The Sleepy Model of Consensus

Rafael Pass(✉) and Elaine Shi

Cornell Tech, Cornell University, New York, USA
rafael@cs.cornell.edu

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* (awake) 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.

Keywords: Blockchains · Distributed consensus · Protocol · Adaptive security

T. Takagi and T. Peyrin (Eds.): ASIACRYPT 2017, Part II, LNCS 10625, pp. 380–409, 2017.
https://doi.org/10.1007/978-3-319-70697-9_14

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 "Facebuck 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 *thousands* of players—makes this latter assumption unrealistic. For instance, in Bitcoin, only a small fraction of users having bitcoins are actually participating as miners. More generally, although it has not been explicitly articulated, an implicit desiderata for "permissioned blockchains" seems to be a notion of "availability-friendliness" where the blockchain should be able to confirm new transaction even if only a small fraction of participants are actually actively running the protocol.

1.1 The Sleepy Model of Consensus

Towards addressing this issue, and formalizing the notion of "availability-friendliness", we here initiate a study of distributed protocols in a "sleepy" model of computation. We here focus on the "standard" permission setting with a fixed number of player and the existence of a PKI. In the sleepy 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 [19] 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. We believe that such a sleepy model of computation is the "right way" to formalize the availability-friendliness desiderata informally articulated for permissioned blockchains, and elucidates the resistence in the blockchain community to adopt classic consensus protocols for permissioned blockchains.

We note that it is easy to show that consensus in the sleepy model is impossible, even in a PKI 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 "fake" execution by the malicious players, and thus must choose the "fake" one with probability at least $\frac{1}{2}$. We formalize this in the online full version [25]. (Note that this simple result already demonstrates a sharp contrast with the classic (non-sleepy) model, where Dolev-Strong's [7] protocol can be used to realize a linearly ordered log of transactions assuming just the existence of a single honest player [25,27].)

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., [7,10,12]) crucially rely on messages being delivered in the next round (or within a known, bounded delay Δ) to

[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.

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 [7] 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., [3,5,8,18,21,28]) a-priori seem to handle the above-mentioned issue with the synchronous protocol: we can simply view a 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 ever 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 [5] (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 a Common *Random* String (CRS)[4] and is proven secure in a simple version of the *timing model* of Dwork-Naor-Sahai [22] 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

[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".

a static (fixed) schedule of which nodes are awake at what time step—we refer to this as a "static sleep schedule".

Theorem 1 (Informal). *Assume the existence of families of 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 sleep 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 against 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); additionally, we here require using a large security parameter (greater than the bound on the total number of players), leading to a less efficient protocol.

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.*

We finally point out that if the CRS is selected after the public keys have been registered, and if we only need security w.r.t. static corruption but adaptive sleep schedules, we do not need to use a subexponential security or a large security parameter. (In fact, we can also get security w.r.t. adaptive corruptions with *erasure* if we additionally assuming the existence of a forward-secure signature, and consider a protocol in the random oracle model.)

Perhaps surprisingly, our protocol significantly departs from the standard approaches to distributed consensus, and we instead rely on key ideas behind Nakamoto's beautiful blockchain protocol [23], while dispensing the need for "proofs-of-work" [9]. 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. [24], 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 [1,2,15], 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 [23]—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 in 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 advantages than honest nodes. Specifically, as mentioned earlier, the honest 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 [11,24] 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 [11,24]. Existing works [11,24] 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 an appropriate prefix of the chain)—which, as shown in earlier works [24,26] 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 [24]. The bad news is that the consistency proofs of prior works [11,24] 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 and made public 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 random oracle/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 the 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 [19]—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 the random oracle: As before, the CRS specifies the seed k_0 of a PRF, and 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. $\mathcal{P}$ 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 $\mathsf{PRF}_{k_0}(\mathcal{P}, t) \oplus \mathsf{PRF}_{k[\mathcal{P}]}(t)$

[5] In essence, what we need is a VRF [20], just like Micali [19], but since we anyway have a CRS, we can rely on weaker primitives.

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 adversary's 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.

Better Efficiency and Assumption in Stronger Models. We note that the loss in efficiency due to the above-mentioned union bounds is non-trivial: the security parameter must now be greater than N; if we only require security with respect to 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 union bound and thus without the loss in efficiency.

