The Sleepy Model of Consensus

Iddo Bentov Rafael Pass Elaine Shi

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

Abstract

The literature on distributed computing (as well as the cryptography literature) typically considers two types of players—*honest* players and *corrupted* players. Resilience properties are then analyzed assuming a lower bound on the fraction of honest players. Honest players, however, are not only assumed to follow the prescribed the protocol, but also assumed to be *online* throughout the whole execution of the protocol. The advent of "large-scale" consensus protocols (e.g., the blockchain protocol) where we may have millions of players, makes this assumption unrealistic. In this work, we initiate a study of distributed protocols in a "sleepy" model of computation where players can be either *online* (alert) or *offline* (asleep), and their online status may change at any point during the protocol. The main question we address is:

> *Can we design consensus protocols that remain resilient under "sporadic participation", where at any given point, only a subset of the players are actually online?*

As far as we know, all standard consensus protocols break down under such sporadic participation, even if we assume that 99% of the online players are honest.

Our main result answers the above question in the affirmative. We present a construction of a consensus protocol in the sleepy model, which is resilient assuming only that a *majority of the online players are honest.* Our protocol relies on a Public-Key Infrastructure (PKI), a Common Random String (CRS) and is proven secure in the timing model of Dwork-Naor-Sahai (STOC'98) where all players are assumed to have weakly-synchronized clocks (all clocks are within Δ of the "real time") and all messages sent on the network are delivered within Δ time, and assuming the existence of sub-exponentially secure collision-resistant hash functions and enhanced trapdoor permutations. Perhaps surprisingly, our protocol significantly departs from the standard approaches to distributed consensus, and we instead rely on key ideas behind Nakamoto's blockchain protocol (while dispensing the need for "proofs-of-work"). We finally observe that sleepy consensus is impossible in the presence of a dishonest majority of online players.

[*]`http://www.initc3.org/`

1 Introduction

Consensus protocols are at the core of distributed computing and also provide a foundational building protocol for multi-party cryptographic protocols. In this paper, we consider consensus protocols for realizing a "linearly ordered log" abstraction—often referred to as *state machine replication* or *linearizability* in the distributed systems literature. Such protocols must respect two important resiliency properties, *consistency* and *liveness*. Consistency ensures that all honest nodes have the same view of the log, whereas liveness requires that transactions will be incorporated into the log quickly.

The literature on distributed computing as well as the cryptography literature typically consider two types of players—*honest* players and *corrupted/adversarial* players. The above-mentioned resiliency properties are then analyzed assuming a lower bound on the fraction of honest players (e.g., assuming that at least a majority of the players are honest). Honest players, however, are not only assumed to follow the prescribed the protocol, but also assumed to be *online* throughout the whole execution of the protocol. Whereas this is a perfectly reasonable assumption for the traditional environments in which consensus protocols typically were deployed (e.g., within a company, say "Facebuck", to support an application, say "Fackbuck Credit", where the number of nodes/players is roughly a dozen), the advent of "large-scale" consensus protocols (such as e.g., the blockchain protocol)—where want to achieve consensus among *millions* of players—makes this latter assumption unrealistic. (For instance, in bitcoin, only a small fraction of users having bitcoins are actually participating as miners.)

1.1 The Sleepy Model of Consensus

Towards addressing this issue, we here initiate a study of distributed protocols in a "sleepy" model of computation. In this model, players can be either *online* ("awake/active") or *offline* ("asleep"), and their online status may change at any point during the protocol execution. The main question we address is:

> *Can we design consensus protocols that remain resilient under "sporadic participation"—where at any given point, only a subset of the players are actually online—assuming an appropriate fraction (e.g., majority) of the online players are honest?*

As far as we know, this question was first raised by Micali [31] in a recent manuscript[1] —he writes "*... a user missing to participate in even a single round is pessimistically judged malicious—although, in reality, he may have only experienced a network-connection problem, or simply taken a break. [..] One possibility would be to revise the current Honest Majority of Users assumption so as it applies only to the "currently active" users rather than the "currently existing" users.*" In Micali's work, however, a different path is pursued.[2] In contrast, our goal here is to address this question. It is easy to see that consensus is impossible in this model unless we assume that at least a majority of the awake players are honest (if the set of awake players can arbitrarily change throughout the execution)—briefly, the reason for this is that a player that wakes up after being asleep for a long time cannot distinguish the real execution by the honest player and an emulated

[1]Although our paper is subsequent, at the original time of writing this paper, we were actually not aware of this; this discussion was present in the arXiv version from August 2016, but is no longer present in the most recent version of his manuscript.

[2]Briefly, rather than designing a protocol that remains resilient under this relaxed honesty assumption, he designs a protocol under an incomparable "honest-but-lazy" assumption, where honest players only are required to participate at infrequent but individually prescribed rounds (and if they miss participation in their prescribed round, they are deemed corrupted). Looking forward, the honest strategy in our protocols also satisfies such a laziness property.

"fake" execution by the malicious players, and thus must choose the "fake" one with probability at least $\frac{1}{2}$. We formalize this in Theorem 5 (in Section 6).

We then consider the following question:

> *Can we design a consensus protocol that achieves consistency and liveness assuming only that a* **majority of the online players** *are honest?*

As far as we know, all standard consensus protocols break down in the sleepy model, even if we assume that 99% of the online players are honest! Briefly, the standard protocols can be divided into two types: 1) protocols that assume *synchronous communication*, where all messages sent by honest players are guaranteed to be received by all other honest nodes in the next round; or, 2) protocols handling *partially synchronous* or *asynchronous* communication, but in this case require knowledge of a tight bound on the number of actually participating honest players. In more detail:

- Traditional synchronous protocols (e.g., [13,17,22]) crucially rely on messages being delivered in the next round (or within a known, bounded delay Δ) to reach agreement. By contrast, in the sleepy model, consider an honest player that falls asleep for a long time (greater than Δ) and then wakes up at some point in the future; it now receives all "old" messages with a significantly longer delay (breaking the synchrony assumption). In these protocols, such a player rejects all these old messages and would never reach agreement with the other players. It may be tempting to modify e.g., the protocol of [13] to have the players reach agreement on some transaction if some threshold (e.g., majority) of players have approved it—but the problem then becomes how to set the threshold, as the protocol is not aware of how many players are actually awake!

- The partially synchronous or asynchronous protocols (e.g., [8,12,14,30,33,39]) a-priori seem to handle the above-mentioned issue with the synchronous protocol: we can simply view the sleeping player as receiving messages with a long delay (which is allowed in the asynchronous model of communication). Here, the problem instead is the fact that the number of awake players may be significantly smaller than the total number of players, and this means that no transactions will even be approved! A bit more concretely, these protocols roughly speaking approve transactions when a certain *number* of nodes have "acknowledged" them–for instance, in the classic BFT protocol of Castro and Liskov [12] (which is resilient in the standard model assuming a fraction $\frac{2}{3}$ of *all players* are honest), players only approve a transaction when they have heard $\frac{2N}{3}$ "confirmations" of some message where N is the total number of parties. The problem here is that if, say, only half of the N players are awake, the protocols stalls. And again, as for the case of synchronous protocols, it is not clear how to modify this threshold without knowledge of the number of awake players.

1.2 Main Result

Our main result answers the above question in the affirmative. We present constructions of consensus protocols in the sleepy model, which are resilient assuming only that a *majority of the awake players are honest*. Our protocols relies on the existence of a "bare" Public-Key Infrastructure (PKI)[3], the existence of Common *Random* String (CRS)[4] and is proven secure in a simple

[3]That is, players have some way of registering public keys; for honest players, this registration happens before the beginning of the protocol, whereas corrupted players may register their key at any point. We do not need players to e.g., prove knowledge of their secret-key.

[4]That is a commonly known truly random string "in the sky".

version of the *timing model* of Dwork-Naor-Sahai [34] where all players are assumed to have weakly-synchronized clocks—all clocks are within Δ of the “real time”, and all messages sent on the network are delivered within Δ time.

Our first protocol relies only on the existence of collision-resistant hash functions (and it is both practical and extremely simple to implement, compared to standard consensus protocols); it, however, only supports static corruptions and a static (fixed) schedule of which nodes are awake at what time step—we refer to this as a “static online schedule”.

Theorem 1 (Informal)**.** *Assume the existence of families of a collision-resistant hash functions (CRH). Then, there exists a protocol for state-machine replication in the Bare PKI, CRS and in the timing model, which achieves consistency and liveness assuming a static online schedule and static corruptions, as long as at any point in the execution, a majority of the awake players are honest.*

Our next construction, enhances the first one by achieving also resilience with an arbitrary adversarial selection of which nodes are online at what time; this protocol also handles adaptive corruptions of players. This new protocol, however, does so at the price of assuming subexponentially secure collision-resistant hash functions and enhanced trapdoor permutations (the latter are needed for the constructions of non-interactive zero-knowledge proofs).

Theorem 2 (Informal)**.** *Assume the existence of families of sub-exponentially secure collision-resistant hash functions (CRH), and enhanced trapdoor permutations (TDP). Then, there exists a state-machine replication protocol in the Bare PKI, CRS and timing model, which achieves consistency and liveness under adaptive corruptions as long as at any point in the execution, a majority of the awake players are honest.*

Perhaps surprisingly, our protocols significantly departs from the standard approaches to distributed consensus, and we instead rely on key ideas behind Nakamoto’s beautiful blockchain protocol [35], while dispensing the need for “proofs-of-work” [15]. As far as we know, our work demonstrates for the first time how the ideas behind Nakamoto’s protocol are instrumental in solving “standard” problems in distributed computing; we view this as our main conceptual contribution (and hopefully one that will be useful also in other contexts).

Our proof will leverage and build on top of the formal analysis of the Nakamoto blockchain by Pass et al. [36], but since we no longer rely on proofs-of-work, several new obstacles arise. Our main technical contribution, and the bulk of our analysis, is a new combinatorial analysis for dealing with these issues.

We finally mention that ad-hoc solutions for achieving consensus using ideas behind the blockchain (but without proof-of-work) have been proposed [2,4,25], none of these come with an analysis, and it is not clear to what extent they improve upon standard state-machine replication protocols (and more seriously, whether they even achieve the standard notion of consensus).

1.3 Technical Overview

We start by providing an overview of our consensus protocol which only handles a static online schedule and static corruptions; we next show how to enhance this protocol to achieve adaptive security.

As mentioned, the design of our consensus protocols draws inspiration from Bitcoin’s proof-of-work based blockchain [35]—the so-called “Nakamoto consensus” protocol. This protocol is designed to work in a so-called “permissionless setting” where anyone can join the protocol execution. In contrast, we here study consensus in the classic “permissioned” model of computation with a fixed set $[N]$ of participating players; additionally, we are assuming that the players can register public keys (whose authenticity can be verified). Our central idea is to eliminate the use of proofs of work in this protocol. Towards this goal, let us start by providing a brief overview of Nakamoto's beautiful blockchain protocol.

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

Removing proofs-of-work. Removing the proof-of-work from the Nakamoto blockchain while maintaining provable guarantees turns out to be subtle and the proof non-trivial. To remove the proof-of-work from Nakamoto's protocol, we proceed as follows: instead of rate limiting through computational power, we impose limits on the type of puzzle solutions that are admissible for each player. More specifically, we redefine the puzzle solution to be of the form $(\mathcal{P}, t)$ where $\mathcal{P}$ is the player's identifier and t is referred to as the block-time. An honest player will always embed the current time step as the block-time. The pair $(\mathcal{P}, t)$ is a "valid puzzle solution" if $\mathsf{H}(\mathcal{P}, t) < D_p$ where H denotes a random oracle (for now, we provide a protocol in the random oracle model, but as we shall see shortly, the random oracle can be instantiated with a CRS and a pseudorandom function), and D_p is a parameter such that the hash outcome is only smaller than D_p with probability p. If $\mathsf{H}(\mathcal{P}, t) < D_p$, we say that $\mathcal{P}$ is *elected leader at time* t. Note that several nodes may be elected leaders at the same time steps.

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

Whereas honest players will only attempt to mine solutions of the form $(\mathcal{P}, t)$ where t is the current time step, so far there is nothing that prevents the adversary from using incorrect block-times (e.g., time steps in past or the future). To prevent this from happening, we additionally impose the following restriction on the block-times in a valid chain:

1. A valid chain must have strictly increasing block-times;
2. A valid chain cannot contain any block-times for the "future" (where "future" is adjusted to account for nodes' clock offsets)

There are now two important technical issues to resolve. First, it is important to ensure that the block-time rules do not hamper liveness. In other words, there should not be any way for an adversary to leverage the block-time mechanism to cause alert nodes to get stuck (e.g., by injecting false block-times).

Second, although our block-time rules severely constrain the adversary, the adversary is still left with some wiggle room, and gets more advantage than alert nodes. Specifically, as mentioned earlier, the alert nodes only "mine" in the present (i.e., at the actual time-step), and moreover they never try to extend different chains of the same length. By contrast, the adversary can try to reuse past block-times in multiple chains. (In the proof of work setting, these types of attacks are not possible since there the hash function is applied also to the history of the chain, so "old" winning solutions cannot be reused over multiple chains; in contrast, in our protocol, the hash function is no longer applied to the history of the chain as this would give the attacker too many opportunities to become elected a leader by simply trying to add different transactions.)

Our main technical result shows that this extra wiggle room in some sense is insignificant, and the adversary cannot leverage the wiggle room to break the protocol's consistency guarantees. It turns out that dealing with this extra wiggle room becomes technically challenging, and none of the existing analysis for proof-of-work blockchains [20, 36] apply. More precisely, since we are using a blockchain-style protocol, a natural idea is to see whether we can directly borrow proof ideas from existing analyses of the Nakamoto blockchains [20, 36]. Existing works [20, 36] define three properties of blockchains—*chain growth* (roughly speaking that the chain grows at a certain speed), *chain quality* (that the adversary cannot control the content of the chain) and *consistency* (that honest players always agree on appropriate prefix of the chain)—which, as shown in earlier works [36, 38] imply the consistency and liveness properties needed for state-machine replication. Thus, by these results, it will suffice to demonstrate that our protocol satisfies these properties.

The good news is that chain growth and chain quality properties can be proven in almost identically the same way as in earlier Nakamoto blockchain analysis [36]. The bad news is that the consistency proofs of prior works [20, 36] break down in our setting (as the attacker we consider is now more powerful as described above). The core of our proof is a new, and significantly more sophisticated analysis for dealing with this.

Removing the random oracle. The above-described protocol relies on a random oracle. We note that we can in fact instantiate the random oracle with a PRF whose seed is selected in a common reference string (CRS). Roughly speaking, the reason for this is that in our proof we actually demonstrate the existence of some simple polynomial-time computable events—which only depend on the output of the hash function/PRF—that determine whether *any* (even unbounded) attacks can succeed. Our proof shows that with overwhelming probability over the choice of the random oracle, these events do not happen. By the security of the PRF, these events thus also happen only with negligible probability over the choice of the seed of the PRF.

Dealing with adaptive sleepiness and corruption. We remark that the above-described protocol only works if the choice of when nodes are awake is made before PRF seed is selected. If not, honest players that are elected leaders could simply be put to sleep at the time step when they need to act. The problem is that it is preditcable when a node will become a leader. To overcome this problem, we take inspiration from a beautiful idea from Micali's work [31]—we let each player pick its own *secret seed* to a PRF and publish a commitment to the seed as part of its public key; the player can then evaluate its own private PRF and also prove in zero-knowledge that the PRF was correctly evaluated (so everyone else can verify the correctness of outputs of the PRF);[5]. Finally, each player now instantiates the random oracle with their own "private" PRF. Intuitively, this prevents the above-mentioned attack, since even if the adversary can adaptively select which honest nodes go to sleep, it has no idea which of them will become elected leaders before they broadcast their block.

Formalizing this, however, is quite tricky (and we will need to modify the protocol). The problem is that if users pick their own seed for the PRF, then they may be able to select a "bad seed" which makes them the leader for a long period of time (there is nothing in the definition of a PRF that prevents this). To overcome this issue, we instead perform a "coin-tossing into the well" for the evaluation of random oracle: As before, the CRS specifies the seed k_0 of a PRF, and

[5]In essence, what we need is a VRF [32], just like Micali [31], but since we anyway have a CRS, we can rely on weaker primitives.

additionally, each user $\mathcal{P}$ commits to the seed $k[\mathcal{P}]$ of a PRF as part of their public key; node $\mathcal{P}$ can then use the following function to determine if it is elected in time t

$$\mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t) < D_p$$

where D_p is a difficulty parameter selected such that any single node is elected with probability p in a given time step. Further, $\mathcal{P}$ additionally proves in zero-knowledge that it evaluated the above leader election function correctly in any block it produces.

