Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake

Phil Daian, Rafael Pass, and Elaine Shi(✉)

Cornell/CornellTech, New York, USA
elaine@cs.cornell.edu

Abstract. We present the a provably secure proof-of-stake protocol called Snow White. The primary application of Snow White is to be used as a "green" consensus alternative for a decentralized cryptocurrency system with open enrollement. We break down the task of designing Snow White into the following core challenges:

1. identify a core "permissioned" consensus protocol suitable for proof-of-stake; specifically the core consensus protocol should offer robustness in an Internet-scale, heterogeneous deployment;
2. propose a robust committee re-election mechanism such that as stake switches hands in the cryptocurrency system, the consensus committee can evolve in a timely manner and always reflect the most recent stake distribution; and
3. relying on the formal security of the underlying consensus protocol, prove the full end-to-end protocol to be secure—more specifically, we show that any consensus protocol satisfying the desired robustness properties can be used to construct proofs-of-stake consensus, *as long as money does not switch hands too quickly.*

Snow White was publicly released in September 2016. It provides the first formal, end-to-end proof of a proof-of-stake system in a truly decentralized, open-participation network, where nodes can join at any time (not necessarily at the creation of the system). We also give the first formal treatment of a well-known issue called "costless simulation" in our paper, proving both upper- and lower-bounds that characterize exactly what setup assumptions are needed to defend against costless simulation attacks. We refer the reader to our detailed chronological notes on a detailed comparison of Snow White and other prior and concurrent works, as well as how subsequent works (including Ethereum's proof-of-stake design) have since extended and improved our ideas.

1 Introduction

Although consensus protocols have been investigated by the distributed systems community for 30 years, in the past decade a new breakthrough called Bitcoin established a new, blockchain-based paradigm for reaching consensus in a distributed system. Relying on proof-of-work, Bitcoin's consensus protocol

I. Goldberg and T. Moore (Eds.): FC 2019, LNCS 11598, pp. 23–41, 2019.
https://doi.org/10.1007/978-3-030-32101-7_2

(often called Nakamoto consensus), for the first time, enabled consensus in an open, unauthenticated environment where nodes do not share any pre-established public keys [11,17,18,22]. One commonly known painpoint with this approach is the enormous energy waste. Motivated by the need for a green alternative, the community searched for a paradigm shift, and hoped to obtain a consensus paradigm, commonly called "proof-of-stake", that is based on the idea of "one vote per unit of stake" (as opposed to "one vote per unit of hash-power").

The design of proof-of-stake protocols was first initiated in online forums and blog-posts and subsequently considered by the academic community [2,3,7,14–16,24–26]. Prior to our work, we were not aware of any candidate protocol that offered provable guarantees.

Snow White is the first work to provide end-to-end, formal proofs of security of a full proof-of-stake protocol. Security is proven in a truly decentralized, open-participation environment where honest nodes can join the protocol late in time (and not necessarily at the system's creation). We give the first formal treatment of the well-known "costless simulation" problem (also called posterior corruption in this paper) pertaining to proof-of-stake, proving upper- and lower-bounds that precisely characterize under what assumptions it is possible to defend against costless simulation.

In the remainder of the introduction, we first present an informal technical overview of our results. We then provide detailed chronological notes that position our work in light of other concurrent and subsequent works, and summarize our work's contributions and impact.

1.1 Robustly Reconfigurable Consensus

We ask the question: what is a suitable consensus protocol for a proof-of-stake system? In a proof-of-stake system, at any point of time, we would like the *present* stake-holders to have voting rights that are weighed by their respective stake amount. Thus if we examine any single snapshot in the system, proof-of-stake in fact requires a "permissioned" core consensus protocol, since the set of public-keys owning stake is publicly known. However, proof-of-stake systems aim to support open participation—and this can be enabled through periodic committee reconfiguration. Suppose that the system starts with a well-known set of stake-holders who form the initial consensus committee. As stake switches hands in the system, the consensus committee should be updated in a timely manner to track the present (and not the past) stake distribution. This is important for the security of a proof-of-stake system, since users who no longer hold stake in the system may be incentivized to deviate, e.g., to launch a double-spending attack.

We formulate the task of designing "a consensus protocol suitable for proof-of-stake" as "robustly reconfigurable consensus". A robustly reconfiguration consensus protocol should have the following desirable properties.