In fact, we can even deal with adaptive corruption of players in a model *with erasures* if we let players sign using a forward secure signature scheme (and at each step erase the old key). For technical reasons, we here, however, can only provide a proof of security in the random oracle model.[6] We defer the details of these approaches to the online full version [25].

1.4 Applications in Permissioned and Permissionless Settings

As mentioned, the variants of our protocols that deal with static corruption (and adaptive corruption with erasures) need not employ a large security parameter and can be implemented and adopted in real-world systems. We believe that our sleepy consensus protocol would be desirable in the following application scenarios.

Permissioned Setting: Consortium Blockchains. At the present, there is a major push where blockchain companies are helping banks across the world build "consortium blockchains". In a consortium blockchain, 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 [5], Byzantine Paxos [16], 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

[6] Technically, the issue is that we need to rely on a PRF that is secure with respect to "selective-opening" and we can only construct this in the random oracle model.

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 [17], 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 Daian et al. [6] 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 [6] extended the our sleepy consensus protocol by introducing a mechanism that relies on the distribution of stake in the system to periodically rotate the consensus committee.

Comparison with Independent Work. Although proof-of-stake is not a focus of our paper, we compare with a few independent works on proof-of-stake [14,19] due to the superficial resemblance of some elements of their protocol in comparison with ours. Specifically, the elegant work by Micali proposes to adapt classical style consensus protocols to realize a proof-of-stake protocol [19]; the concurrent and independent work by Kiayias et al. [14] 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. We also point out that if we were to replace Kiayias et al.'s coin toss protocol with an ideal random beacon, their proof would still fail in the sleepy model. Other proof-of-stake protocols [1,2,15] 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. As far as we are aware, we are the first to formally show how to remove proof-of-work from Nakamoto's protocol in a provably secure way, while maintaining its desirable "availiability-friendliness" property.

1.5 Paper Organization

The remainder of the paper is organized as follows. In Sect. 2, we formally define the sleepy execution model. In Sect. 3, we define the abstraction realized by a blockchain protocol and a state machine replication protocol. Our goal in this paper is to realize state machine replication where a set of nodes agree on an growing log of transactions—but we achieve this through realizing a blockchain protocol—as previous works show [26], blockchains give a direct method of instantiating state machine replication.

Then, in Sect. 4, we describe the sleepy consensus protocol that implements the blockchain abstraction in the sleepy execution model, assuming a static sleep

schedule and static corruptions. We present an overview of our proof for this statically secure protocol in Sect. 5, and defer the full proofs to the online full version [25].

Finally, in Sect. 6, we describe intuitively how to achieve adaptive security by leveraging additional cryptographic building blocks such as non-interactive zero-knowledge proofs and by relying on subexponential security and large security parameters (and the above-mentioned union bounds).

We defer the full presentation of the adaptively secure sleepy consensus protocol, as well as the more efficient variants mentioned above, to the online full version [25]. The full version also contain new lower bounds for the sleepy model.

2 Definitions

2.1 Protocol Execution Model

We assume a standard Interactive Turing Machine (ITM) model often adopted in the cryptography literature. A protocol specifies a set of instructions for the protocol participants to interact with each other. The protocol's execution is directed by an environment denoted $\mathcal{Z}$ who is in charge of activating a number of parties (also referred to as nodes) that will interact with each other.

Notations for Randomized Protocol Execution. We use the notation $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ to denote a randomized execution of the protocol Π with security parameter λ and w.r.t. to an $(\mathcal{A}, \mathcal{Z})$ pair. Specifically, view is a random variable containing an ordered sequence of all inputs, outputs, and messages sent and received by all Turing Machines during the protocol's execution. We use the notation $|\mathsf{view}|$ to denote the number of time steps in the execution trace view.

Weak Clock Synchrony Assumptions. We assume the protocol execution proceeds in atomic time steps called rounds. Henceforth in the paper, we will use the terms "round" and "time" interchangably. We assume that all honest nodes are aware of the present round number (i.e., time). As is well-known, such a model can also capture the case of *weak clock synchrony* by treating the clock offset as part of the network delay. Specifically, in a setting where the maximum network delay and the maximum clock offset are both Δ (and if the clock offset is larger than the maximum network delay, we take the maximum of the two to be Δ), we can equivalently treat it as a setting with perfectly synchronized clocks but 3Δ maximum network delay. Our clock synchrony assumptions are similar to those described by Dwork et al. [8].