But, have we actually gained anything? A malicious user may still pick its seed $k[\mathcal{P}]$ after seeing k_0 and this may potentially cancel out the effect of having $\mathsf{PRF}_{k_0}(\cdot)$ there in the first place! (For instance, the string $\mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t)$ clearly is not random any more.) We note, however, that if the user seed $k[\mathcal{P}]$ is *significantly shorter* than the seed k_0, and the cryptographic primitives are subexponentially secure, we can rely on the same method that we used to replace the random oracle with a PRF to argue that even if $k[\mathcal{P}]$ is selected as a function of k_0, this only increases the adversaries success probability by a factor 2^L for each possibly corrupted user where $L := |k[\mathcal{P}]|$ is the bit-length of each user's seed (and thus at most 2^{NL} where N is the number of players) which still will not be enough to break security, if using a sufficiently big security parameter for the underlying protocol. We can finally use a similar style of a union bound to deal also with adaptive corruptions. (Note, however, that the loss in efficiency due to these complexity leveraging is non-trivial: the security parameter must now be greater than N; if we only require static corruption, and allow the CRS to be selected *after* all public keys are registered—which would be reasonable in practice—then, we can deal with adaptive sleepiness without this complexity leveraging and thus without the loss in efficiency).

1.4 Applications in Permissioned and Permissionless Settings

As mentioned earlier, the variants of our protocols that deal with static corruption (and static or adaptive sleepiness) need not employ complexity leveraging, thus they can be implemented and adopted in real-world systems. We believe that our sleepy consensus protocol would be highly desirable in the following application scenarios and the alike.

Permissioned setting: consortium blockchains. At the present, there is a major push where blockchain companies are helping banks across the world build "consortium blockchains". A consortium blockchain is where a consortium of banks each contribute some nodes and jointly run a consensus protocol, on top of which one can run distributed ledger and smart contract applications. Since enrollment is controlled, consortium blockchain falls in the classical "permissioned" model of consensus. Since the number of participating nodes may be large (e.g., typically involve hundreds of banks and possibly hundreds to thousands of nodes), many conjecture that classical protocols such as PBFT [12], Byzantine Paxos [27], and others where the total bandwidth scales quadratically w.r.t. the number of players might not be ideal in such settings. Our sleepy consensus protocol provides a compelling alternative in this setting — with sleepy consensus, tasks such as committee re-configuration can be achieved simply without special program paths like in classical protocols [28], and each bank can also administer their nodes without much coordination with other banks.

Permissionless setting: proof-of-stake. The subsequent work Snow White by Bentov, Pass, and Shi [5] adapted our protocol to a permissionless setting, and obtained one of the first provably secure proof-of-stake protocols. A proof-of-stake protocol is a permissionless consensus protocol to be run in an open, decentralized setting, where roughly speaking, each player has voting power proportional to their amount of stake in the cryptocurrency system (*c.f.* proof-of-work is where players have voting power proportional to their available computing power). Major cryptocurrencies such as Ethereum are eager to switch to a proof-of-stake model rather than proof-of-work to dispense with wasteful computation. To achieve proof-of-stake, the Snow White [5] extended the our sleepy consensus protocol by introducing a mechanism that relies the distribution of stake in the system to periodically rotate the consensus committee. Further Snow White dealt with other issues such as "nothing at stake" and posterior corruption that are well-known for proof-of-stake systems — note that these issues pertain only to proof-of-stake systems and are thus out of scope for our paper.

Comparison with independent work. Although proof-of-stake is not a focus of our paper, we comapre with a few independent works on proof-of-stake [24,31] due to the superficial resemblance of some elements of their protocol in comparison with ours. Specificaly, Micali proposes to adapt classical style consensus protocols to realize a proof-of-stake protocol [31]; the concurrent and independent work by Kiayias et al. [24] proposes to use a combination of blockchain-style protocol and classical protocols such as coin toss to realize proof-of-stake. Both these works would fail in the sleepy model like any classical style protocol. In comparison, we use a blockchain style protocol in a pure manner which is essential to achieving consensus in the sleepy model. We also point out that even when we replace Kiayias's coin toss protocol with an ideal random beacon, Kiayias's proof would still fail in the sleepy model — and there does not seem to be a trivial way to reinterpret their proof such that it works in the sleepy model. Other proof-of-stake protocols [2,4,25] may also bear superficial resemblance but they do not have formal security models or provable guarantees, and these protocols may also miss elements that turned out essential in our proofs.

1.5 Related Work

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

Models for permissioned consensus. Consensus in the permissioned setting [3,6–8,12–14,17–19,22,26–30,39] 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 [29] where nodes are interconnected with authenticated channels[6]; and the "*authenticated Byzantine*" model [13], where a public-key infrastructure exists, such that nodes can sign messages and such digital signatures can then be transferred.

Permissioned, synchronous protocols. Many feasibility and infeasibility results have been shown. Notably, Lamport et al. [29] 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 [13], 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 [13]

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

— however, if randomness is allowed, existing works have demonstrated expected constant round protocols that can tolerate up to a half corruptions [17, 22].

Permissioned, asynchronous protocols. A well-known lower bound by Fischer, Lynch, and Paterson [18] 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 crash. 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 [8, 19, 33], nodes' local randomness [3, 39], or randomness in network delivery [7]; and 2) clocks and timeouts, where nodes are allowed to read a clock and make actions based on the clock's value. This approach has been taken by well-known protocols such as PBFT [12] and FaB [30] 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. [14], who showed that no protocol (even when allowing randomness or clocks) can achieve security in the presence of a $\frac{1}{3}$ (or larger) corrupt coalition.

Guerraoui et al. [21] propose a technique to dynamically partition nodes into clusters with nice properties, such that they can achieve consensus in a hostile environment where nodes join and leave dynamically. Their scheme also fails in the sleepy model, when the set of online honest nodes in adjacent time steps can be completely disjoint.

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

Amazingly, cryptocurrencies such as Bitcoin and Ethereum have popularized the permissionless setting, and have demonstrated to us, that perhaps contrary to the common belief, highly interesting and non-trivial tasks can be attained in the permissionless setting. Underlying these cryptocurrency systems is a fundamentally new type of consensus protocol commonly referred to as proof-of-work blockchains [35]. Upon closer examination, these protocols circumvent known lower bounds such as those by Canetti et al. [1] and Lamport et al. [29] 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 [20, 36–38]. Notably, Garay et al. [20] formally analyze the Nakamoto blockchain protocol in synchronous networks. Pass et al. [36] extend their analysis to asynchronous networks. More recently, Pass and Shi [38] show how to perform committee election using permissionless consensus and then bootstrap instances of permissioned consensus — in this way, they show how to asymptotically improve the response time for permissionless consensus.

Finally, existing blockchains are known to suffer from a selfish mining attack [16], where a coalition wielding $\frac{1}{3}$ of the computation power can reap up to a half of the rewards. Pass and Shi [37] recently show how to design a fair blockchain (called Fruitchains) from any blockchain protocol with positive chain quality. Since our Sleepy consensus protocol is a blockchain-style protocol, we also inherit the same selfish mining attack. However, we can leverage the same techniques as Pass and Shi [37] to build a fair blockchain from Sleepy.

2 Definitions

2.1 Protocol Execution Model

We assume a standard Interactive Turing Machine (ITM) model [9–11] 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.

Corruption model. At the beginning of any time step t, $\mathcal{Z}$ can issue instructions of the form

$$(\texttt{corrupt}, i) \text{ or } (\texttt{sleep}, i, t_0, t_1) \text{ where } t_1 \geq t_0 \geq t$$

$(\texttt{corrupt}, i)$ causes node i to become corrupt at the current time, whereas $(\texttt{sleep}, i, t_0, t_1)$ where $t_1 \geq t_0 \geq t$ will cause node i to sleep during $[t_0, t_1]$. Note that since `corrupt` or `sleep` instructions must be issued at the very beginning of a time step, $\mathcal{Z}$ cannot inspect an honest node's message to be sent in the present time step, and then retroactively make the node sleep in this time step and erase its message.

Following standard cryptographic modeling approaches [9–11], at any 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.

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*.

Henceforth, we say that corruption (or sleepiness resp.) is *static* if $\mathcal{Z}$ must issue all `corrupt` (or `sleep` resp.) instructions before the protocol execution starts. We say that corruption (or sleepiness resp.) is *adaptive* if $\mathcal{Z}$ can issue `corrupt` (or `sleep` resp.) instructions at any time during the protocol's execution.

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.

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

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

Public-key infrastructure. We assume the existence of a public-key infrastructure (PKI). Specifically, we adopt the same technical definition of a PKI as in the Universal Composition framework [9]. Specifically, we shall assume that the PKI is an ideal functionality $\mathcal{F}_{\text{CA}}$ that does the following:

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

2.2 Notational Conventions

Negligible functions. A function $\mathsf{negl}(\cdot)$ is said to be *negligible* if for every polynomial $p(\cdot)$, there exists some λ_0 such that $\mathsf{negl}(\lambda) \leq \frac{1}{p(\lambda)}$ for every $\lambda \geq \lambda_0$.

Variable conventions. In this paper, unless otherwise noted, all variables are by default (polynomially bounded) functions of the security parameter λ. Whenever we say $\mathsf{var}_0 > \mathsf{var}_1$, this means that $\mathsf{var}_0(\lambda) > \mathsf{var}_1(\lambda)$ for every $\lambda \in \mathbb{N}$. 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 λ, we shall explicitly state it by calling it a constant.*

Protocol conventions. We adopt the universal composition framework [9–11] for formal modeling. Each protocol instance and functionality is associated with a session identifier *sid*. We omit writing this session identifier explicitly without risk of ambiguity. We assume that ideal functionalities simply ignore all messages from parties not pertaining to the protocol instance of interest.

3 Problem Definitions

In this section, we formally define a state machine replication protocol. State machine replication has been studied by the distributed systems literature for 30 years. In state machine replication, nodes agree on a linearly ordered log over time, in a way that satisfies consistency and liveness. In this section, we make explicit the formal abstraction for state machine replication. We then define an alternative blockchain abstraction first proposed by Garay et al. [20] and Pass et al. [36]. We point out that a blockchain abstraction implies the classical state machine replication abstraction as shown by Pass and Shi [38]. Therefore, while our final goal is to achieve classical state machine replication, we will construct a blockchain protocol as a stepping stone. Separately, this connection between modern blockchains and classical state machine replication is also interesting in its own right — this has been the common wisdom in the community, but we formalize this intuition.

3.1 State Machine Replication

We will aim to realize a state machine replication abstraction, also frequently referred to as a "totally ordered log" or "linearity" by the distributed systems literature. In a replicated state machine, nodes agree on a LOG over time that is basically a list of transactions; and further, consistency and liveness are guaranteed.

More formally, a state machine replication abstraction satisfies the following — here we adopt the same definitions as Pass and Shi [38].

Inputs and outputs. The environment $\mathcal{Z}$ may input a set of transactions txs to each alert node in every time step. In each time step, an alert node outputs to the environment $\mathcal{Z}$ a totally ordered LOG of transactions (possibly empty).

Security definitions. Let p.p.t. algorithms $(\mathcal{A}, \mathcal{Z})$ be compliant with respect to a state machine replication protocol Π. Let T_{confirm} be a polynomial function in $\lambda, N, N_{\text{crupt}}$, and Δ. We say that a state machine replication protocol Π is secure w.r.t. $(\mathcal{A}, \mathcal{Z})$ and has transaction conformation time T_{confirm} if there exists a negligible function negl such that for every sufficiently large $\lambda \in \mathbb{N}$, all but $\mathsf{negl}(\lambda)$ fraction of the views sampled from $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ satisfy the following properties:

- *Consistency*: An execution trace view satisfies consistency if the following holds:
  - *Common prefix.* Suppose that in view, an alert node i outputs LOG to $\mathcal{Z}$ at time t, and an alert node j (same or different) outputs LOG' to $\mathcal{Z}$ at time t', it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that in view, a node i is alert at time t and $t' \geq t$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.
- *Liveness*: An execution trace view satisfies T_{confirm}-liveness if the following holds: suppose that in view, the environment $\mathcal{Z}$ inputs txs to an alert node at time $t \leq |\mathsf{view}| - T_{\text{confirm}}$. Then, for any node i alert at any time $t' \geq t + T_{\text{confirm}}$, let LOG be the output of node i at time t', it holds that any $\mathsf{tx} \in \mathsf{txs}$ is included in LOG.

  Intuitively, liveness says that transactions input to an alert node gets included in their LOGs within T_{confirm} time.

3.2 Blockchain Formal Abstraction

In this section, we define the formal abstraction and security properties of a blockchain. As Pass and Shi [38] recently show, a blockchain abstraction implies a classical state machine replication abstraction. Our definitions follow the approach of Pass et al. [36], which in turn are based on earlier definitions from Garay et al. [20], and Kiayias and Panagiotakos [23].

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

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

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

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

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

3.2.1 Chain Growth

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

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

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

where we quantify over nodes i, j such that i is alert in time step t and j is alert in time $t + t'$ in view.

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

- **(consistent length)** for all time steps $t \leq |\mathsf{view}| - \Delta$, $t + \Delta \leq t' \leq |\mathsf{view}|$, for every two players i, j such that in view i is alert at t and j is alert at t', we have that $|\mathsf{chain}_j^{t'}(\mathsf{view})| \geq |\mathsf{chain}_i^t(\mathsf{view})|$
- **(chain growth lower bound)** for every time step $t \leq |\mathsf{view}| - t_0$, we have
$$\mathsf{min\text{-}chain\text{-}increase}^{t,t_0}(\mathsf{view}) \geq T.$$
- **(chain growth upper bound)** for every time step $t \leq |\mathsf{view}| - t_1$, we have
$$\mathsf{max\text{-}chain\text{-}increase}^{t,t_1}(\mathsf{view}) \leq T.$$

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

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

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

3.2.2 Chain Quality

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

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

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

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

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

3.2.3 Consistency

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

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

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

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

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).

3.3 Blockchain Implies State Machine Replication

We note that a blockchain protocol implies state machine replication, if alert nodes simply output the stablized part of their respective chains (i.e., $\mathsf{chain}[: -\lambda]$) as their LOG. This draws a tight connection between modern blockchains and classical consensus (i.e., state machine replication) protocols that have been studied by the distributed systems literature for 30 years. In this paper, to obtain a classical state machine replication protocol, we will instead construct a blockchain protocol as a stepping stone.

Lemma 1 (Blockchains imply state machine replication [38]). *Let $\Pi_{\text{blockchain}}$ denote a blockchain protocol that satisfies (T_G, g_0, g_1)-chain growth, (T_Q, μ)-chain quality, and T_C-consistency. It holds that if alert nodes output* $\mathsf{chain}[: -T_C]$ *as their* LOG*s, then the resulting protocol realizes a state machine replication abstraction with transaction confirmation time* $T_{\text{confirm}} := O(\frac{T_G+T_Q+T_C}{g_0} + \Delta)$.

Proof. This lemma was proved in the Hybrid Consensus paper by Pass and Shi [38]. □

4 Sleepy Consensus for Static Corruptions

In this section, we will describe our basic Sleepy consensus protocol that is secure under static corruptions and static sleepiness. In other words, the adversary (and the environment) must declare upfront which nodes are corrupt as well as which nodes will go to sleep during which intervals. Furthermore, the adversary (and the environment) must respect the constraint that at any moment of time, roughly speaking the majority of *online* nodes are honest.

