine Shi. The sleepy model of consensus. Manuscript, 2016.

[10] 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.

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

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

[13] 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.

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

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

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

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

[18] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gun Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains (a position paper). In *Bitcoin Workshop*, 2016.

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

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

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

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

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

[24] 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.

[25] 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.

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

[27] Mark Gimein. Virtual bitcoin mining is a real-world environmental disaster. `http://www.bloomberg.com/news/articles/2013-04-12/virtual-bitcoin-mining-is-a-real-world-environmental-disaster`.

[28] Jacob Grimm and Wilhelm Grimm. Little snow-white, 1812.

[29] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. Flexible paxos: Quorum intersection revisited. `https://arxiv.org/abs/1608.06696`.

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

[31] Aggelos Kiayias, Ioannis Konstantinou, Alexander Russell, Bernardo David, and Roman Oliynykov. A provably secure proof-of-stake blockchain protocol, `https://eprint.iacr.org/2016/889`.

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

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

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

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

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

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

[38] 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.

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

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

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

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

[43] 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/.

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

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

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

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

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

[49] John Quiggin. Bitcoins are a waste of energy - literally. http://www.abc.net.au/news/2015-10-06/quiggin-bitcoins-are-a-waste-of-energy/6827940.

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

[51] 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.

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