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 [4]. Specifically, we shall assume that the PKI is an ideal functionality $\mathcal{F}_{\mathrm{CA}}$ (available only to the present protocol instance) that does the following:

- On receive `register`(`upk`) from $\mathcal{P}$: remember $(\texttt{upk}, \mathcal{P})$ and ignore any future message from $\mathcal{P}$.
- On receive `lookup`($\mathcal{P}$): return the stored `upk` corresponding to $\mathcal{P}$ or $\perp$ if none is found.

In this paper, we will consider a Bare PKI model, nodes are allowed register their public keys with $\mathcal{F}_{\text{CA}}$ any time during the execution—although typically, the honest protocol may specify that honest nodes register their public keys upfront at the beginning of the protocol execution (nonetheless, corrupt nodes may still register late).

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, 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.

Communication Model. 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.

Henceforth in this paper, we assume that our protocol is aware of the maximum network delay Δ—in fact, we prove that no protocol without knowledge of Δ can reach consensus in the sleepy model (see our online version [25]).

2.2 Compliant Executions

Parameters of an Execution. Globally, we will use N to denote (an upper bound on) the total number of nodes, and N_{crupt} to denote (an upper bound on) the number of corrupt nodes, and Δ to denote the maximum delay of messages between alert nodes. More formally, we can define a $(N, N_{\text{crupt}}, \Delta)$-respecting $(\mathcal{A}, \mathcal{Z})$ as follows.

Definition 1 ($(N, N_{\text{crupt}}, \Delta)$-respecting $(\mathcal{A}, \mathcal{Z})$). *Henceforth, we say that $(\mathcal{A}, \mathcal{Z})$ is $(N, N_{\text{crupt}}, \Delta)$-respecting w.r.t. protocol Π, iff the following holds: for any* view *in the support of* $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$,

- *$(\mathcal{A}, \mathcal{Z})$ spawns a total of N nodes in* view *among which N_{crupt} are corrupt and the remaining are honest.*
- *If an alert node i multicasts a message at time t in* view*, then any node j alert at time $t' \geq t + \Delta$ (including ones that wake up after t) will have received the message.*

Henceforth when the context is clear, we often say that $(\mathcal{A}, \mathcal{Z})$ is $(N, N_{\text{crupt}}, \Delta)$-respecting omitting stating explicitly the protocol Π of interest.

Protocol-Specific Compliance Rules. A protocol Π may ensure certain security guarantees only in executions that respect certain compliance rules. Compliance rules can be regarded as constraints imposed on the $(\mathcal{A}, \mathcal{Z})$ pair. Henceforth, we assume that besides specifying the instructions of honest parties, a protocol Π will additionally specify a set of compliance rules. We will use the notation a

$$\Pi\text{-compliant } (\mathcal{A}, \mathcal{Z}) \text{ pair}$$

to denote an $(\mathcal{A}, \mathcal{Z})$ pair that respects the compliance rules of protocol Π—we also say that $(\mathcal{A}, \mathcal{Z})$ is compliant w.r.t. to the protocol Π.

2.3 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 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 $\lambda \in \mathbb{N}$. Similarly, if we say that a variable var is positive or non-negative, it means positive or non-negative for every input λ. 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.*

Unless otherwise noted, we assume that all variables are non-negative (functions of λ). Further, unless otherwise noted, all variables are *polynomially bounded* (or *inverse polynomially bounded* if smaller than 1) functions of λ.

3 Problem Definitions

In this section, we first formally define a state machine replication protocol; roughly speaking, in state machine replication, nodes agree on a *linearly ordered log* over time, in a way that satisfies consistency and liveness. We next define a blockchain abstraction following Pass et al. [24] (which in turn relies on properties from Garay et al. [11]). For both of these definitions, we extend the definitions to the sleepy model of computation. Finally, as shown by Pass and Shi [26], any protocol satisfying the blockchain abstraction can be turned (by simply truncating the last few blocks) into a state machine replication protocol, and the same results applies also in the sleepy model. As a consequence, it will later suffice to simply prove that our protocol securely implements the blockchain abstraction.

3.1 State Machine Replication

We turn to formalizing the notion of state machine replication; we use the definition from [26] and extend it to the sleepy model by simply replacing honest nodes for alert nodes.