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

4.1 Format of Real-World Blocks

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

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

Valid blocks. We say that a tuple

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

is a valid block iff

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

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

- $chain[0] = genesis = (\bot, \bot, \mathsf{time} = 0, \bot, \bot, h = \vec{0})$, commonly referred to as the genesis block;
- $chain[-1].\mathsf{time} \leq t$; and
- for all $i \in [1..\ell]$ where $\ell := |chain|$, the following holds:
  1. $chain[i]$ is a valid block;
  2. $chain[i].h_{-1} = chain[i-1].h$;

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

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

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

Every time step:
  • receive input transactions(txs) from $\mathcal{Z}$
  • let $t$ be the current time, if $\mathsf{eligible}^t(\mathcal{P})$ where $\mathcal{P}$ is the current node's party identifier:
        let $\sigma := \Sigma.\mathtt{sign}(\mathsf{sk}, chain[-1].h, \mathsf{txs}, t)$, $h' := \mathsf{d}(chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \sigma)$,
        let $B := (chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \sigma, h')$, let $chain := chain||B$ and gossip $chain$
  • output $\mathsf{extract}(chain)$ to $\mathcal{Z}$ where extract is the function outputs an ordered list containing
    the txs extracted from each block in $chain$

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

Figure 1: The sleepy consensus protocol. The difficulty parameter D_p is defined such that the hash outcome is less than D_p with probability p. For simplicity, here we describe the scheme with a random oracle H — however as we explain in this section, H can be removed and replaced with a pseurdorandom function and a common reference string.

3. $chain[i].\mathsf{time} > chain[i-1].\mathsf{time}$, i.e., block-times are strictly increasing; and
4. let $t := chain[i].\mathsf{time}$, $\mathcal{P} := chain[i].\mathcal{P}$, it holds that $\mathsf{eligible}^t(\mathcal{P}) = 1$.

4.2 The Sleepy Consensus Protocol

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

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

Removing the random oracle. Although we described our scheme assuming a random oracle H, it is not hard to observe that we can replace the random oracle with a common reference string crs and a pseudo-random function PRF. Specifically, the common reference string $k_0 \leftarrow_{\$} \{0,1\}^\lambda$ is randomly generated after $\mathcal{Z}$ spawns all corrupt nodes and commits to when each honest node shall sleep. Then, we can simply replace calls to $\mathsf{H}(\cdot)$ with with $\mathsf{PRF}_{k_0}(\cdot)$.

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

Remark on foreknowledge of Δ. Note that our protocol $\Pi_{\text{sleepy}}(p)$ is parametrized with a parameter p, that is, the probability that any node is elected leader in any time step. Looking ahead, due to our compliance rules explained later in Section 4.3, it is sufficient for the protocol to have foreknowledge of both N and Δ, then to attain a targeted resilience (i.e., the minimum ratio of alert nodes over corrupt ones in any time step), the protocol can choose an appropriate value for p based on the "resilience" compliance rules (see Section 4.3).

Later in Section 6, we will justify why foreknowledge of Δ is necessary: we prove a lower bound showing that any protocol that does not have foreknowledge of Δ cannot achieve state machine replication even when all nodes are honest.

4.3 Compliant Executions

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

Compliant executions. We say that $(\mathcal{A}, \mathcal{Z})$ is Π_{sleepy}-compliant w.r.t. parameters $(\phi, N, N_{\text{crupt}}, \Delta, p)$ if the following holds for any $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{sleepy}}}(\mathcal{A}, \mathcal{Z}, \lambda)$ with non-zero support — henceforth whenever the context is clear, we often say that $(\mathcal{A}, \mathcal{Z})$ is Π_{sleepy}-compliant while omitting the parameters.

- **Static corruption and sleepiness.** $\mathcal{Z}$ must issue all `corrupt` and `sleep` instructions prior to the start of the protocol execution. We assume that $\mathcal{A}$ cannot query the random oracle H prior to protocol start.
- **Number of nodes.** $(\mathcal{A}, \mathcal{Z})$ spawns a total of N nodes among which N_{crupt} are corrupt and the remaining are honest.
- **Δ-bounded network delivery.** Although the adversary $\mathcal{A}$ can delay or reorder network messages, it must respect the following constraint: if an alert node i gossips a message at time t, then any node j alert at time $t' \geq t + \Delta$ (including ones that wake up later) will have received the message.
- **Resilience.** Let $\mathsf{alert}^t(\mathsf{view})$ be the number of nodes that are alert at time t in view. We require that for every $t \leq |\mathsf{view}|$,

  $$\frac{\mathsf{alert}^t(\mathsf{view})}{N_{\text{crupt}}} \geq \frac{1+\phi}{1-2pN\Delta}$$

  and further there is some constant $0 < c < 1$ such that $2pN\Delta < 1 - c$ — note that the latter condition makes sure that if the fraction of corrupt nodes is smaller than $O(1)$, we simply will upper bound it with a constant.

Informally, we require that at any point of time, there are more alert nodes than corrupt ones by a constant margin.

Looking forward, in our analysis and theorem statements, we always assume that ϕ is a constant independent of the security parameter λ.

Useful notations. We define additional notations that will become useful later.

1. Let $N_{\text{alert}} := N_{\text{crupt}} \cdot \frac{1+\phi}{1-2pN\Delta}$ be a lower bound on the number of alert nodes in every time step;
2. Let $\alpha := pN_{\text{alert}}$ be a lower bound on the expected number of alert nodes elected leader in any single time step;
3. Let $\beta := pN_{\text{crupt}} \geq 1-(1-p)^{N_{\text{crupt}}}$ be the expected number of corrupt nodes elected leader in any single time step; notice that β is also an upper bound on the probability that some corrupt node is elected leader in one time step.

4.4 Theorem Statement

We now state our theorem for static corruption.

Theorem 3 (Security of Π_{sleepy} under static corruption)**.** *Assume that the* PRF *scheme and that the signature scheme Σ are secure. For any positive constants $\phi, \epsilon > 0$, any super-logarithm $T_0 = \omega(\log \lambda)$, any positive N, N_{crupt}, and Δ, $\Pi_{sleepy}(p)$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0 consistency against any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ that is Π_{sleepy}-compliant w.r.t parameters $(\phi, N, N_{\text{crupt}}, \Delta, p)$, where relevant parameters are defined below:*

- *chain growth lower bound parameter $g_0 = (1-\epsilon)(1-2pN\Delta)\alpha$;*
- *chain growth upper bound parameter $g_1 = (1+\epsilon)Np$; and*
- *chain quality parameter $\mu = 1 - \frac{1-\epsilon}{1+\phi}$;*

The proof of this theorem will be presented in Section 7.

Corollary 1 (Statically secure state machine replication in the sleepy model.)**.** *Assume the existence of collision-resistant hash functions. Then, for any constant $\epsilon > 0$, there exists a protocol that achieves state machine replication in the Bare PKI, CRS, and in the timing model, assuming static corruptions and static sleepiness, as long as $\frac{1}{2}+\epsilon$ fraction of awake nodes are honest in any time step.*

Proof. Straightforward from Theorem 3 and Lemma 1. □

5 Achieving Adaptive Security

So far, we have assumed that the adversary issues both `corrupt` and `sleep` instructions statically upfront. In this section, we will show how to achieve adaptive security with complexity leveraging. It turns out even with complexity leveraging the task is non-trivial.

5.1 Intuition: Achieving Adaptive Sleepiness

To simplify the problem, let us first consider how to achieve adaptive sleepiness (but static corruption). In our statically secure protocol Π_{sleepy}, the adversary can see into the future for all honest and corrupt players. In particular, the adversary can see exactly in which time steps each honest node is elected leader. If `sleep` instructions could be adaptively issued, the adversary could simply put a node to sleep whenever he is elected leader, and wake up him when he is not leader. This way, the adversary can easily satisfy the constraint that at any time, the majority of the online nodes must be honest, while ensuring that no alert nodes are ever elected leader (with extremely high probability).

To defeat such an attack and achieve adaptive sleepiness (but static corruption), we borrow an idea that was (informally) suggested by Micali [31]. Basically, instead of computing a "leader ticket" η by hashing the party's (public) identifier and the time step t and by checking $\eta < D_p$ to determine if the node is elected leader, we will instead have an honest node compute a pseudorandom "leader ticket" itself using some secret known only to itself. In this way, the adversary is no longer able to observe honest nodes' future. The adversary is only able to learn that an honest node is elected leader in time step t when the node actually sends out a new chain in t — but by then, it will be too late for the adversary to (retroactively) put that node to sleep in t.

A naïve attempt. Therefore, a naïve attempt would be the following.

- Each node $\mathcal{P}$ picks its own PRF key $k[\mathcal{P}]$, and computes a commitment $c := \mathsf{comm}(k[\mathcal{P}]; r)$ and registers c as part of its public key with the public-key infrastructure $\mathcal{F}_{\text{CA}}$. To determine whether it is elected leader in a time step t, the node computes

$$\mathsf{PRF}_{k[\mathcal{P}]}(t) < D_p$$

  where D_p is a difficulty parameter related to p, such that any node gets elected with probability p in a given time step.

- Now for $\mathcal{P}$ to prove to others that it is elected leader in a certain time step t, $\mathcal{P}$ can compute a non-interactive zero-knowledge proof that the above evaluation is done correctly (w.r.t. to the commitment c that is part of $\mathcal{P}$'s public key).

A second attempt. This indeed hides honest nodes' future from the adversary; however, the adversary may not generate $k[\mathcal{P}^*]$ at random for a corrupt player $\mathcal{P}^*$. In particular, the adversary can try to generate $k[\mathcal{P}^*]$ such that $\mathcal{P}^*$ can get elected in more time steps. To defeat such an attack, we include a relatively long randomly chosen string k_0 in the common reference string. For a node $\mathcal{P}$ to be elected leader in a time step t, the following must hold:

$$\mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t) < D_p$$

As before, a node can compute a non-interactive zero-knowledge proof (to be included in a block) to convince others that it computed the leader election function correctly.

Now the adversary can still adaptively choose $k[\mathcal{P}^*]$ after seeing the common reference string k_0 for a corrupt node $\mathcal{P}^*$ to be elected in more time steps; however, it can only manipulate the outcome to a limited extent: in particular, since k_0 is much longer than $k[\mathcal{P}^*]$, the adversary does not have enough bits in $k[\mathcal{P}^*]$ to manipulate to defeat all the entropy in k_0.

Parametrization and analysis. Using the above scheme, we can argue for security against an adaptive sleepiness attack. However, as mentioned above, the adversary can still manipulate the outcome of the leader election to some extent. For example, one specific attack is the following:

suppose that the adversary controls $O(N)$ corrupt nodes denoted $\mathcal{P}_0^*, \ldots, \mathcal{P}_{O(N)}^*$ respectively. With high probability, the adversary can aim for the corrupt nodes to be elected for $O(N)$ consecutive time slots during which period the adversary can sustain a consistency and a chain quality attack. To succeed in such an attack, say for time steps $[t : t+O(N)]$, the adversary can simply try random user PRF keys on behalf of $\mathcal{P}_0^*$ until it finds one that gets $\mathcal{P}_0^*$ to be elected in time t (in expectation only $O(\frac{1}{p})$ tries are needed); then the adversary tries the same for node $\mathcal{P}_1^*$ and time $t+1$, and so on.

Therefore we cannot hope to obtain consistency and chain quality for $O(N)$-sized windows. Fortunately, as we argued earlier, since the adversary can only manipulate the leader election outcome to a limited extent given that the length of k_0 is much greater than the length of each user's PRF key, it cannot get corrupt nodes to be consecutively elected for too long. In our proof, we show that as long as we consider sufficiently long windows of N^c blocks in length (for an appropriate constant c and assuming for simplicity that $N = \omega(\log \lambda)$), then consistency and chain quality will hold except with negligible probability.

5.2 Intuition: Achieving Adaptive Corruption

Once we know how to achieve adaptive sleepiness and static corruption, we can rely on complexity leveraging to achieve adaptive corruption. This part of the argument is standard: suppose that given an adversary under static corruption that can break the security properties of the consensus protocol, there exists a reduction that breaks some underlying complexity assumption. We now modify the reduction to guess upfront which nodes will become corrupt during the course of execution, and it guesses correctly with probability $\frac{1}{2^N}$. This results in a 2^N loss in the security reduction, and therefore if we assume that our cryptographic primitives, including the PRF, the digital signature scheme, the non-interactive zero-knowledge proof, the commitment scheme, and the collision-resistant hash family have sub-exponential hardness, we can lift the static corruption to adaptive corruption.

Below, we put the aforementioned ideas together and present our adaptively secure scheme formally.

5.3 Preliminary: Non-Interactive Zero-Knowledge Proofs

In the remainder of this section, $f(\lambda) \approx g(\lambda)$ means that there exists a negligible function $\nu(\lambda)$ such that $|f(\lambda) - g(\lambda)| < \nu(\lambda)$.

A non-interactive proof system henceforth denoted NIZK for an NP language $\mathcal{L}$ consists of the following algorithms:

- $\mathsf{crs} \leftarrow \texttt{gen}(1^\lambda, \mathcal{L})$: Takes in a security parameter λ, a description of the language $\mathcal{L}$, and generates a common reference string crs.
- $\pi \leftarrow \texttt{prove}(\mathsf{crs}, \mathsf{stmt}, w)$: Takes in crs, a statement stmt, a witness w such that $(\mathsf{stmt}, w) \in \mathcal{L}$, and produces a proof π.
- $b \leftarrow \texttt{ver}(\mathsf{crs}, \mathsf{stmt}, \pi)$: Takes in a crs, a statement stmt, and a proof π, and outputs 0 or 1, denoting accept or reject.
- $(\overline{\mathsf{crs}}, \tau) \leftarrow \overline{\texttt{gen}}(1^\lambda, \mathcal{L})$: Generates a simulated common reference string $\overline{\mathsf{crs}}$ and a trapdoor τ.
- $\pi \leftarrow \overline{\texttt{prove}}(\overline{\mathsf{crs}}, \tau, \mathsf{stmt})$: Uses trapdoor τ to produce a proof π without needing a witness.

Perfect completeness. A non-interactive proof system is said to be perfectly complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{stmt}, w) \in \mathcal{L}$, we have that

$$\Pr\left[\ \mathsf{crs} \leftarrow \mathtt{setup}(1^\lambda, \mathcal{L}),\ \pi \leftarrow \mathtt{prove}(\mathsf{crs}, \mathsf{stmt}, w) : \mathtt{ver}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1\ \right] = 1$$

Computational zero-knowlege. Informally, an NIZK system is computationally zero-knowledge if the proof does not reveal any information about the witness to any polynomial-time (or subexponential time resp.) adversary. More formally, a NIZK system is said to have computational zero-knowledge, if for all non-uniform polynomial-time adversary $\mathcal{A}$ (or subexponential-time $\mathcal{A}$ resp.),

$$\Pr\left[\mathsf{crs} \leftarrow \mathtt{gen}(1^\lambda, \mathcal{L}) : \mathcal{A}^{\mathtt{prove}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1\right] \approx \Pr\left[(\overline{\mathsf{crs}}, \tau, \_) \leftarrow \overline{\mathtt{gen}}(1^\lambda, \mathcal{L}) : \mathcal{A}^{\overline{\mathtt{prove}}_1(\overline{\mathsf{crs}},\tau,\cdot,\cdot)}(\overline{\mathsf{crs}}) = 1\right]$$

In the above, $\mathtt{prove}_1(\overline{\mathsf{crs}}, \tau, \mathsf{stmt}, w)$ verifies that $(\mathsf{stmt}, w) \in \mathcal{L}$, and if so, outputs $\overline{\mathtt{prove}}(\overline{\mathsf{crs}}, \tau, \mathsf{stmt})$ which simulates a proof without knowing a witness. Otherwise, if $(\mathsf{stmt}, w) \notin \mathcal{L}$, the experiment aborts.

Computational soundness. We say that a NIZK scheme is computationally sound against any p.p.t. (or subexponential-time resp.) adversary, if for any p.p.t. (or subexponential-time resp.) adversary $\mathcal{A}$, it holds that

$$\Pr\left[\mathsf{crs} \leftarrow \mathtt{gen}(1^\lambda, \mathcal{L}), (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) : \mathtt{ver}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1 \text{ but } \mathsf{stmt} \notin \mathcal{L}\right] \approx 0$$

NP language used in our construction. In our construction, we will use the following NP language $\mathcal{L}$. A pair $(\mathsf{stmt}, w) \in \mathcal{L}$ iff

- parse $\mathsf{stmt} := (\eta, c, k_0, \mathcal{P}, \mathsf{time})$, parse $w := (k, r)$;
- it holds that $c = \mathsf{comm}(k; r)$ and $\mathsf{PRF}_k(\mathsf{time}) \oplus \mathsf{PRF}_{k_0}(\mathcal{P}, \mathsf{time}) = \eta$

5.4 Sleepy Consensus with Adaptive Security

Henceforth we use the shorthand $\mathcal{P}.\mathsf{upk}$ to mean $\mathcal{F}_{\mathrm{CA}}.\mathtt{lookup}(\mathcal{P})$. Specifically, $\mathcal{P}.\mathsf{upk}$ can be parsed as $\mathcal{P}.\mathsf{upk} := (\mathsf{pk}, c)$ where pk denotes a signature public key, and c corresponds to a perfectly binding commitment of a user's PRF key.