Robustness in the Presence of Sporadic Participation. In a large-scale, decentralized environment, users tend to have *sporadic participation*, and it may

be difficult to anticipate how many users will be online at any point of time. Almost all classical-style consensus protocols rely on tallying sufficiently many votes to make progress. If fewer than the anticipated number of users actually show up to vote, the consensus protocol may get stuck.

To address this challenge, Snow White employs the recently proposed "sleepy consensus" [21] paradigm as its core permissioned consensus building block. Sleepy consensus [21] is inspired by the beautiful "longest-chain" idea behind Nakamoto's consensus [17], but the idea is instead applied to a non-proof-of-work, permissioned setting with a public-key infrastructure (PKI). Pass and Shi prove that the resulting consensus protocol is robust in the presence of sporadic participation: concretely, the protocol need not be parametrized with an a-priori fixed number of players that are expected to show up. As long as the majority of *online* players are honest, the protocol guarantees consistency and liveness.

Robust Committee Reconfiguration. Roughly speaking, our system proceeds in epochs. In each epoch, a most recent set of stake-holders are elected as committee and may be randomly chosen to generate blocks. We argue that committee reconfiguration and random block-proposer selection are challenging and subtle due to the following two possible attacks.

1. *Adaptive key selection attacks.* Since proof-of-stake systems admit open participation, anyone can buy up stake in the system and participate. This also means anyone can (possibly maliciously) choose their public-keys through which they participate in the consensus. A possible attack, therefore, is to adaptively choose public-keys, after gathering partial information about the randomness seed used for block-proposer selection, such that corrupt nodes are elected more often as block-proposer than their fair chance.
2. *Randomness-biasing attacks* (commonly known as the "grinding attack"). Another important question is: how do we obtain the randomness needed for block proposer selection? A most straightforward idea is to use the hash of past blocks—but as several works have shown [4], the blocks' hashes can be subject to adversarial influence, and it is unclear what security can be guaranteed when we use such randomness sources with adversarial bias for block proposer selection. For example, the adversary can bias the randomness in a way that allows corrupt nodes to be selected more often.

In the worst case, if through possibly a combination of the attacks, the adversary can control the majority of the block-proposer slots, consistency of the underlying consensus (in our case, sleepy consensus) can be broken.

Snow White proposes a novel "two-lookback" mechanism that addresses the above two challenges simultaneously[1]. We determine each epoch's new consensus committee and randomness seed in a two-phase process, where each phase spans

[1] Subsequent works, including newer versions of Algorand [6] released *after* our publication, Ouroboros Praos [9], and the latest Ethereum's proof-of-stake proposal [1] incorporated elements of this design and suggested improvements, e.g., for concrete security. See Sect. 1.3 for more discussions.

roughly κ blocks of time for some appropriate security parameter[2] κ. This two-phase process is enabled by two look-back parameters as we describe informally below (a formal description is deferred to the technical sections)—henceforth suppose that *chain* is the current longest chain.

1. We look back 2κ blocks, and use the prefix $chain[: -2\kappa]$ (i.e., the prefix of *chain* removing the trailing 2κ blocks) to determine the new consensus committee.
2. We look back κ blocks, and extract the randomness contained in the blocks $chain[-2\kappa : -\kappa]$ (i.e., the part of *chain* from 2κ blocks ago to κ blocks ago) to form a randomness seed—this seed then seeds a random oracle used for block-proposer selection in the current epoch.

Roughly speaking, we defeat the adaptively chosen key attack by determining the consensus committee κ blocks earlier than the randomness seed, such that when corrupt nodes choose their public keys, they cannot predict the randomness seed, which will be generated much later in time and with sufficient entropy contributed by honest nodes as we explain below. We argue that due to chain quality of the underlying sleepy consensus, the blocks $chain[-2\kappa : -\kappa]$ must contain an honest block. Since honest nodes embed a sufficiently long uniform random seed in its block, we can extract sufficiently high-entropy randomness from $chain[-2\kappa : -\kappa]$ which is then used to seed the block-proposer-selection random oracle. Even though the extracted randomness is subject to adversarial bias, as long as it is high-entropy, and importantly, as long as *the same randomness is used to seed the block-proposer selection sufficiently many times*, we can achieve the desired measure concentration properties. More specifically, although indeed, the adversary can bias the random seed to allow corrupt nodes to be selected (as block-proposers) quite surely for a few number of slots; the adversary is not able to consistently gain advantage over a sufficiently large number of slots, i.e., corrupt nodes cannot own noticeably more block-proposer slots than its fair share.