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 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 and has transaction conformation time T_{confirm} if for every Π-compliant $(\mathcal{A}, \mathcal{Z})$ that is $(N, N_{\text{crupt}}, \Delta)$-respecting, 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 $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$. Here the relation $\preceq$ means "is a prefix of". By convention we assume that $\emptyset \preceq x$ and $x \preceq x$ 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 $\mathsf{LOG} \preceq \mathsf{LOG}'$.
– *Liveness*: An execution trace view satisfies T_{confirm}-liveness if the following holds: suppose that in view, the environment $\mathcal{Z}$ inputs some set that contains tx to an alert node at time $t \leq |\mathsf{view}| - T_{\text{confirm}}$, or that tx appears in some honest node's LOG at time t. 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 $\mathsf{tx} \in \mathsf{LOG}$.
  Intuitively, liveness says that transactions input to an alert node get included in their LOGs within T_{confirm} time; further, if some transaction shows up in an honest node's LOG, it will show up in all other alert nodes' logs quickly.

3.2 Blockchain Formal Abstraction

In this section, we define the formal abstraction and security properties of a blockchain. Our definitions is identical to the abstraction of Pass et al. [24] (which in turn is based on earlier definitions from Garay et al. [11], and Kiayias and Panagiotakos [13]), and again simply replaces honest nodes with alert nodes.

Inputs and Outputs. We assume that in every time step, the environment $\mathcal{Z}$ provides a possibly empty input to every alert node. Further, in every time step, an alert node sends an output to the environment $\mathcal{Z}$. Given a specific execution trace view 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.

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 2 (Chain growth). *A blockchain protocol Π satisfies (T_0, g_0, g_1)-chain growth, if for all Π-compliant pair $(\mathcal{A}, \mathcal{Z})$, there exists a 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:*

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

Additionally, we say that a blockchain protocol Π satisfies (T_0, g_0, g_1)-chain growth w.r.t. failure probability $\mathsf{negl}(\cdot)$ if the above definition is satisfied when the negligible function is fixed to $\mathsf{negl}(\cdot)$ for any Π-compliant $(\mathcal{A}, \mathcal{Z})$.

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 3 (Chain quality). *A blockchain protocol Π has* (T_0, μ)-chain quality*, if for all Π-compliant 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:*

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

Additionally, we say that a blockchain protocol Π satisfies (T_0, μ)-chain quality w.r.t. failure probability $\mathsf{negl}(\cdot)$ if the above definition is satisfied when the negligible function is fixed to $\mathsf{negl}(\cdot)$ for any Π-compliant $(\mathcal{A}, \mathcal{Z})$.

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 4 (Consistency). *A blockchain protocol* Π *satisfies* T_0-consistency*, if for all* Π*-compliant 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:*

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

Additionally, we say that a blockchain protocol Π satisfies T_0-consistency w.r.t. failure probability $\mathsf{negl}(\cdot)$ if the above definition is satisfied when the negligible function is fixed to $\mathsf{negl}(\cdot)$ for any Π-compliant $(\mathcal{A}, \mathcal{Z})$.

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

Following [26], 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.

Lemma 1 (Blockchains imply state machine replication [26]). *If there exists a blockchain protocol that satisfies* (T_G, g_0, g_1)*-chain growth,* (T_Q, μ)*-chain quality, and* T_C*-consistency, then there exists a secure state machine replication protocol with 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 [26] for a different execution model, but the same proof effectively holds in our sleepy execution model. Specifically, let $\Pi_{\text{blockchain}}$ be such a blockchain protocol. We can consider the following state machine replication protocol denoted Π': whenever an alert node is about to output chain to the environment $\mathcal{Z}$ in $\Pi_{\text{blockchain}}$, it instead outputs $\mathsf{chain}[:-T_C]$. Further, suppose that Π''s compliance rules are the same as $\Pi_{\text{blockchain}}$'s. Using the same argument as the hybrid consensus paper [26], it is not hard to see that the resulting protocol is a secure state machine replication protocol with confirmation time $O(\frac{T_G+T_Q+T_C}{g_0} + \Delta)$.

As a consequence, henceforth, we will focus on realizing a blockchain protocol as this directly yields a state machine replication protocol.

4 Sleepy Consensus Under 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 first describe our scheme pretending that there is a random oracle H; and then describe how to remove the random oracle assuming a common reference string.