Valid blocks and valid blockchains are defined in a similar fashion as in the earlierstatically secure scheme — but we need to make minor changes to block format and validity rules to incorporate the fact that now each block carries its own zero-knowledge proof to vouch for its validity.

Valid blocks. We say that a tuple

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

is a valid block with respect to the difficulty parameter D_p and public parameters params iff

1. $\mathcal{P}$ is a valid node of the current protocol instance and has registered with $\mathcal{F}_{\mathrm{CA}}$;
2. Parse $\mathcal{P}.\mathsf{upk} := (\mathsf{pk}, \_)$, it holds that $\Sigma.\mathtt{ver}_{\mathsf{pk}}((h_{-1}, \mathsf{txs}, \mathsf{time}, \pi); \sigma) = 1$;
3. Parse $\mathcal{P}.\mathsf{upk} := (\_, c)$, parse $\mathsf{params} := (k_0, \mathsf{crs})$, it holds that $\mathsf{NIZK}.\mathtt{ver}(\mathsf{crs}, \mathsf{stmt}) = 1$ where $\mathsf{stmt} := (\eta, c, k_0, \mathcal{P}, \mathsf{time})$;
4. $\eta < D_p$; and

```
Protocol $\Pi^*_{\text{sleepy}}(p, \mathsf{params} := (k_0, \mathsf{crs}))$

On input init() from $\mathcal{Z}$:
    let $(\mathsf{pk}, \mathsf{sk}) := \Sigma.\mathtt{gen}(1^L)$, let $k \leftarrow_\$ \{0,1\}^L$, let $c := \mathsf{comm}(k; r)$ for $r \leftarrow_\$ \{0,1\}^L$;
    let $chain := genesis$, let $\mathsf{usk} := (\mathsf{sk}, c, k, r)$, register $\mathsf{upk} := (\mathsf{pk}, c)$ with $\mathcal{F}_{\text{CA}}$;

On receive $chain'$:
    assert $|chain'| > |chain|$ and $chain'$ is valid w.r.t. $D_p$, params, and the current time $t$;
    $chain := chain'$ and gossip $chain$

Every time step:
  • receive input transactions(txs) from $\mathcal{Z}$
  • let $t$ be the current time, let $\mathcal{P}$ be the current party's identifier, parse $\mathsf{usk} := (\mathsf{sk}, c, k, r)$
  • let $\eta := \mathsf{PRF}_k(t) \oplus \mathsf{PRF}_{k_0}(\mathcal{P}, t)$, if $\eta < D_p$:
        let $\pi := \mathsf{NIZK}.\mathtt{prove}(\mathsf{crs}, \mathsf{stmt}, w)$ where $\mathsf{stmt} := (\eta, c, k_0, \mathcal{P}, t)$, $w := (k, r)$
        let $\sigma := \Sigma.\mathtt{sign}_{\mathsf{sk}}(chain[-1].h, \mathsf{txs}, t, \eta, \pi)$, $h' := \mathsf{d}(chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \eta, \pi, \sigma)$,
        let $B := (chain[-1].h, \mathsf{txs}, t, \mathcal{P}, \eta, \pi, \sigma, h')$, let $chain := chain||B$ and gossip $chain$
  • output extract($chain$) to $\mathcal{Z}$ where extract is the function outputs an ordered list containing
    the txs extracted from each block in $chain$
```

Figure 2: The sleepy consensus protocol with adaptive security. The common reference string params is generated as follows: $k_0 \leftarrow_\$ \{0,1\}^{L_0}$, and $\mathsf{crs} \leftarrow \mathsf{NIZK}.\mathtt{gen}(1^L, \mathcal{L})$.

5. $h = \mathsf{d}(h_{-1}, \mathsf{txs}, \mathsf{time}, \mathcal{P}, \eta, \pi, \sigma)$, where $\mathsf{d} : \{0,1\}^* \to \{0,1\}^\lambda$ 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 *chain* denote an ordered chain of real-world blocks, we say that *chain* is a valid blockchain w.r.t. the difficulty parameter D_p, public parameters params, and t iff

- $chain[0] = genesis = (\bot, \bot, \mathsf{time} = 0, \bot, \bot, h = \vec{0})$, commonly referred to as the genesis block;
- $chain[-1].\mathsf{time} \leq t$; and
- for all $i \in [1..\ell]$, the following holds:
  1. $chain[i]$ is a valid block w.r.t. the difficulty parameter D_p and public parameters params;
  2. $chain[i].h_{-1} = chain[i-1].h$; and
  3. $chain[i].\mathsf{time} > chain[i-1].\mathsf{time}$, i.e., block-times are strictly increasing.

Protocol description. We present our adaptively secure scheme Π^*_{sleepy} in Figure 2. The main differences from the previous statically secure protocol are the following. As mentioned earlier, each node $\mathcal{P}$ picks a PRF secret key $k[\mathcal{P}]$ and registers a commitment c of $k[\mathcal{P}]$ with the public-key infrastructure $\mathcal{F}_{\text{CA}}$. Further, there is a longer random seed k_0 included in the common reference string. To determine whether a node $\mathcal{P}$ is elected leader in a given time step t, $\mathcal{P}$ checks whether

$\mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t) < D_p$. If $\mathcal{P}$ is elected leader, it can extend the chain with a block, and it includes a non-interactive zero-knowledge proof π in the block proving that it computed the leader election function correctly.

Compliant executions. We say that a p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ is Π^*_{sleepy}-compliant w.r.t. parameters $(\phi, N, N_{\text{crupt}}, \Delta, p)$ if $(\mathcal{A}, \mathcal{Z})$ is Π_{sleepy}-compliant w.r.t. parameters $(\phi, N, N_{\text{crupt}}, \Delta, p)$ — except that now we allow $\mathcal{Z}$ to adaptively corrupt nodes and make nodes sleep during the protocol execution. Note that $\mathcal{A}$ is allowed to register corrupt nodes's public keys with $\mathcal{F}_{\text{CA}}$ after seeing the common reference string.

Parameter choices for cryptographic building blocks. We assume that the the PRF function, the collision resistance hash, the signature scheme, and the NIZK have sub-exponential hardness. Throughout this paper, sub-exponential hardness means that except with 2^{-k^δ} probability, the cryptographic primitive with input length k is secure against any adversary running in time 2^{k^δ} for a fixed constant $\delta < 1$. We will use the following parameters:

- Each user's PRF key k has bit length $L = (2N + \log^2 \lambda)^{\frac{1}{\delta}}$;
- The common reference string k_0 has bit length $L_0 = (2LN)^{\frac{1}{\delta}}$;
- All other cryptographic schemes such as the hash function, the digital signature scheme, and the NIZK have input length $L = (2N + \log^2 \lambda)^{\frac{1}{\delta}}$.

5.5 Theorem Statement

Theorem 4 (Security of Π^*_{sleepy} under adaptive corruption)**.** *Assume that the* PRF*, the collision resistant hash family, and the signature scheme Σ all have subexponential security, and that the* NIZK *is perfectly complete, computational zero-knowledge and computationally sound against sub-exponential adversaries. Then, for any positive constants $\phi, \epsilon > 0$, any $T_0 \geq \Omega(LN)$, any positive $N, N_{\text{crupt}}, \Delta$, protocol $\Pi^*_{sleepy}(p)$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2 consistency against any p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ that is Π^*_{sleepy}-compliant w.r.t. $(\phi, N, N_{\text{crupt}}, \Delta, p)$ where relevant parameters are defined below:*

- *chain growth lower bound parameter $g_0 = (1 - \epsilon)(1 - 2pN\Delta)\alpha$;*
- *chain growth upper bound parameter $g_1 = (1 + \epsilon)Np$; and*
- *chain quality parameter $\mu = 1 - \frac{1-\epsilon}{1+\phi}$.*

The proof of this theorem will be presented in Section 8.

Corollary 2 (Adaptively secure state machine replication in the sleepy model.)**.** *Assume the existence of sub-exponentially hard collision-resistant hash functions, and sub-exponentially hard enhanced trapdoor permutations. Then, for any constant $\epsilon > 0$, there exists a protocol that achieves state machine replication in the Bare PKI, CRS, and the timing model against adaptive corruptions and adaptive sleepiness, as long as $\frac{1}{2} + \epsilon$ fraction of awake nodes are honest in any time step.*

Proof. Straightforward from Theorem 4 and Lemma 1. □

Remark 1 (A variant of practical interest.)**.** *Our complexity leveraging makes the security parameter dependent on N, the total number of players. This necessarily means that transaction confirmation will need to wait for* $\mathsf{poly}(N)$ *blocks.*

We point out a different variant that is of practical interest and which does not incur such blowup in security parameter and transaction confirmation time — this variant is directly implied *by our proofs in Section 8. Specifically, if we are willing to assume adaptive sleepiness and static corruption, and assume that the CRS may be chosen after registration of all public keys, then we will not need complexity leveraging, and therefore we can achieve state machine replication with the same protocol as in Figure 2, but with a tight security parameter λ that is independent of N. This also means that the transaction confirmation time is independent of N.*

6 Lower Bounds

6.1 Lower Bound on Resilience

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

Theorem 5 (Majority honest is necessary)**.** *In the sleepy execution model, it is not possible to realize state machine replication if there can be as many corrupt nodes than alert nodes — and this lower bound holds even assuming static corruption and the existence of a public-key infrastructure.*

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

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

□

6.2 Foreknowledge of Δ is Necessary

Recall that in our model, we assume that alert nodes can receive messages from other alert nodes within at most Δ delay. Further, we assume that Δ (or an upper bound on the network delay) is known to our protocol. Below, we show that making this assumption is necessary, since any protocol that does not have a-priori knowledge of Δ cannot securely realize state machine replication in the sleepy model.

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

On init: $\mathsf{tree} := \mathsf{genesis}$, $\mathsf{time}(\mathsf{genesis}) := 0$
On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{A}$ or internally:
    if $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] := \begin{cases} 1 & \text{with probability } p \\ 0 & \text{o.w.} \end{cases}$
    return $\Gamma[\mathcal{P}, t]$
On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B})$ from $\mathcal{P}$: let $t$ be the current time:
    assert $\mathsf{chain} \in \mathsf{tree}$, $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$, and $\texttt{leader}(\mathcal{P}, t)$ outputs 1
    append $\mathsf{B}$ to $\mathsf{chain}$ in $\mathsf{tree}$, 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 $\mathsf{B}$ to $\mathsf{chain}$ in $\mathsf{tree}$, 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 $\mathcal{F}_{\text{tree}}$.

Theorem 6. *In the sleepy model, any protocol that does not take an upper bound on the network delay Δ as input cannot realize state machine replication even when all awake nodes are honest (and the adversary therefore is merely a network adversary).*

Proof. Consider any such protocol that has no foreknowledge of Δ. Consider the following adversary $\mathcal{A}$: it does not corrupt any nodes or make any nodes sleep; however, it divides the alert nodes into two camps, with a large $\Delta = \mathsf{poly}(\lambda, N)$ in between the two camps.

After executing the protocol for some $\mathsf{poly}(\lambda, N)$ time, due to the requirement of achieving liveness even when a polynomial fraction of the nodes are sleeping, alert nodes in both camps must output a non-empty LOG — since nodes in one camp cannot distinguish if there is a long network delay between the camps, or if the other camp has fallen asleep. However, if the environment $\mathcal{Z}$ sent different inputs to the nodes in the two camps, their output LOGs will be different. This breaks consistency. $\square$

7 Proofs for Static Corruption and Static Sleepiness

In this section, we start by analyzing a very simple ideal protocol denoted Π_{ideal}, where nodes interact with an ideal functionality $\mathcal{F}_{\text{tree}}$ that keeps track of all valid chains at any moment of time. Later in Section 7.8, we will show that the real-world protocol Π_{sleepy} securely emulates the ideal-world protocol.

7.1 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. $\mathcal{F}_{\text{tree}}$ flips random coins to decide whether a node is the elected leader for every time step, and an adversary $\mathcal{A}$ can query this information through the `leader` query interface. Finally, alert and corrupt nodes can call $\mathcal{F}_{\text{tree}}.\texttt{extend}$ to extend known chains with new blocks if

Protocol Π_{ideal}

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

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

Every time step:

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

Figure 4: Ideal protocol Π_{ideal}

they are the elected leader for a specific time step. $\mathcal{F}_{\text{tree}}$ keeps track of all valid chains, such that alert nodes will call $\mathcal{F}_{\text{tree}}.\texttt{verify}$ to decide if any chain they receive is valid. Alert nodes always store the longest valid chains they have received, and try to extend it.

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

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

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

Theorem 7 (Security of Π_{ideal})**.** *For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \lambda$, Π_{sleepy} satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2-consistency against any Π_{ideal}-compliant,* **computationally unbounded** *pair $(\mathcal{A}, \mathcal{Z})$, with $\exp(-\Omega(\lambda))$ failure probability and the following parameters:*

- chain growth lower bound parameter $g_0 = (1 - \epsilon)(1 - 2pN\Delta)\alpha$;
- chain growth upper bound parameter $g_1 = (1 + \epsilon)Np$; and
- chain quality parameter $\mu = 1 - \frac{1-\epsilon}{1+\phi}$.

We will now prove the above Theorem 7.

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

Although our $\mathcal{F}_{\text{tree}}$ allows the adversary to claim potentially false block-times, we can rely on the following block-time invariants in our proofs: 1) honest blocks always have faithful block-times;

and 2) any chain in $\mathcal{F}_{\text{tree}}$ must have strictly increasing block-times. Having observed these, we show that Pass et al.'s chain growth and chain quality proofs [36] can be easily adapted for our scenario.

Unfortunately, the main challenge is how to prove consistency. As mentioned earlier, our adversary is much more powerful than the adversary for the Nakamoto blockchain and can launch a much wider range of attacks where he reuses the time slots during which he is elected. In Sections 7.5 and 7.6, we present new techniques for analyzing the induced stochastic process.

7.2 Convergence Opportunities

We now define a useful pattern called convergence opportunities, which we shall later use in both our chain growth lower bound proof as well as consistency proof.

Convergence opportunity. Given a view, suppose $T \leq |\mathsf{view}| - \Delta$, we say that $[T - \Delta, T + \Delta]$ is a convergence opportunity iff

- For any $t \in [\max(0, T - \Delta), T)$, no node alert at time t is elected leader;
- A single node alert at T is elected leader at time T;
- For any $t \in (T, T + \Delta]$, no node alert at time t is elected leader.

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

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

Henceforth, we use the notation $\mathbf{C}(\mathsf{view})[t' : t]$ to denote the number of convergence opportunities contained within the window $[t' : t]$ in view.

Many convergence opportunities. We now show that convergence opportunities happen sufficiently often.

Lemma 2 (There are many convergence opportunities)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any $0 < t_0 \leq t_1 \leq |view| - \Delta$, any positive λ, any positive constant ϵ, there exists a constant ϵ' that depends on ϵ, the following holds where $t := t_1 - t_0$:*

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

Proof. We prove this lemma in a similar fashion as Pass et al. [36]. Consider some view, and imagine that $\mathcal{F}_{\text{tree}}$ flips $\mathsf{alert}^r(\mathsf{view})$ coins for alert nodes (henceforth referred to as alert coins for short) in some time step r, where $\mathsf{alert}^r(\mathsf{view})$ denotes the number of alert nodes in time step r in view. Henceforth, we imagine all these alert coins are sequentialized.

- Let $\mathbf{X}$ denote the total number of heads in all the alert coins during $[t_0, t_1]$. Due to the Chernoff bound, it is not hard to see that for any $\epsilon > 0$, it holds that

$$\Pr[\mathbf{X} > (1 - \epsilon) \cdot \alpha t] \leq \exp(-\Omega(\alpha t))$$

Henceforth let $L := (1 - \epsilon) \cdot \alpha t$ for a sufficiently small constant ϵ.