4.1 Valid Blocks and Blockchains

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[7]

1. $\Sigma.\mathsf{ver}_{\mathsf{pk}}((h_{-1}, \mathsf{txs}, \mathsf{time}); \sigma) = 1$ where $\mathsf{pk} := \mathcal{F}_{\text{CA}}.\mathtt{lookup}(\mathcal{P})$; an
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.

[7] Note that since corrupt nodes can register their public keys with $\mathcal{F}_{\text{CA}}$ late into the protocol, validity is actually defined w.r.t. a point of time during the execution.

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 Fig. 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. eligible and time t iff

- $chain[0] = genesis = (\bot, \bot, \mathsf{time} = 0, \bot, \bot, h = \mathbf{0})$, 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. $chain[i].h_{-1} = chain[i-1].h$;
  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 Basic Sleepy Consensus Protocol

We present our basic Sleepy consensus protocol in Fig. 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 $\mathsf{PRF}_{k_0}(\cdot)$.

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).

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

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

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

On receive $chain'$:

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

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 multicast $chain$
- output $\mathsf{extract}(chain)$ to $\mathcal{Z}$ where $\mathsf{extract}$ is the function outputs an ordered list containing the txs extracted from each block in $chain$

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

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

[a] Without loss of generality, we may assume protocol participants are numbered 1 to N.

Fig. 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 pseudorandom function and a common reference string.

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 Sect. 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 Sect. 4.3).

In our online version [?], 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 $\Pi_{\text{sleepy}}(p)$-compliant if the following holds:

- *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.
- *Resilience.* There are parameters $(N, N_{\text{crupt}}, \Delta)$ such that $(\mathcal{A}, \mathcal{Z})$ is $(N, N_{\text{crupt}}, \Delta)$-respecting w.r.t. $\Pi_{\text{sleepy}}(p)$, and moreover, the following conditions are respected:
  - There is a positive constant ϕ, such that for any view in the support of $\mathsf{EXEC}^{\Pi_{\text{sleepy}}(p)}(\mathcal{A}, \mathcal{Z}, \lambda)$, for every $t \leq |\mathsf{view}|$,

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

    where $\mathsf{alert}^t(\mathsf{view})$ denotes the number of nodes that are alert at time t in view.
  - Further, there is some constant $0 < c < 1$ such that $2pN\Delta < 1 - c$.

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

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 the existence of a common reference string (CRS), a bare public-key infrastructure (PKI), and that the signature scheme Σ is secure against any p.p.t. adversary. Then, for any constants $\epsilon, \epsilon_0 > 0$, any $0 < p < 1$, any $T_0 \geq \epsilon_0 \lambda$, $\Pi_{sleepy}(p)$ satisfies (T_0, g_0, g_1)-chain growth, (T_0, μ)-chain quality, and T_0^2 consistency with $\exp(-\Omega(\lambda))$ failure probability for the following set of 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* $\mu = 1 - \frac{1-\epsilon}{1+\phi}$.

where N, Δ, α *and* ϕ *are parameters that can be determined by* $(\mathcal{A}, \mathcal{Z})$ *as well as* p *as mentioned earlier.*

The proof of this theorem will be presented in Sect. 5.

Corollary 1 (Statically secure state machine replication in the sleepy model). *Assume the existence of a common reference string (CRS), a bare public-key infrastructure (PKI), and that the signature scheme* Σ *is secure against any p.p.t. adversary. For any constant* $\epsilon > 0$*, there exists a protocol that achieves state machine replication assuming static corruptions and static sleepiness, and that* $\frac{1}{2} + \epsilon$ *fraction of awake nodes are honest in any time step.*

Proof. Straightforward from Theorem 3 and Lemma 1.

5 Proofs for Static Security

In this section, we present the proofs for the basic sleepy consensus protocol presented in Sect. 4. We assume static corruption and static sleepiness and the random oracle model. Later in our paper, we will describe how to remove the random oracle, and further extend our protocol and proofs to adaptive sleepiness and adaptive corruptions.

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 our online version [25], we will show that the real-world protocol Π_{sleepy} securely emulates the ideal-world protocol.