- Let $\mathbf{Y}_i = 1$ iff after the i-th heads in the alert coin sequence during $[t_0, t_1]$, there exists a heads in the next $\alpha\Delta$ coin flips. Notice that all of the $\mathbf{Y}_i$'s are independent — to see this, another way to think of $\mathbf{Y}_i$ is that $\mathbf{Y}_i = 1$ iff the i-th coin flip and the $(i+1)$-th coin flip are at least $\alpha\Delta$ apart from each other.

  Let $\mathbf{Y} := \sum_{i=1}^{L} \mathbf{Y}_i$. We have that

$$\mathbf{E}[\mathbf{Y}] \leq (1 - (1-p)^{N_{\text{alert}}\Delta}) \cdot L \leq pN_{\text{alert}}\Delta \cdot L = \alpha\Delta L$$

  By Chernoff bound, it holds that for any $\epsilon_0 > 0$,

$$\Pr[\mathbf{Y} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\Omega(L)) = \exp(-\Omega(\alpha t))$$

- Let $\mathbf{Z}_i = 1$ iff before the i-th heads in the alert coin sequence during $[t_0, t_1]$, there exists a heads in the previous $\alpha\Delta$ coin flips. Similar as before, all of the $\mathbf{Z}_i$'s are independent. Let $\mathbf{Z} := \sum_{i=1}^{L} \mathbf{Z}_i$. We have that

$$\mathbf{E}[\mathbf{Z}] \leq (1 - (1-p)^{N_{\text{alert}}\Delta}) \cdot L \leq pN_{\text{alert}}\Delta \cdot L = \alpha\Delta L$$

  By Chernoff bound, it holds that for any $\epsilon_0 > 0$,

$$\Pr[\mathbf{Z} > \alpha\Delta L + \epsilon_0 L] \leq \exp(-\Omega(L)) = \exp(-\Omega(\alpha t))$$

- Observe that

$$\mathbf{C}(\mathsf{view})[t_0 : t_1] \geq \mathbf{X}(\mathsf{view}) - \mathbf{Y}(\mathsf{view}) - \mathbf{Z}(\mathsf{view})$$

  Observe that our compliance rule implies that $\alpha\Delta \leq pN\Delta < \frac{1}{2}$. For any view where the aforementioned relevant bad events do not happen, we have that for any $\epsilon' > 0$, there exist sufficiently small positive constants ϵ_0 and ϵ such that the following holds:

$$\begin{aligned}\mathbf{X} - \mathbf{Y} - \mathbf{Z} \geq &(1 - 2\alpha\Delta - 2\epsilon_0)L = (1 - 2\alpha\Delta - 2\epsilon_0) \cdot (1 - \epsilon) \cdot \alpha t \\ \geq &(1 - \epsilon')(1 - 2\alpha\Delta) \cdot \alpha t \\ \geq &(1 - \epsilon')(1 - 2pN\Delta) \cdot \alpha t\end{aligned}$$

  Observe that there are at most $\exp(-\Omega(\alpha t))$ fraction of bad view that we could have ignored.

□

7.3 Chain Growth Lower Bound

Fact 1. For any view, any t_0, any $t_1 \geq t_0$, it holds that

$$\mathbf{C}(\mathsf{view})[t_0 : t_1 - \Delta] \leq \mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t_1]$$

where $\mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t_1]$ is the length of the shortest honest chain at the beginning of time step t_1 minus the length of the longest honest chain at the beginning of time step t_0 in view.

Proof. By simple induction: given any view, any t_0, suppose that the fact holds for any $t_1 \leq t^*$. We now show that it holds for time $t_1 = t^* + 1$ as well. If time $t^* - \Delta + 1$ does not correspond to a convergence opportunity, the induction step is trivial. Otherwise, if time $t^* - \Delta + 1$ corresponds to a convergence opportunity, by definition of convergence opportunity, we have that

$$\mathbf{C}(\mathsf{view})[t_0 : t^* + 1 - \Delta] = \mathbf{C}(\mathsf{view})[t_0 : t^* - \Delta] + 1 = \mathbf{C}(\mathsf{view})[t_0 : t^* + 1 - 2\Delta] + 1$$

By induction hypothesis, we have that

$$\mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t^* + 1 - \Delta] + 1 \geq \mathbf{C}(\mathsf{view})[t_0 : t^* + 1 - 2\Delta] + 1 = \mathbf{C}(\mathsf{view})[t_0 : t^* + 1 - \Delta] \quad (1)$$

Additionally, we have that at the end of time step $t^*+1-\Delta$, there is an honest chain whose length is at least $\mathsf{min\_alert\_len}^{t^*+1-\Delta}(\mathsf{view}) + 1$, where $\mathsf{min\_alert\_len}^{t^*+1-\Delta}(\mathsf{view})$ denotes the length of the shortest alert chain at the beginning time $t^* + 1 - \Delta$. Since network delay is bounded by Δ, at the beginning of time time $t^* + 1$, every alert node's chain must be at least $\mathsf{min\_alert\_len}^{t^*+1-\Delta}(\mathsf{view}) + 1$ blocks long. In other words, we have that

$$\mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t^* + 1] \geq \mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t^* + 1 - \Delta] + 1 \quad (2)$$

The remainder of the induction step follows directly from Equations 1 and 2. □

Lemma 3 (Chain growth lower bound)**.** *For any t and any positive constant ϵ, except with* $\exp(-\Omega(\alpha t))\mathsf{poly}(\lambda)$ *probability over the choice of* view *of a compliant execution, it holds that for any* t_0*,* $\mathsf{min\_chain\_increase}(\mathit{view})[t_0 : t_0 + t] \geq (1 - \epsilon)(1 - 2pN\Delta)\alpha t - 1$.

Proof. Due to Fact 1 and Lemma 2, ignore the $\exp(-\Omega(\alpha t))\mathsf{poly}(\lambda)$ fraction of bad views, for every remaining good view, it holds that for every positive constant ϵ,

$$\begin{aligned} &\mathsf{min\_chain\_increase}(\mathsf{view})[t_0 : t_0 + t] > (1 - \epsilon)(1 - 2pN\Delta)\alpha(t - \Delta) \\ = (1 - \epsilon)(1 - 2pN\Delta)\alpha t &- (1 - \epsilon)(1 - 2pN\Delta)\alpha\Delta \geq (1 - \epsilon)(1 - 2pN\Delta)\alpha t - 1 \end{aligned}$$

where the last inequality is due to the fact $\alpha\Delta < 2pN\Delta < 1$ which stems from the compliance rules. □

7.4 Chain Quality

We now prove chain quality. To do this, we adapt the chain quality proof of Pass et al. [36] to our setting.

Adversarial block upper bound. Given a view, let $\mathbf{A}(\mathsf{view})[t_0 : t_1]$ denote the number of time steps in which at least one corrupt node is elected leader during the window $[t_0 : t_1]$. Let $\mathbf{A}^t(\mathsf{view})$ denote the *maximum* number of time steps in which at least one corrupt node is elected leader in any t-sized window in view.

Fact 2 (Upper bound on adversarial time slots)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any t, any $t_0 \leq t_1 \leq |\mathsf{view}|$ such that $t_1 - t_0 = t$, for any constant $0 < \epsilon < 1$ and any λ,

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

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

Fact 3 (Upper bound on adversarial time slots)**.** For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any t, for any constant $0 < \epsilon < 1$ and any λ,

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

Proof. Straightforward by Fact 2 and taking union bound over all possible time steps in view. □

Lemma 4 (Chain quality)**.** *For any positive constant ϵ, for all but $\exp(-\Omega(T))\mathsf{poly}(\lambda)$ fraction of view, the chain quality over any T-sized window in any honest chain is lower bounded by $\mu := 1 - \frac{1+\epsilon}{1+\phi}$.*

Proof. Let r be any time step, let i be any node honest at $r \leq |\mathsf{view}|$. Consider an arbitrary honest chain $\mathsf{chain} := \mathsf{chain}_i^r(\mathsf{view})$, and an arbitrary sequence of T blocks $\mathsf{chain}[j+1..j+T] \subset \mathsf{chain}_i^r$, such that $\mathsf{chain}[j]$ is not adversarial (either an honest block or genesis); and $\mathsf{chain}[j+T+1]$ is not adversarial either (either an honest block or $\mathsf{chain}[j+T]$ is end of chain_i^r). Note that if a sequence of blocks is not sandwiched between two honest blocks (including genesis or end of chain), we can always expand the sequence to the left and right to find a maximal sequence sandwiched by honest blocks (including genesis or end of chain). Such an expansion will only worsen chain quality.

For an honest block, its block-time must be faithful, i.e., corresponding to the time step in which the block was mined (recall that the block-time of $\mathsf{genesis}$ is 0). Consequently, by definition of Π_{ideal} and $\mathcal{F}_{\text{tree}}$, the block-times of all blocks in $\mathsf{chain}[j+1..j+T]$ must be bounded in between r' and $r'+t$, where r' denotes the time step in which the honest (or genesis) block $\mathsf{chain}[j]$ was mined, and $r'+t$ denotes the time step in which $\mathsf{chain}[j+T+1]$ is mined (or let $r'+t := r$ if $\mathsf{chain}[j+T]$ is end of chain_i^r).

We ignore any views where bad events related to chain growth lower bound or adversarial block upper bound take place. The fraction of views ignored is upper bounded $\exp(-\Omega(T)) \cdot \mathsf{poly}(\lambda)$.

- Now, due to chain growth lower bound, for any positive constant ϵ, we have that

$$t < \frac{T}{(1-\epsilon)(1-2pN\Delta)\alpha}$$

- Due to adversarial block upper bound, for any positive constant $\epsilon'' > 0$, there exists a sufficiently small positive constants ϵ' (which depends on ϵ, ϵ'', and ϕ), such that

$$\begin{aligned}\mathbf{A}[r' : r'+t] \leq & \mathbf{A}[r' : r' + \frac{T}{(1-\epsilon)(1-2pN\Delta)\alpha}] \\ \leq & \frac{(1+\epsilon')\beta T}{(1-\epsilon)(1-2pN\Delta)\alpha} \\ \leq & \frac{(1+\epsilon')(1-2pN\Delta)T}{(1-\epsilon)(1-2pN\Delta)(1+\phi)} \\ \leq & \frac{(1+\epsilon'')T}{1+\phi}\end{aligned}$$

- Therefore, the fraction of honest blocks in this length T sequence is lower bounded by

$$1 - \frac{1+\epsilon''}{1+\phi}$$

□

7.5 Consistency: Proof Intuition

Since this is the most non-trivial part of our proof and where we significantly depart from earlier blockchain proofs [20, 36], we will first explain the intuition before presenting the formal proof.

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

Now, Pass et al. [36] observes that for the adversary to cause divergence at some time s or earlier, for every convergence opportunity after time s, the adversary must mine a chain of length $\ell + 1$ where ℓ is the maximum chain length of any honest node at the beginning of the convergence period. This means that from time

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

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

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

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

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

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

7.6 Consistency: the Proof

Recall that a convergence opportunity is formally defined in Section 7.2, and that we use the notation $\mathbf{C}(\mathsf{view})[t' : t]$ to denote the number of convergence opportunities contained within the window $[t' : t]$ in view. Further, in Lemma 2, we showed that there are many convergence opportunities in a way similar to Pass et al. [36]. We remark that this lemma was the core technical contribution of Pass et al. [36]'s consistency proof. Our proof will build upon their conclusions — but as we show, on top of their analayis, we would need non-trivial new techniques to prove consistency for Sleepy.

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

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

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

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

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

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

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

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

Lemma 5 (Divergence cannot happen before a pivot)**.** *Let view be any execution trace where the bad events related to Lemma 2 do not happen. Let i be honest at time r and j be honest at $r' \geq r$ in view, if $0 < t < r - \frac{\lambda}{\beta}$ is a pivot in view, then chain_i^r and $\mathsf{chain}_j^{r'}$ cannot diverge at t in view.*

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

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

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

Suppose that chain_i^r and $\mathsf{chain}_j^{r'}$ diverge at t, where $t < r - \frac{\lambda}{\beta}$ is a pivot. By Lemma 2, there must be at least one convergence opportunity between the window (t, r). Obviously this also guarantees that there are blocks whose block-times are greater than t in the post-processed chain_i^r and $\mathsf{chain}_j^{r'}$. We now look at the first block $\mathsf{B} \in \mathsf{chain}_i^r$ (or $\mathsf{chain}_j^{r'}$) whose block-time is greater than or equal to t, and we argue that this block must be honest.

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

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

Lemma 6 (Adversarial time slots vs. convergence opportunities)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists some positive constant η, such that for any positive λ, for any $0 < t_0 < t_1 \leq |\mathit{view}|$ satisfying $t := t_1 - t_0 \geq c'\Delta$ for a sufficiently large constant c', we have that*

$$\Pr\left[\mathit{view} \leftarrow_{\$} \mathit{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathbf{A}(\mathit{view})[\max(0, t_0 - \Delta) : t_1] \geq \mathbf{C}(\mathit{view})[t_0 : t_1]\right] < \exp(-\eta\beta t)$$

and

$$\Pr\left[\mathit{view} \leftarrow_{\$} \mathit{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathbf{A}(\mathit{view})[t_0 : \min(t_1 + \Delta, |\mathit{view}|)] \geq \mathbf{C}(\mathit{view})[t_0 : t_1]\right] < \exp(-\eta\beta t)$$

In other words, the lemma says that for any fixed time window, except with negligible probability, the adversary always has less elected time slots than there are convergence opportunities — even if the adversary is given an extra Δ time steps at the beginning or end.

Proof. We prove one of the above cases with the extra Δ given to the adversary at the beginning. The other case is similar. Due to Fact 2, for any positive ϵ_1,

$$\Pr\left[\mathbf{A}[\max(0, t_0 - \Delta) : t_0 + t] > (1 + \epsilon_1)\beta(t + \Delta)\right] < \exp(-\frac{\epsilon_1^2 \beta t}{3})$$

Due to Lemma 2, for any positive ϵ_2, there exists positive ϵ' that depends on ϵ_2, such that

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

Since we know that

$$\frac{\alpha}{\beta} > \frac{1+\phi}{1-2pN\Delta}$$

and moreover $2\beta\Delta < 2pN\Delta < 1$, it holds that for sufficiently small constants ϵ_1 and ϵ_2, and $t \geq c' \cdot \Delta$ for a sufficiently large constant c',

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

The rest of the proof is straightforward. □

Fact 5. Let $t' < t$. If for every non-negative integer k, $\mathbf{C}(\mathsf{view})[t'-k\Delta : t] > \mathbf{A}(\mathsf{view})[t'-(k+1)\Delta : t]$ or $\mathbf{A}(\mathsf{view})[t-(k+1)\Delta : t] = 0$, then, it holds that for any $r \leq t'$,

$$\mathbf{A}(\mathsf{view})[r : t] < \mathbf{C}(\mathsf{view})[r : t] \text{ or } \mathbf{A}(\mathsf{view})[r : t] = 0$$

In the above, if the array index is negative, it is automatically rounded up to 0.

Intuitively, the above fact says that if we would like to check if t is a backward (or forward resp.) pivot in view, we only need to check every Δ steps as long as we give the adversary Δ extra for every window we check. This fact will be used later in the proof of Lemma 7 when we take a union bound (doing it step by step will lose too much in the union bound, but doing it every Δ steps will give a tight enough bound).

Proof. Basically for every $s \in [t'-(k+1)\Delta, t'-k\Delta]$, we use $\mathbf{C}(\mathsf{view})[t'-k\Delta : t]$ as a lower bound of $\mathbf{C}(\mathsf{view})[s : t]$; and we use $\mathbf{A}(\mathsf{view})[t'-(k+1)\Delta : t]$ as an upper bound of $\mathbf{A}(\mathsf{view})[s : t]$. The rest of the proof is straightforward. □

Lemma 7 (Probability that any given time is a backward pivot)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a positive constant c, such that for any positive λ, for any $0 < t \leq |\mathsf{view}|$,*

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

Proof. As before, in the following proof, if array indices are negative, they are automatically rounded up to 0. For simplicity, for $t' < t$, let $\mathsf{bad}(t')$ denote the bad event that $\mathbf{C}[t' : t] \leq \mathbf{A}[t'-\Delta : t]$. Let $t_c := \max(1, t - \frac{c_1}{\beta\eta})$ where c_1 is a suitable constant and η is the positive constant corresponding to Lemma 6. Observe also since $2\beta\Delta < 2pN < 1$ and hence $\beta < 0.5$, it holds that $(1-\beta)^{\frac{1}{\beta}} > 0.25$.