5.1 Simplified Ideal Protocol Π_{ideal}

Ideal Protocol. Following [24], we first define a simplified protocol Π_{ideal} parametrized with an ideal functionality $\mathcal{F}_{\text{tree}}$—see Figs. 2 and 3. Looking forward, our ideal functionality, $\mathcal{F}_{\text{tree}}$, gives more power to the adversary than the ideal functionality used in [24] (i.e., our ideal functionality is *weaker* than the one in [24]); we provide a more detailed comparison below. As a consequence, our proof of security in this idealized model will be (significantly) more complicated than the one in [24]. We mention that the reason for using this weaker functionality is that we later aim to implement it *without using proofs-of-work* (and in particular, without assuming that a majority of the *computing power* is held by honest player, but rather under the assumption that a majority of the registered public keys are held by honest players).

Roughly speaking, $\mathcal{F}_{\text{tree}}$ 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 (i.e., whether any node is a leader in any time step) through the `leader` query

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

On `init`: tree := genesis, time(genesis) := 0

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

assert chain $\in$ tree, chain$||$B $\notin$ tree, and `leader`$(\mathcal{P}, t)$ outputs 1
append B to chain in tree, record time(chain$||$B) := t, and return "succ"

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

assert chain $\in$ tree, chain$||$B $\notin$ tree, `leader`$(\mathcal{P}^*, t')$ outputs 1, and time(chain) $< t' \leq t$
append B to chain in tree, record time(chain$||$B) = t', and return "succ"

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

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

interface. Finally, alert and corrupt nodes can call $\mathcal{F}_{\text{tree}}$.`extend` to extend known chains with new blocks—$\mathcal{F}_{\text{tree}}$ will then check if the caller is a leader for the time step to decide if the `extend` operation is allowed. $\mathcal{F}_{\text{tree}}$ keeps track of all valid chains, such that alert nodes will call $\mathcal{F}_{\text{tree}}$.`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.

Observe that $\mathcal{F}_{\text{tree}}$ has two entry points named `extend`—one of them is the honest version and the other is the corrupt version. In this ideal protocol, alert nodes always mine in the present, i.e., they always call the honest version of `extend` that uses the current time t. In this case, if the honest node succeeds in mining a new chain denoted chain, $\mathcal{F}_{\text{tree}}$ records the current time t as chain's

Protocol Π_{ideal}

On `init`: chain := genesis

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

Every time step:

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

Fig. 3. Ideal protocol Π_{ideal}

block-time by setting $\mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{time}(\mathsf{chain}) = t$. On the other hand, corrupt nodes are allowed to call a malicious version of `extend` and supply a past time step t'. When receiving an input from the adversarial version of `extend`, $\mathcal{F}_{\text{tree}}$ verifies that the new block's purported time t' respects the strictly increasing rule. If the corrupt node succeeds in mining a new block, then $\mathcal{F}_{\text{tree}}$ records the purported time t' as the chain's block-time.

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 a block-time of t if $\mathcal{F}_{\text{tree}}(\mathsf{view}).\mathsf{time}(\mathsf{chain}) = t$. 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 $\mathcal{F}_{\text{tree}}.\mathtt{extend}(\mathsf{chain}, \mathsf{B})$ or $\mathcal{F}_{\text{tree}}.\mathtt{extend}(\mathsf{chain}, \mathsf{B}, \_)$ 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 $\Pi_{\text{ideal}}(p)$-compliant iff the pair is $\Pi_{\text{sleepy}}(p)$-compliant. Since the protocols' compliance rules are the same, we sometimes just write compliant for short.

Theorem 4 (Security of Π_{ideal}). *For any constant $\epsilon_0, \epsilon > 0$, any $T_0 \geq \epsilon_0 \lambda$, 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 $\mu = 1 - \frac{1-\epsilon}{1+\phi}$.*

where N, Δ, α and ϕ are parameters that can be determined by $(\mathcal{A}, \mathcal{Z})$ as well as p as mentioned earlier.

In the remainder of this section, we will now prove the above Theorem 4.

Intuitions and Differences from the Ideal Protocol in [24]. The key difference between our ideal protocol and Nakamoto's ideal protocol as described by Pass et al. [24] 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 slot, he can reuse the elected slot in 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 [24] can be 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 our online version [25], we present new techniques for analyzing the induced stochastic process.

5.2 Overview of Ideal-World Proofs

As mentioned, chain growth and chain quality follow essentially in the same was as in [24], we thus here focus on giving an overview of the consistency proof.