We now have the following:

$$\begin{aligned}
&\Pr[t \text{ is a backward pivot}] \geq \Pr[t \text{ is a backward pivot and } \mathbf{A}[t_c : t] = 0] \\
&\geq \Pr[\mathbf{A}[t_c : t] = 0] \cdot \Pr\left[\text{for any } t' < t_c\text{: } \overline{\mathsf{bad}}(t') \,|\mathbf{A}[t_c : t] = 0\right] \\
&\geq \Pr[\mathbf{A}[t_c : t] = 0] \cdot \Pr\left[\text{for any } t' < t_c\text{: } \overline{\mathsf{bad}}(t')\right] \\
&\geq \left((1-\beta)^{\frac{1}{\beta}}\right)^{\frac{c_1}{\eta}} \cdot (1 - \Pr[\mathsf{bad}(t_c)] - \Pr[\mathsf{bad}(t_c-\Delta)] - \Pr[\mathsf{bad}(t_c-2\Delta)]\ldots) && \text{union bound, Fact 5} \\
&\geq \left(\frac{1}{4}\right)^{\Theta(1)} \cdot \left(1 - e^{-c_1} - e^{-c_1+\eta\beta\Delta} - e^{-c_1+2\eta\beta\Delta} - \ldots\right) && \text{Lemma 6, } c_1 \text{ sufficiently large const} \\
&= \left(\frac{1}{4}\right)^{\Theta(1)} \cdot \left(1 - \frac{e^{-c_1}}{1-e^{-\eta\beta\Delta}}\right)
\end{aligned}$$

Since $\beta\Delta = \Theta(1)$, as long as we pick constant c_1 such that $e^{-c_1} < 1 - e^{-\eta\beta\Delta}$, the last line above is a constant greater than 0. $\square$

Lemma 8 (Probability that any given time is a forward pivot)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a positive constant c, such that for any positive λ, for any $0 < t \leq |\mathsf{view}|$,*

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

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

Corollary 3 (Probability that any time step is a pivot)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, there exists a constant c, such that for any positive λ, for any $0 < t \leq |\mathsf{view}|$,*

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

Proof. At a very high level, we just need to show the probability that t is both a backward and forward pivot. There is a small technical issue, though, since the event that t is a backward pivot and the event that t is a forward pivot are not independent. Fortunately this is not too difficult to deal with. Roughly speaking, the event that t is a backward pivot and the event that $t + 2\Delta$ is a forward pivot are independent, because the former event depends only on the coins till $t + \Delta$, and the latter event depends only on the coins after $t + \Delta$ — additionally, we may require that no corrupt or alert node is elected leader during this $[t, t + 2\Delta]$ window which happens with constant probability. If it is the case that t is a backward pivot and $t + 2\Delta$ is a forward pivot, and moreover, no corrupt or alert node is elected leader during this $[t, t + 2\Delta]$ window, then it holds that t is a pivot. The following proof formalizes this intuition:

Let $t_c = 2\Delta$, since $2Np\Delta < 1$, $2\beta < Np$, and $\Delta \geq 1$, we have that $(1 - \beta)^{t_c} \geq 0.25$. Further, due to our compliance rule, we have that $2\alpha\Delta < 2Np\Delta < 1$, and therefore $(1 - \alpha)^{t_c} \geq 0.25$.

$$\begin{aligned}
&\Pr[t \text{ is a pivot}] \\
\geq &\Pr\left[\begin{array}{l} t \text{ is a backward pivot and } \mathbf{A}[t : t + t_c] = 0 \\ \text{and no alert node is leader in } [t : t + t_c] \text{ and } t + t_c \text{ is a forward pivot} \end{array}\right] \\
\geq &\Pr[t \text{ is a backward pivot}] \cdot \Pr[\mathbf{A}[t : t + t_c] = 0] \cdot \Pr[\text{no alert node is leader in } [t : t + t_c]] \\
&\cdot \Pr[t + t_c \text{ is a forward pivot } \mid \text{no alert node is leader in } [t : t + t_c]] \\
\geq &\Pr[t \text{ is a backward pivot}] \cdot (1 - \beta)^{t_c} \cdot (1 - \alpha)^{t_c} \cdot \Pr[t + t_c \text{ is a forward pivot}] \\
= &\Theta(1)
\end{aligned}$$

Note that the above proof makes use of the following simple facts which are not hard to see:

$$\Pr[t + t_c \text{ is a forward pivot } \mid \text{no alert node is leader in } [t : t + t_c]] \geq \Pr[t + t_c \text{ is a forward pivot}]$$

Further, $\mathbf{A}[t : t{+}t_c] = 0$ is independent of the event that $t + t_c$ is a forward pivot and the event that no alert node is leader in $[t : t + t_c]$. Further, the event that t is a backward pivot is independent of $\mathbf{A}[t : t + t_c]$ and the event that $t + t_c$ is a forward pivot (since $t_c = 2\Delta$). And finally,

$$\Pr[t \text{ is a backward pivot}] \geq \Pr[t \text{ is a backward pivot} \mid \text{no alert node is leader in } [t : t + t_c]]$$

$\square$

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

Theorem 8 (There are many pivot points)**.** *For any Π_{ideal}-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$, for any λ, the following holds where $w = \frac{2\lambda}{\beta}$:*

$$\Pr\left[\mathit{view} \leftarrow_\$ \mathit{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{many\text{-}pivots}^w(\mathit{view}) = 1\right] < \exp(-\Omega(\sqrt{\lambda})) \cdot \mathsf{poly}(\lambda)$$

We now prove the above theorem.

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

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

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

Due to Lemma 6, it is not hard to see that only $\exp(-\Omega(\sqrt{\lambda}))$ fraction of views are bad.

Fact 6. Conditioned on views that are not considered bad by the above definition for windows of length at least $\frac{\sqrt{\lambda}}{\beta}$, every $\mathbf{G}_i$ is independent of $\{\mathbf{G}_j\}_{j \neq i}$.

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

We now return to the proof of Theorem 8. Let $\ell := \lfloor\frac{(r-s)\beta-\lambda}{3\sqrt{\lambda}}\rfloor$. Now, conditioned on good views where the aforementioned bad events do not happen, we have the following due to independence:

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

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

$$\Pr[\overline{\mathbf{G}}_1, \ldots, \overline{\mathbf{G}}_\ell] \leq \exp(-\Omega(\sqrt{\lambda}))$$

Proof of consistency. Ignore the negligible fraction of views where the bad events we care about happen. By Lemma 5 and the above theorem, let $s = r - \frac{3\lambda}{\beta}$, then chain_i^r and $\mathsf{chain}_j^{r'}$ must diverge after s. Let $\mathsf{chain}_i^r\langle < s\rangle$ and $\mathsf{chain}_j^{r'}\langle < s\rangle$ denote the prefix of the chains with block-time less than s. It holds that $\mathsf{chain}_i^r\langle < s\rangle = \mathsf{chain}_j^{r'}\langle < s\rangle$. Finally, due to a simple application of Chernoff bound, at most $(1+\epsilon)Np(r-s) = \frac{3(1+\epsilon)Np\lambda}{\beta} = O(\lambda)$ blocks in chain_i^r can have block-times between s and r.

7.7 Chain Growth Upper Bound

Below we adapt Pass et al. [36] chain growth upper bound proof to our setting.

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

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

Lemma 9 (No long block withholding)**.** *For every Π_{ideal}-compliant p.p.t. $(\mathcal{A}, \mathcal{Z})$ pair, for every constant $0 < \epsilon < 1$, we have that*

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

Proof. Given our new definition of $\mathsf{withhold\text{-}time}$, we can prove exactly the same "no long block withholding" lemma as Pass et al. [36], using our new definition of $A_t(\mathsf{view})$ in Section 7.4. We omit the full proof since the proof is almost identical to that of Pass et al. [36]. □

Lemma 10 (Chain growth upper bound)**.** *For any t, any positive constant $\epsilon > 0$, except with $\exp(-\Omega(\beta t)) \cdot \mathsf{poly}(\lambda)$ probability over the choice of view of a compliant execution, it holds that for any t_0, $\mathsf{max\_chain\_increase}(\mathit{view})[t_0 : t_0{+}t] \leq (1{+}\epsilon)Npt{+}1$, where $\mathsf{max\_chain\_increase}(\mathit{view})[t_0 : t_0{+}t]$ denotes the length of the shortest honest chain at time $t_0 + t$ minus the length of the longest honest chain at time t_0.*

Proof. Consider any fixed t_0 and $t_1 := t_0 + t$. Note that the maximum chain increase (as defined above) between t_0 and t_1 is upper bounded by the max-to-max honest chain growth between $t_0 - \Delta$ and t_1, where max-to-max chain growth between $t_0 - \Delta$ and t_1 refers to the difference in length between the longest honest chains at times $t_0 - \Delta$ and t_1 respectively. Henceforth let $t_0' := \max(0, t_0 - \Delta)$.

Observe that the max-to-max chain growth between t_0' and t_1 is upper bounded by the number of blocks that first appeared in an honest chain between t_0' and t_1. Let B denote any block that first appeared in an honest chain between t_0' and $t_1 \leq t_0' + t + \Delta$, due to the "no long block witholding lemma", for any positive constant ϵ', except with $\exp(-\Omega(\beta t))\mathsf{poly}(\lambda)$ probability, the block B must have a block-time between $t_0' - \epsilon' t$ and t_1. Due to Chernoff bound, for any ϵ'', it holds that except with $\exp(-\Omega(Npt))$ probability, during the window $[t_0' - \epsilon' t, t_1]$, the maximum number of distinct time steps in which any alert or corrupt is elected leader is less than $(1 + \epsilon'')Np(1 + \epsilon')(t + \Delta)$.

Observe that for any positive constant $0 < \epsilon < 1$, there exist positive constants ϵ' and ϵ'' such that $(1 + \epsilon'')Np(1 + \epsilon')(t + \Delta) \leq (1 + \epsilon)Np(t + \Delta) = (1 + \epsilon)Npt + (1 + \epsilon)Np\Delta \leq (1 + \epsilon)Npt + 1$ where the last inequality arises from the fact $2Np\Delta < 1$ which is implied by our compliance rules.

Finally, the above proof assumes a fixed t_0, we can now conclude the proof by taking a union bound over the choice of t_0. □

7.8 Real World Emulates the Ideal World

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

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

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

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

  If not, $\mathcal{S}$ checks if $\mathcal{P}$ is a party identifier corresponding to this protocol instance. If not, $\mathcal{S}$ generates a random number of appropriate length and returns it. Else if the mapping succeeds, $\mathcal{S}$ queries $b \leftarrow \mathcal{F}_{\text{tree}}.\mathtt{leader}(\mathcal{P}, t)$. If $b = 1$, $\mathcal{S}$ rejection samples a random string h of appropriate length, until $h < D_p$; it then returns h. Else if $b = 0$, $\mathcal{S}$ rejection samples a random string h of appropriate length, until $h \geq D_p$; it then returns h.

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

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

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

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

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

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

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

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

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

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

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

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

Fact 8. The simulated execution never aborts with $\mathsf{sig\text{-}failure}$ except with negligible probability.

Proof. We ignore all views where the bad event $\mathsf{duplicate\text{-}block\text{-}failure}$ happens.

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

Lemma 11 (Indistinguishability)**.** *Conditioned on the fact that all of the aforementioned bad events do not happen, then the simulated execution is identically distributed as the real-world execution from the perspective of $\mathcal{Z}$.*

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

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

Fact 9. If $(\mathcal{A}, \mathcal{Z})$ is Π_{sleepy}-compliant, then $(\mathcal{S}^{\mathcal{A}}, \mathcal{Z})$ is Π_{ideal}-compliant.

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

Finally, since the simulated execution is compliant, it respects all the desired properties as Theorem 7 states. Now, since real-world execution and the simulated execution are indistiguishable, it holds that all the desired properties hold in the same way for the real-world execution. We therefore complete the proof of our main theorem, that is, Theorem 3 of Section 5.

7.9 Removing the Random Oracle in the Proof

It is not hard to modify the proof when we remove the random oracle, and instead use $\mathsf{PRF}_{k_0}(\mathcal{P}, t) < D_p$ as the leader election function, where k_0 is a random string to be included in the common reference string. We state the modifications necessary to the proof below:

- First, we introduce an intermediate hybrid protocol where the ideal functionality $\mathcal{F}_{\text{tree}}$ selects k_0 at random prior to protocol start, and discloses k_0 to the adversary $\mathcal{A}$. Meanwhile, instead of generating random bits to determine leader for both honest and corrupt nodes, the ideal functionality $\mathcal{F}_{\text{tree}}$ instead uses $\mathsf{PRF}_{k_0}(\mathcal{P}, t) < D_p$.

  We can argue that such a hybrid protocol is also secure against computationally unbounded, compliant $(\mathcal{A}, \mathcal{Z})$. In particular, observe that in our previous ideal protocol analysis, once we fix the random bits $\vec{\nu}$ of the random oracle (RO), we can define certain bad events (that depend only on the random bits of the random oracle, but those not of $(\mathcal{A}, \mathcal{Z})$). Provided that these bad events do not happen, even a computationally unbounded $(\mathcal{A}, \mathcal{Z})$ cannot break the chain growth, chain quality, or consistency properties. Further, observe that there is a polynomial-time algorithm that can efficient check for bad events given the random bits of the random oracle.

Therefore, when we replace the random oracle with $\mathsf{PRF}_{k_0}(\cdot)$, over the probability space defined over the choice of k_0, these bad events should not happen except with negligible probability as well — otherwise the algorithm that checks for the bad events can be used as an efficient adversary that distinguishes the PRF from the random oracle. Similarly, in the PRF case, as long as the bad events do not happen, even a computationally unbounded adversary should not be able to break the security properties.

- Now, we can modify our simulation proof to prove that the real-world protocol emulates the modified hybrid protocol as mentioned above. Most of the simulation proof is identical to the random oracle case presented above, except that now when the simulator learns k_0 from $\mathcal{F}_{\text{tree}}$, it simply gives k_0 to $\mathcal{A}$, and the simulator no longer needs to simulate random oracle queries for $\mathcal{A}$.

8 Proofs for Adaptive Sleepiness and Adaptive Corruption

We first describe how to prove security under adaptive sleepiness but static corruption: this will be the more interesting part of the proof, and to achieve this, we will need to rely on complexity leveraging, but in this case how to do complexity leveraging turns out to be rather subtle. Once we are able to do this, we then describe how to leverage additional, standard complexity leveraging techniques (Section 8.4) to upgrade the security to the case of adaptive sleepiness and corruption.

8.1 Ideal-World Protocol: Adaptive Sleepiness and Static Corruption

Ideal functionality $\mathcal{F}^*_{\text{tree}}$. In Figure 5, we modify the ideal functionality $\mathcal{F}_{\text{tree}}$ for static corruption (see Section 7) to $\mathcal{F}^*_{\text{tree}}$. The main difference between $\mathcal{F}_{\text{tree}}$ and $\mathcal{F}^*_{\text{tree}}$ is the highlighted blue line: in $\mathcal{F}_{\text{tree}}$, the adversary $\mathcal{A}$ is allowed to query the ideal functionality to check if anyone (including honest nodes) is elected leader at any time. However, in $\mathcal{F}^*_{\text{tree}}$, each party can only make such queries for itself. In other words, the adversary $\mathcal{A}$ can see into the future for corrupt parties but not for honest parties. In our new ideal protocol, the adversary $\mathcal{A}$ can only learn that an honest party $\mathcal{P}$ is elected for a time step t when $\mathcal{P}$ actually announces a valid new block in time step t.

Ideal protocol Π^*_{ideal}. The ideal protocol Π^*_{ideal} is identical to Π_{ideal} except that now $\mathcal{F}_{\text{tree}}$ is replaced with $\mathcal{F}^*_{\text{tree}}$.