Review: Consistency Proof for the Nakamoto Blockchain. We first review how Garay et al. [11] and Pass et al. [24] proved consistency for a proof-of-work blockchain, and explain why their proof fails in our setting. To prove consistency, [24] relies on a notion of a *convergence opportunity* (and [11] relies on an analog of this notion in the synchronous setting). Roughly speaking, 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 (or in our setting, alert) node mines a block; and (3) followed by yet another Δ-long period of silence in which no honest node mines a block.

Intuitively, convergence opportunities are a "good pattern" that helps with consistency. In particular, if the unique honest block mined during a convergence opportunity (henceforth denoted B^*) is at length ℓ, then the adversary must mine a block also at length ℓ, otherwise all honest nodes' chains must have a unique block, that is, B^* at length ℓ—and this forces convergence of the entire prefix of the chain up to block B^*.

Finally, to prove consistency, we need to show that in any sufficiently long window, there must be more convergence opportunities than there are adversarially mined blocks. [24] provides a lower bound on the number of convergence opportunities (this is the difficult part of the proof); in contrast, providing an upper bound on the number of adversarially mined blocks is easy and directly follows from a Chernoff bound due to the honest majority of computing power assumption.

Why the Proof Breaks Down in Our Setting. The above-mentioned bound on the number of adversarial blocks, however, relies on the fact that when an adversary successfully extends a chain with a block, he cannot simply transfer this block to some other chain: each mined block requires a separate "computational effort" (i.e., a successful mining), and thus the upperbound simply follows by upperbounding the number of "successful calls" to $\mathcal{F}_{\text{tree}}$.

In contrast, in our setting, if a corrupt node is elected leader in a certain time step t, he can now reuse this credit to extend *multiple* chains, *possibly even at different lengths.*

As such, the above argument (and the proofs of [11,24]) can no longer be applied: we cannot use an upperbound on the number of times the adversary is elected leader (the direct analogy of how many times an adversary mines a block in a proof-of-work blockchain) to get an upperbound on how many *chain lengths* the adversary can attack.

To overcome this, we devise a different proof strategy. The notion of a convergence opportunity will still be important to us (as well as the lower bound on the number of convergence opportunities), but our method for showing convergence will be more sophisticated.

Roadmap of Our Proof. We will define a good event called a (strong) *pivot* point. Roughly speaking, a (strong) pivot is a point of time t, such that if one draws any window of time $[t_0, t_1]$ that contains t, the number of adversarial time slots in that window, if non-zero, must be strictly smaller than the number of convergence opportunities in the same window. We will show the following:

- *A pivot forces convergence:* for any view where certain negligible-probability bad events do not happen: if there is such a pivot point t in view, then the adversary cannot have caused divergence prior to t.
- *Pivots happen frequently:* for all but negligible fraction of the views, pivot points happen frequently in time—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.

We defer the full proofs for chain quality, chain growth, and consistency for the ideal-world protocol Π_{ideal} to our online version [25].

5.3 Real-World Emulates Ideal-World

So far, we have argued security properties for the ideal world protocol. We next need to prove that the same properties, namely, chain growth, chain quality, and consistency translate to the real-world protocol as well. To show this, we rely on a standard simulation argument. For any real-world adversary $\mathcal{A}$, we construct an ideal-world adversary $\mathcal{S}$ (i.e., a simulator), such that no p.p.t. environment $\mathcal{Z}$ can distinguish whether it is in the real- or ideal-world. Recall that the security properties for our protocols are defined w.r.t. honest nodes' output to the environment $\mathcal{Z}$. Thus, such a simulation proof would immediately imply that all relevant security properties we have proven for the ideal world immediately hold in the real world as well. In essence, we prove the following lemma:

Lemma 2 (Real world emulates the ideal world). *For any p.p.t. adversary of the real-world protocol Π_{sleepy}, there exists a p.p.t. simulator $\mathcal{S}$ of the ideal-world protocol Π_{ideal}, such that for any p.p.t. environment $\mathcal{Z}$, for any $\lambda \in \mathbb{N}$, we have the following where $\overset{c}{\equiv}$ denotes computational indistinguishability.*

$$\{\mathsf{view}_{\mathcal{Z}}(\mathsf{EXEC}^{\Pi_{ideal}}(\mathcal{S}, \mathcal{Z}, \lambda))\}_{\lambda} \overset{c}{\equiv} \{\mathsf{view}_{\mathcal{Z}}(\mathsf{EXEC}^{\Pi_{sleepy}}(\mathcal{A}, \mathcal{Z}, \lambda))\}_{\lambda}$$

We defer the proof of this lemma to our online version [25].