Compliant executions: adaptive sleepiness and static corruption. A $\Pi^*_{\text{ideal}}(p)$-compliant p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ is defined in exactly the same way as a $\Pi^*_{\text{ideal}}(p)$-compliant $(\mathcal{A}, \mathcal{Z})$ except that now we allow $\mathcal{Z}$ to make nodes sleep adaptively. However, we require that $\mathcal{Z}$ still declares corruptions statically upfront.

Theorem 9 (Security of the protocol Π^*_{ideal} under adaptive sleepiness and static corruption)**.** *For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \lambda$, Π_{sleepy} satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2 consistency against any Π_{ideal}-compliant, computationally unbounded pair $(\mathcal{A}, \mathcal{Z})$ with $\exp(-\Omega(\lambda))$ failure probability and the following parameters:*

- *chain growth lower bound parameter $g_0 = (1 - \epsilon)(1 - 2pN\Delta)\alpha$;*

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

On init: $\mathsf{tree} := \mathsf{genesis}$, $\mathsf{time}(\mathsf{genesis}) := 0$
On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{P}$ itself or internally:
    if $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] := \begin{cases} 1 & \text{with probability } p \\ 0 & \text{o.w.} \end{cases}$
    return $\Gamma[\mathcal{P}, t]$
On receive $\texttt{extend}(\mathsf{chain}, \mathsf{B})$ from $\mathcal{P}$: let $t$ be the current time:
    assert $\mathsf{chain} \in \mathsf{tree}$, $\mathsf{chain}||\mathsf{B} \notin \mathsf{tree}$, and $\texttt{leader}(\mathcal{P}, t)$ outputs 1
    append $\mathsf{B}$ to $\mathsf{chain}$ in $\mathsf{tree}$, 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 $\mathsf{B}$ to $\mathsf{chain}$ in $\mathsf{tree}$, 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: Modified ideal functionality $\mathcal{F}^*_{\text{tree}}$.

- *chain growth upper bound parameter $g_1 = (1 + \epsilon)Np$; and*
- *chain quality parameter $\mu = 1 - \frac{1-\epsilon}{1+\phi}$;*

Proof. Notice that in comparison with Π_{ideal}, here our Π^*_{ideal} does not allow the adversary to see into future random bits of honest parties, however, we allow the adversary to adaptively make nodes sleep. It is not hard to observe that this change does not matter to the stochastic analysis for the Π_{ideal} protocol presented in Section 7, and the same proof still holds. □

8.2 Intermediate Hybrid Protocol

We make a few modifications to the ideal-world protocol Π^*_{ideal}, and introduce the following hybrid protocols.

Hybrid protocol $\Pi^{\langle 1 \rangle}_{\text{hyb}}$.
Recall that in the ideal-world protocol Π^*_{ideal}, the ideal functionality $\mathcal{F}^*_{\text{tree}}$ generates fresh coins to decide of a player is elected leader for a time step. In the hybrid protocol $\Pi^{\langle 1 \rangle}_{\text{hyb}}$, we modify $\mathcal{F}^*_{\text{tree}}$ to obtain a new $\mathcal{F}^{\langle 1 \rangle}_{\text{hyb}}$ that works as follows:

- Any any time during the protocol execution, $\mathcal{F}^{\langle 1 \rangle}_{\text{hyb}}$ allows the adversary $\mathcal{A}$ to specify what $k[\mathcal{P}]$ value to use for a corrupt party $\mathcal{P}$ (if one has not been chosen before).
- The function $\texttt{leader}(\mathcal{P}, t)$ is implemented as the following instead. On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{P}$ or internally: If $\Gamma[\mathcal{P}, t]$ has been populated, return $\Gamma[\mathcal{P}, t]$. Else,
  - if $\mathcal{P}$ is honest, choose $\Gamma[\mathcal{P}, t]$ at random as before, and return $\Gamma[\mathcal{P}, t]$.
  - else if $\mathcal{P}$ is corrupt: if $\mathcal{A}$ has not registered $k[\mathcal{P}]$ with $\mathcal{F}^{\langle 1 \rangle}_{\text{hyb}}$, return 0 (and without populating table Γ); else let $\Gamma[\mathcal{P}, t] := (\mathsf{H}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t) < D_p)$ where H denotes a random function, and return $\Gamma[\mathcal{P}, t]$.

- $\mathcal{F}_{\text{hyb}}^{\langle 1 \rangle}$ is otherwise identical to $\mathcal{F}_{\text{tree}}^{*}$.

The protocol $\Pi_{\text{hyb}}^{\langle 1 \rangle}$ is identical to Π_{ideal}^{*} except that the players interact with the new $\mathcal{F}_{\text{hyb}}^{\langle 1 \rangle}$ instead of $\mathcal{F}_{\text{tree}}^{*}$. We say that $(\mathcal{A}, \mathcal{Z})$ is $\Pi_{\text{hyb}}^{\langle 1 \rangle}(p)$-compliant iff the pair is $\Pi_{\text{ideal}}^{*}(p)$-compliant.

Note that the main difference between $\Pi_{\text{hyb}}^{\langle 1 \rangle}$ and Π_{ideal}^{*} is the following: in $\Pi_{\text{hyb}}^{\langle 1 \rangle}$, corrupt nodes can influence the choice of the coins used to decide whether corrupt nodes are leaders, by setting the values of $k[\mathcal{P}]$. In particular, the adversary can choose the values of $k[\mathcal{P}]$ after querying $\mathsf{H}(\mathcal{P}, \_)$ for varying t's for any corrupt party $\mathcal{P}$. Below, we argue that despite this ability, since the number of bits $\vec{k}_{\text{corrupt}} := \{k[\mathcal{P}] : \mathcal{P} \text{ corrupt}\}$ that can be controlled by the adversary is small, there is still a significantly large fraction of random strings H that are good even for the worst-case choice of $\vec{k}_{\text{corrupt}}$.

Claim 1 (Security of $\Pi_{\text{hyb}}^{\langle 1 \rangle}$)**.** For any $T_0 \geq cLN$ where c is an appropriate constant, protocol $\Pi_{\text{hyb}}^{\langle 1 \rangle}$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2 consistency against any $\Pi_{\text{hyb}}^{\langle 1 \rangle}$-compliant, *computationally unbounded* pair $(\mathcal{A}, \mathcal{Z})$, where g_0, g_1, μ are defined in the same way as in Theorem 9, and moreover, with security failure probability $\exp(-\Omega(LN))$.

Proof. We abuse notation and sometimes use H to denote the random string generated by $\mathcal{F}_{\text{hyb}}^{\langle 1 \rangle}$. We use the notation υ to denote the random bits $\mathcal{F}_{\text{hyb}}^{\langle 1 \rangle}$ generated to decide whether honest nodes are elected leaders.

Given a fixed $\vec{k}_{\text{corrupt}}$, we say that the random string (H, υ) is good for $\vec{k}_{\text{corrupt}}$, if in any view consistent with H, υ, and $\vec{k}_{\text{corrupt}}$, no bad events related to (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2-consistency occur where the paramters T_0, g_0, g_1, μ are as given in the theorem statement. In other words, (H, υ) is good for $\vec{k}_{\text{corrupt}}$ if the combination of H, υ, and $\vec{k}_{\text{corrupt}}$ does not permit any bad events.

Due to Theorem 9, for every fixed $\vec{k}_{\text{corrupt}}$ and an appropriate choice of c, all but e^{-LN} fraction of random strings (H, υ) are good for $\vec{k}_{\text{corrupt}}$.

Now by union bound over the choice of $\vec{k}_{\text{corrupt}}$, we conclude that at least $1 - e^{-LN} \cdot 2^{LN}$ fraction of random strings (H, υ) are good for all choices of $\vec{k}_{\text{corrupt}}$. $\square$

Hybrid protocol $\Pi_{\text{hyb}}^{\langle 2 \rangle}$. Almost identical to $\Pi_{\text{hyb}}^{\langle 1 \rangle}$ except that now, the new ideal functionality $\mathcal{F}_{\text{hyb}}^{\langle 2 \rangle}$ generates a random PRF key k_0, discloses it to $\mathcal{A}$; and further $\mathcal{F}_{\text{hyb}}^{\langle 2 \rangle}$ replaces calls to the random function $\mathsf{H}(\_, \_)$ with calls to $\mathsf{PRF}_{k_0}(\_, \_)$.

Claim 2 (Security of $\Pi_{\text{hyb}}^{\langle 2 \rangle}$)**.** Suppose that the PRF function with input length k is secure against all 2^{k^δ}-time adversaries for some fixed constant $\delta < 1$. Suppose that $L \geq \log^2 \lambda$, $L_0 := |k_0| \geq (2LN)^{\frac{1}{\delta}}$. Then, for any $T_0 \geq cLN$ where c is an appropriate constant, protocol $\Pi_{\text{hyb}}^{\langle 2 \rangle}$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2 consistency against any $\Pi_{\text{hyb}}^{\langle 2 \rangle}$-compliant, *computationally unbounded* pair $(\mathcal{A}, \mathcal{Z})$, where g_0, g_1, μ are defined in the same way as in Theorem 9, and moreover with security failure probability $\exp(-\Omega(LN))$.

Proof. Given a random or pseudorandom string $r \in \{0,1\}^{\mathsf{poly}(\lambda, N)}$ either sampled at random from $\mathsf{H}(\cdot)$, or generated from $\mathsf{PRF}_{k_0}(\cdot)$ for a randomly chosen k_0, and a random string υ corresponding to randomness used for honest leader election, and a fixed $\vec{k}_{\text{corrupt}}$, there is an algorithm running in time $\mathsf{poly}(\lambda, N)$ that checks if (r, υ) is good for $\vec{k}_{\text{corrupt}}$.

Therefore, given (r, v), there is an algorithm running in time $\mathsf{poly}(\lambda, N) \cdot 2^{LN}$ that can check if (r, v) is good for all $\vec{k}_{\text{corrupt}}$. Specifically, this algorithm brute-force enumerates all possible $\vec{k}_{\text{corrupt}}$, and checks if (r, v) is good for every $\vec{k}_{\text{corrupt}}$.

When the PRF's input length $L_0 = (2LN)^{\frac{1}{\delta}}$, clearly the above algorithm runs in time that is subexponential in the PRF's input length. Due to the subexponential hardness of PRF, it holds that

$$\begin{aligned} &\Pr\left[k_0 \leftarrow_{\$} \{0,1\}^{L_0}, r \leftarrow \mathsf{PRF}_{k_0}(\cdot), v \leftarrow_{\$} \{0,1\}^{\mathsf{poly}(N,\lambda)} : (r, v) \text{ good for every } \vec{k}_{\text{corrupt}}\right] \\ \leq &\Pr\left[r \leftarrow_{\$} \mathsf{H}, v \leftarrow_{\$} \{0,1\}^{\mathsf{poly}(N,\lambda)} : (r, v) \text{ good for every } \vec{k}_{\text{corrupt}}\right] - 2^{-L_0^{\delta}} \end{aligned}$$

Since otherwise, one can easily construct a reduction, such that when given a string r, the reduction generates a random v, and calls the above algorithm to check if (r, v) is good for all $\vec{k}_{\text{corrupt}}$ — in this way, the reduction can effectively distinguish whether r is truly random or pseudorandom, and thus break the security of the PRF.

□

Hybrid protocol $\Pi_{\text{hyb}}^{\langle 3 \rangle}$. $\Pi_{\text{hyb}}^{\langle 3 \rangle}$ is almost the same as $\Pi_{\text{hyb}}^{\langle 2 \rangle}$, except now the ideal functionality computes honest parties' random strings using pseudorandomness too, whereas earlier in $\Pi_{\text{hyb}}^{\langle 2 \rangle}$, the ideal functionality uses true randomness when deciding if honest parties are leaders.

More formally, in $\Pi_{\text{hyb}}^{\langle 3 \rangle}$, we modify the ideal functionality to obtain a new ideal functionality $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$ that works as follows:

- During initialization, $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$ generates a fresh $k[\mathcal{P}] \leftarrow_{\$} \{0,1\}^L$ for every honest player $\mathcal{P}$.
- Next, $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$ generates a random seed $k_0 \leftarrow_{\$} \{0,1\}^{L_0}$, and discloses k_0 to the adversary $\mathcal{A}$.
- At any time during the protocol execution, $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$ allows the adversary $\mathcal{A}$ to specify what $k[\mathcal{P}]$ value to use for a corrupt party $\mathcal{P}$ (if one has not been chosen before).
- The function `leader`$(\mathcal{P}, t)$ is implemented as the following instead. On receive `leader`$(\mathcal{P}, t)$ from $\mathcal{P}$ or internally: If $\Gamma[\mathcal{P}, t]$ has been populated, return $\Gamma[\mathcal{P}, t]$. Else,
  - if $\mathcal{P}$ is corrupt and $\mathcal{A}$ has not registered $k[\mathcal{P}]$ with $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$, then return 0 without populating the Γ table;
  - else, compute $\eta := \mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t)$, populate the table $\Gamma[\mathcal{P}, t] := (\eta < D_p)$, notify $\mathcal{A}$ of the tuple $(\mathcal{P}, t, \eta)$ and return $\Gamma[\mathcal{P}, t]$.
- $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$ is otherwise identical to $\mathcal{F}_{\text{hyb}}^{\langle 3 \rangle}$.

Recall that we use L to denote the input length of each player's PRF and all other cryptographic primitives. We now have the following claim.

Claim 3 (Security of $\Pi_{\text{hyb}}^{\langle 3 \rangle}$ under adaptive sleepiness and static corruption)**.** Assume that the PRF is subexponentially hard. Then, if there is a $\Pi_{\text{hyb}}^{\langle 3 \rangle}$-compliant $(\mathcal{A}, \mathcal{Z})$ running in time subexponential in L that can cause bad events related to chain growth, quality, or consistency to happen with probability ϵ in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{\langle 3 \rangle}}(\mathcal{A}, \mathcal{Z}, \lambda)$, then there exists a $\Pi_{\text{hyb}}^{\langle 2 \rangle}$-compliant $(\mathcal{A}', \mathcal{Z}')$ that can cause the same bad events to happen in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{\langle 2 \rangle}}(\mathcal{A}', \mathcal{Z}', \lambda)$ with probability $\epsilon - 2^{-L^{\delta}}$.

Proof. By straightforward reduction to the subexponential security of PRF — in particular, we can have a sequence of hybrids and replace each honest nodes' random coins one by one with pseudorandom bits. □

Hybrid protocol $\Pi_{\mathbf{hyb}}^{\langle 4\rangle}$. $\Pi_{\rm hyb}^{\langle 4\rangle}$ is almost identical to $\Pi_{\rm hyb}^{\langle 3\rangle}$ except that now, we modify the ideal functionality slightly as follows and obtain $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$:

- During initialization, the new $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$ will honestly compute commitments $k[\mathcal{P}]$ for every honest node $\mathcal{P}$, and send the committed value to $\mathcal{A}$.
- During initialization, the new $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$ will call $\mathsf{crs} \leftarrow \mathtt{gen}(1^L, \mathcal{L})$ and send crs to the adversary $\mathcal{A}$.
- The new $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$ allows $\mathcal{A}$ to additionally query $\mathtt{nizk}(\mathcal{P}, t')$ at time $t > t'$ and for an honest party $\mathcal{P}$. Upon such a query, if $\mathcal{P}$ was not elected a leader in time t', return $\perp$. Otherwise, $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$ computes $\eta := \mathsf{PRF}_{k[\mathcal{P}]}(t') \oplus \mathsf{PRF}_{k_0}(\mathcal{P}, t')$, and $\pi := \mathsf{NIZK}.\mathtt{prove}(\mathsf{crs}, \mathsf{stmt}, w)$ where $\mathsf{stmt} := (\eta, c[\mathcal{P}], k_0, \mathcal{P}, t')$, $w := (k[\mathcal{P}], r[\mathcal{P}])$, and sends η, π to $\mathcal{A}$. In the above, $k[\mathcal{P}]$ is the honest party's key chosen for $\mathcal{P}$ by $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$, $c[\mathcal{P}]$ was the commitment for party $\mathcal{P}$ computed by $\mathcal{F}_{\rm hyb}^{\langle 4\rangle}$ and revealed to $\mathcal{A}$, and $r[\mathcal{P}]$ was the randomness used in this commitment.