5.4 Removing the Random Oracle

It is not difficult to modify our proof when the random oracle query $\mathsf{H}(\mathcal{P}, t)$ is actually instantiated with $\mathsf{PRF}_{k_0}(\mathcal{P}, t)$ where k_0 denotes a sufficiently long common reference string. Specifically, the formal proof introduces a hybrid world in which $\mathcal{F}_{\text{tree}}$'s internal random coins are replaced with outcomes from evaluating the PRF. Although the PRF key k_0 is observable by the adversary, we stress that our ideal-world protocol is secure against a computationally unbounded adversary. Moreover, our ideal-world protocol is secure as long as certain polynomial-time checkable properties, defined over the outcome of the random function or the PRF, are respected. Due to the pseudo-randomness of the PRF, if these polynomial-time checkable properties are respected for all but a negligible fraction of the random functions, then they are respected for all but a negligible fraction of the PRF family too. In our online version [25], we formalize this intuition and present a formal proof.

6 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.

6.1 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 [19]. 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 $\mathcal{F}_{\mathrm{CA}}$. 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.

6.2 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.

6.3 Detailed Protocol and Proofs

We defer the detailed presentation of our adaptively secure protocol and proofs to our online version [25]—however, we state our main theorem for adaptive security below.

Theorem 5 (Adaptively secure state machine replication in the sleepy model). *Assume the existence of a Bare PKI, a CRS; 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 against adaptive corruptions and adaptive sleepiness, as long as $\frac{1}{2} + \epsilon$ fraction of awake nodes are honest in any time step.*

References

1. U. "BCNext": NXT (2014). http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt
2. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Financial Cryptography Bitcoin Workshop (2016)
3. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 524–541. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_31
4. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: FOCS (2001)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI (1999)
6. Daian, P., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. https://eprint.iacr.org/2016/919.pdf

7. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. SIAMCOMP **12**(4), 656–666 (1983)
8. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**, 288–323 (1988)
9. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-48071-4_10
10. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM J. Comput. **26**, 873–933 (1997)
11. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10
12. Katz, J., Koo, C.-Y.: On expected constant-round protocols for byzantine agreement. J. Comput. Syst. Sci. **75**(2), 91–112 (2009)
13. Kiayias, A., Panagiotakos, G.: Speed-security tradeoffs in blockchain protocols. IACR Cryptology ePrint Archive 2015:1019 (2015)
14. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
15. King, S., Nadal, S.: PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake, August 2012
16. Lamport, L.: Fast paxos. Distrib. Comput. **19**(2), 79–103 (2006)
17. Lamport, L., Malkhi, D., Zhou, L.: Vertical paxos and primary-backup replication. In: PODC, pp. 312–313 (2009)
18. Martin, J.-P., Alvisi, L.: Fast byzantine consensus. IEEE Trans. Dependable Secur. Comput. **3**(3), 202–215 (2006)
19. Micali, S.: Algorand: the efficient and democratic ledger (2016). https://arxiv.org/abs/1607.01341
20. Micali, S., Vadhan, S., Rabin, M.: Verifiable random functions. In: FOCS (1999)
21. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of BFT protocols. In: ACM CCS (2016)
22. Mockapetris, P., Dunlap, K.: Development of the domain name system. In: SIGCOMM, Stanford, CA, pp. 123–133 (1988)
23. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
24. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. https://eprint.iacr.org/2016/454
25. Pass, R., Shi, E.: The sleepy model of consensus (2016). https://eprint.iacr.org/2016/918
26. Pass, R., Shi, E.: Hybrid consensus: efficient consensus in the permissionless model. In: DISC (2017)
27. Pass, R., Shi, E.: Thunderella: blockchains with optimistic instant confirmation. Manuscript (2017)
28. Song, Y.J., van Renesse, R.: Bosco: one-step byzantine asynchronous consensus. In: Taubenfeld, G. (ed.) DISC 2008. LNCS, vol. 5218, pp. 438–450. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87779-0_30