Claim 4 (Security of $\Pi_{\rm hyb}^{\langle 4\rangle}$ under adaptive sleepiness and static corruption)**.** Assume that the commitment scheme is hiding both against subexponential adversaries, and the NIZK scheme satisfies computational zero-knowledge against subexponential adversaries. Then, if there is a $\Pi_{\rm hyb}^{\langle 4\rangle}$-compliant $(\mathcal{A}, \mathcal{Z})$ running in time subexponential in L that can cause bad events related to chain growth, quality, or consistency to happen with probability ϵ in $\mathsf{EXEC}^{\Pi_{\rm hyb}^{\langle 4\rangle}}(\mathcal{A}, \mathcal{Z}, \lambda)$, then there exists a subexponential, $\Pi_{\rm hyb}^{\langle 3\rangle}$-compliant $(\mathcal{A}', \mathcal{Z}')$ that can cause the same bad events to happen in $\mathsf{EXEC}^{\Pi_{\rm hyb}^{\langle 3\rangle}}(\mathcal{A}', \mathcal{Z}', \lambda)$ with probability $\epsilon - 2^{-L^\delta}$.

Proof. By straightforward reduction to the hiding property of the commitment scheme and the computational zero-knowledge property of the zero-knowledge proof against subexponential adversaries. □

Hybrid protocol $\Pi_{\mathbf{hyb}}^{\langle 5\rangle}$. $\Pi_{\rm hyb}^{\langle 5\rangle}$ is almost identical to $\Pi_{\rm hyb}^{\langle 4\rangle}$ except with the following changes (we call the new ideal functionality $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}$ in $\Pi_{\rm hyb}^{\langle 5\rangle}$):

- Instead of having $\mathcal{A}$ register $k[\mathcal{P}]$ with $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}$ for a corrupt party $\mathcal{P}$, we now have $\mathcal{A}$ register $(k[\mathcal{P}], r[\mathcal{P}], c[\mathcal{P}])$ with $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}$ (if such a tuple has not been chosen before) such that $c[\mathcal{P}] = \mathsf{com}(k[\mathcal{P}]; r[\mathcal{P}])$.
- Whenever $\mathcal{A}$ or $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}$ internall calls $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}.\mathtt{leader}(\mathcal{P}, t)$ on for a corrupt party $\mathcal{P}$, $\mathcal{F}_{\rm hyb}^{\langle 5\rangle}$ performs the following:
  1. If $\mathcal{A}$ has earlier supplied *i)* a tuple $(k[\mathcal{P}], r[\mathcal{P}], c[\mathcal{P}])$ for corrupt party $\mathcal{P}$, *ii)* a value $\eta < D_p$, and *iii)* a valid NIZK proof π for the statement $\mathsf{stmt} := (\eta, c[\mathcal{P}], k_0, \mathcal{P}, t)$, then if $w = (k[\mathcal{P}], r[\mathcal{P}])$ is not a valid witness for stmt, abort outputting $\mathsf{soundness\text{-}failure}$; else return 1.
  2. In all other cases, return 0.

Claim 5 (Security of $\Pi_{\text{hyb}}^{\langle 5\rangle}$ under adaptive sleepiness and static corruption)**.** If there is a $\Pi_{\text{hyb}}^{\langle 5\rangle}$-compliant $(\mathcal{A}, \mathcal{Z})$ running in time subexponential in L that can cause bad events related to chain growth, quality, or consistency to happen with probability ϵ in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{\langle 5\rangle}}(\mathcal{A}, \mathcal{Z}, \lambda)$, then there exists a subexponential, $\Pi_{\text{hyb}}^{\langle 4\rangle}$-compliant $(\mathcal{A}', \mathcal{Z}')$ that can cause the same bad events to happen in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{\langle 4\rangle}}(\mathcal{A}', \mathcal{Z}', \lambda)$ with probability ϵ.

Proof. The proof is trivial. □

Hybrid protocol Π_{hyb}^{*}. Π_{hyb}^{*} is almost identical to $\Pi_{\text{hyb}}^{\langle 5\rangle}$ except that the new ideal functionality $\mathcal{F}_{\text{tree}}^{*}$ does not check for $\mathsf{soundness\text{-}failure}$, and the adversary $\mathcal{A}$ only registers $c[\mathcal{P}]$ for corrupt party $\mathcal{P}$ without having to explain the commitment with $k[\mathcal{P}], r[\mathcal{P}]$.

Claim 6 (Security of Π_{hyb}^{*} under adaptive sleepiness and static corruption)**.** Assume that the commitment scheme is perfectly binding and that the NIZK scheme satisfies computational soundness against subexponential adversaries. Then, if there is a Π_{hyb}^{*}-compliant $(\mathcal{A}, \mathcal{Z})$ running in time subexponential in L that can cause bad events related to chain growth, quality, or consistency to happen with probability ϵ in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{*}}(\mathcal{A}, \mathcal{Z}, \lambda)$, then there exists a subexponential, $\Pi_{\text{hyb}}^{\langle 5\rangle}$-compliant $(\mathcal{A}', \mathcal{Z}')$ that can cause the same bad events to happen in $\mathsf{EXEC}^{\Pi_{\text{hyb}}^{\langle 5\rangle}}(\mathcal{A}', \mathcal{Z}', \lambda)$ with probability $\epsilon - 2^{-L^{\delta}}$.

Proof. First, we show that $\Pi_{\text{hyb}}^{\langle 5\rangle}$ does not abort with $\mathsf{soundness\text{-}failure}$ except with $2^{-L^{\delta}}$ probability. Since the commitment scheme is perfectly binding, if there is a valid witness, it must be $(k[\mathcal{P}], r[\mathcal{P}])$. Therefore, if $(k[\mathcal{P}], r[\mathcal{P}])$ is not a valid witness then the statement must be false; but if $\mathcal{A}$ can forge a valid NIZK proof for such a statement with more than $2^{-L^{\delta}}$ probability, we can easily build a reduction that breaks the computational soundness of the NIZK.

Due to the above, we may consider a version of $\mathcal{F}_{\text{hyb}}^{*}$ does not check for $\mathsf{soundness\text{-}failure}$ but $\mathcal{A}$ still submits a valid explanation $k[\mathcal{P}], r[\mathcal{P}]$ along with $c[\mathcal{P}]$. Since $\mathsf{soundness\text{-}failure}$ happens only with $2^{-L^{\delta}}$ failure probability, for any $(\mathcal{A}, \mathcal{Z})$, any bad event (related to chain quality, chain growth, or consistency) that happens in $\Pi_{\text{hyb}}^{\langle 5\rangle}$ with probability ϵ can happen with probability at most $\epsilon + 2^{-L^{\delta}}$ here. Now, since $k[\mathcal{P}], r[\mathcal{P}]$ is never used by $\mathcal{F}_{\text{hyb}}^{*}$, we do not require $\mathcal{A}$ to submit $k[\mathcal{P}], r[\mathcal{P}]$, and this should not affect the probability of any bad event (related to chain growth, quality, or consistency). □

8.3 The Real World Emulates the Hybrid World

Simulator construction. We construct the following simulator $\mathcal{S}$.

- In the beginning, the simulator $\mathcal{S}$ learns from $\mathcal{F}_{\text{hyb}}^{*}$ the value of k_0, $\mathsf{NIZK.crs}$, as well as commitments of $k[\mathcal{P}]$ for every honest node $\mathcal{P}$. The simulator sets $\mathsf{params} := (k_0, \mathsf{NIZK.crs})$ as the common reference string, and supplies it to $\mathcal{A}$ any time upon query.

- For each honest node $\mathcal{P}$, the simulator $\mathcal{S}$ chooses a signing key pair $(\mathsf{pk}[\mathcal{P}], \mathsf{sk}[\mathcal{P}])$ honestly.

  The simulator simulates $\mathcal{F}_{\text{CA}}$. At the start of the execution, for each honest party $\mathcal{P}$: the simulator and registers $(\mathsf{pk}[\mathcal{P}], c[\mathcal{P}])$ on behalf of $\mathcal{P}$ with the internally simulated $\mathcal{F}_{\text{CA}}$, where $\mathsf{pk}[\mathcal{P}]$ was chosen earlier by $\mathcal{S}$ and $c[\mathcal{P}]$ denotes the commitment $\mathcal{S}$ received earlier from $\mathcal{F}_{\text{tree}}^{*}$ for honest party $\mathcal{P}$.

- If $\mathcal{A}$ tries to register the pair $(\mathsf{pk}[\mathcal{P}], c[\mathcal{P}])$ with $\mathcal{F}_{\text{CA}}$ on behalf of corrupt party $\mathcal{P}$, $\mathcal{S}$ simply forwards the request to the simulated $\mathcal{F}_{\text{CA}}$ and registers $c[\mathcal{P}]$ with $\mathcal{F}^*_{\text{hyb}}$.
- $\mathcal{S}$ keeps track of the "real-world" chain for every honest node $\mathcal{P}$. Whenever it sends *chain* to $\mathcal{A}$ on behalf of an honest $\mathcal{P}$, it updates this state for node $\mathcal{P}$. Whenever $\mathcal{A}$ sends *chain* to honest node $\mathcal{P}$, it may also update $\mathcal{P}$'s state in ways to be described later.
- Whenever $\mathcal{A}$ sends *chain* on behalf of corrupt party $\mathcal{P}'$ to to honest node $\mathcal{P}$, $\mathcal{S}$ checks the (real-world) validity of *chain* w.r.t. params and the current state of $\mathcal{F}_{\text{CA}}$. If the check fails, the simulator simply ignores this message. Otherwise, do the following.

  (a) If *chain* is longer than the current real-world chain for the honest recipient $\mathcal{P}$, $\mathcal{S}$ saves *chain* as the new real-world chain for $\mathcal{P}$.

  (b) Let $\mathsf{chain} := \mathsf{extract}(chain)$, and let $\mathsf{chain}[:\ell] \prec \mathsf{chain}$ be the longest prefix such that $\mathcal{F}^*_{\text{hyb}}.\mathtt{verify}(\mathsf{chain}[:\ell]) = 1$. The simulator checks to see if there exists a block in $chain[\ell{+}1:]$ signed by an honest $\mathcal{P}$. If so, abort outputting $\mathsf{sig\text{-}failure}$. Else, for each $j \in [\ell+1, |\mathsf{chain}|]$,

    (i) Let $\mathcal{P}^* := chain[j].\mathcal{P}$, let $t^* := chain[j].\mathsf{time}$, $\pi := chain[j].\pi$, and $\eta := chain[j].\eta$.

    (ii) Note that since the *chain* verifies it must be the case that $\mathcal{A}$ has registered $(\mathsf{pk}[\mathcal{P}^*], c[\mathcal{P}^*])$ with $\mathcal{S}$. Now, $\mathcal{S}$ supplies π to $\mathcal{F}^*_{\text{hyb}}$ for the statement $\mathsf{stmt} := (\eta, c[\mathcal{P}^*], k_0, \mathcal{P}^*, t^*)$

    (iii) $\mathcal{S}$ then calls $\mathcal{F}^*_{\text{hyb}}.\mathtt{extend}(\mathsf{chain}[:j-1], \mathsf{chain}[j], t^*)$ on behalf of corrupt party $\mathcal{P}^*$.

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

  let $(\eta, \pi) := \mathcal{F}^*_{\text{hyb}}.\mathtt{nizk}(\mathcal{P}, t)$
  let $\sigma := \Sigma.\mathtt{sign}_{\mathsf{sk}}(chain[-1].h, \mathsf{B}, t, \eta, \pi)$, $h' := \mathsf{d}(chain[-1].h, \mathsf{B}, t, \mathcal{P}, \eta, \pi, \sigma)$
  let $B := (chain[-1].h, \mathsf{B}, t, \mathcal{P}, \eta, \pi, \sigma, h')$, let $chain' := chain||B$ and gossip *chain*

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

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

Indistinguishability. We now prove that the simulated execution and the real-world executions are computationally indistinguishable.

Fact 10. Assume that the collision resistant hash function and the signature scheme are secure. The simulated execution never aborts with $\mathsf{duplicate\text{-}block\text{-}failure}$ or $\mathsf{sig\text{-}failure}$ except with negligible probability.

Proof. Same as the proofs of Facts 7 and 8. If the above bad events happen with non-negligible probability, we can construct a polynomial-time reduction that breaks the collision resistance of the hash family or the signature scheme. □

Fact 11. Conditioned on no $\mathsf{duplicate\text{-}block\text{-}failure}$ and no $\mathsf{sig\text{-}failure}$ the simulated execution is identically distributed as the real execution from the view of $\mathcal{Z}$.

Proof. Straightforward to observe. In particular, we point out that whenever $\mathcal{S}$ receives a valid *chain* from $\mathcal{A}$, either $\mathsf{extract}(chain)$ is already in $\mathcal{F}^*_{\text{hyb}}$ or the simulator $\mathcal{S}$ must succeed in adding $\mathsf{extract}(chain)$ to $\mathcal{F}^*_{\text{hyb}}$. □

8.4 Proofs for Adaptive Sleepiness and Adaptive Corruption

So far, we have proved security under static corruption but adaptive sleepiness. Now, we would like to prove security under adaptive corruption — here rely on standard complexity leveraging techniques.

Our earlier proof shows the following: if there is a real-world p.p.t. $(\mathcal{A}, \mathcal{Z})$ that statically corrupts nodes and can break the security properties of Π^*_{sleepy}, then we can construct a p.p.t. reduction that interacts with $(\mathcal{A}, \mathcal{Z})$ and breaks either the security of either the PRF, the hash function, the NIZK, or the digital signature scheme.

Now, suppose we have an adaptive adversary $(\mathcal{A}', \mathcal{Z}')$ that can break the security properties of Π^*_{sleepy} with probability $\epsilon = \frac{1}{\mathsf{poly}(\lambda, N)}$. We can construct a static adversary $(\mathcal{A}, \mathcal{Z})$ that makes random guesses as to what $(\mathcal{A}', \mathcal{Z}')$. If the guess turns out to be wrong later, $(\mathcal{A}, \mathcal{Z})$ simply aborts. Such a $(\mathcal{A}, \mathcal{Z})$ pair can break the security properties of Π^*_{sleepy} with probability $\frac{\epsilon}{2^N}$ since $(\mathcal{A}, \mathcal{Z})$ can guess correctly with probability 2^{-N}. It holds that $(\mathcal{A}, \mathcal{Z})$ must be able to break either the PRF, the hash function, the NIZK, or the digital signature scheme with probability $\frac{\epsilon}{2^N}$. Therefore, if we choose the security parameter of these cryptographic schemes to be $L := (2N + \log^2 \lambda)^{\frac{1}{\delta}}$, we have that $\mathsf{poly}(\lambda, N) \cdot 2^N \ll 2^{((2N+\log^2 \lambda)^{\frac{1}{\delta}})^{\delta}} + \exp(-\Omega(LN))$, then this should not be possible by our subexponential hardness assumptions.

References

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

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

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

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

[5] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. `https://eprint.iacr.org/2016/919.pdf`.

[6] Alysson Neves Bessani, João Sousa, and Eduardo Adílio Pelinson Alchieri. State machine replication for the masses with BFT-SMART. In *DSN*, 2014.

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

[8] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

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

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

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

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

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

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

[15] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.

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

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

[18] 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.

[19] 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.

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

[21] Rachid Guerraoui, Florian Huc, and Anne-Marie Kermarrec. Highly dynamic distributed computing with byzantine failures. In *PODC*, pages 176–183, 2013.

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

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

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

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

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

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

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

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

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

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

[32] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. In *FOCS*, 1999.

[33] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *ACM CCS*, 2016.

[34] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *SIGCOMM*, pages 123–133, Stanford, CA, 1988.

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

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

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

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

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

A Chernoff Bound

For completeness, we quote the version of Chernoff Bound adopted in this paper.

Theorem 10 (Chernoff bound). *Let* $\mathbf{X} := \sum_{i=1}^{n} \mathbf{X}_i$*, where each* $\mathbf{X}_i = 1$ *with probability* p_i*, and* $\mathbf{X}_i = 0$ *with probability* $1 - p_i$*; and further, all* $\mathbf{X}_i$*'s are independent. Let* $\mu := \mathbf{E}[\mathbf{X}] = \sum_{i=1}^{n} p_i$*. Then, we have that*

$$\Pr[\mathbf{X} > (1+\delta)\mu] \leq e^{\frac{\delta^2}{2+\delta}\mu} \textit{ for all } \delta > 0$$