We stress that turning the above intuitive argument into a formal proof requires significant and non-trivial effort which is part our main contributions. In our technical sections, we formally prove security of this approach under a mildly adaptive adversary, i.e., when the adversary is subject to a mild corruption delay and as long as nodes remain honest till shortly after they stop serving on a consensus committee, our robustly reconfigurable consensus protocol is secure. Subsequent works (including newer versions of the Algorand paper that are published *after* the release of Snow White, as well as the subsequent work Ourboros Praos [9]) have suggested approaches for achieving fully adaptive security, but relying on the fact that the majority of nodes will *erase* secret signing keys from memory after signing a block (and by introducing mild addi-

[2] Suppose that except with negligible in κ probability, the underlying sleepy consensus guarantees consistency by chopping off the trailing κ blocks, and guarantees the existence of an honest block in every consecutive window of κ blocks.

tional complexity in the cryptographic schemes employed)—see Sect. 1.3 for a more detailed comparison.

Understanding Posterior Corruption, i.e., "Costless Simulation" Attacks. A oft-cited attack for proof-of-stake systems is the so-called "costless simulation" attack (also referred to as a posterior corruption attack in this paper). The idea is that when stake-holders have sold their stake in the system, nothing prevents them from performing a history-rewrite attack. Specifically, suppose that a set of nodes denoted C control the majority stake in some past committee. These nodes can collude to fork the history from the point in the past when they control majority—and in this alternate history money can transfer in a way such that C continues to hold majority stake (possibly transferred to other pseudonyms of the corrupt nodes) such that the attack can be sustained.

In this paper, we formally prove that under a mild setup assumption—when nodes join the system they can access a set of online nodes the majority of whom are honest—we can provably defend against such a posterior corruption attack. This is achieved by having the newly joining user obtain a somewhat recent checkpoint from the set of nodes it can access upon joining.

We also prove a corresponding lower bound, that absent this setup assumption, defense against such posterior corruption attacks is *impossible*—to the best of our knowledge, ours is the first formal treatment of this well-known costless simulation attack in the context of proof-of-stake.

1.2 From Robustly Reconfigurable Consensus to Proof-of-Stake

Application to Proof-of-Stake and Achieving Incentive Compatibility. We show how to apply such a "robustly reconfigurable consensus" protocol to realize proof-of-stake (the resulting protocol called Snow White), such that nodes obtain voting power roughly proportional to their stake in the cryptocurrency system. As long as *money does not switch hands too fast* (which is enforceable by the cryptocurrency layer), we show that the resulting proof-of-stake protocol can attain security when the adversary controls only a minority of the stake in the system. Further, borrowing ideas from the recent Fruitchain work [19], we suggest *incentive compatible* mechanisms for distributing rewards and transaction fees, such that the resulting protocol achieves a coalition-resistant ϵ-Nash equilibrium, i.e., roughly speaking, as long as the adversary controls a minority of the stake, it cannot obtain more than ϵ fraction more than its fair share of payout, even when it has full control of network transmission and can deviate arbitrarily from the protocol specification.

Preventing Nothing-at-Stake Attacks. Later in Sect. 3, we will also discuss how to leverage guarantees provided by our core consensus protocol, and build additional mechanisms that not only discourage *nothing-at-stake* attackers, but in fact penalize them.

1.3 Chronological Notes, Closely Related, and Subsequent Works

Comparison with Algorand. The first manuscript of Algorand [6] was published prior to our work. Algorand also proposes a proof-of-stake system. Their core consensus protocol is a newly designed classical-style consensus protocol, and therefore they cannot guarantee progress under sporadic participation—instead, Algorand proposes a notion of "lazy participation", where users know when they are needed to vote in the consensus and they only need to be online when they are needed. However, if many users who are anticipated to show up failed to do so, progress will be hampered. Algorand employs a Verifiable Random Function (VRF) to perform random leader/committee election.

Algorand's algorithm has been improved for several iterations. The version of Algorand that existed before the publication of Snow White gave proofs of their core consensus protocol but did not provide end-to-end proofs for the full proof-of-stake system. In particular, the version of Algorand that existed prior to Snow White's publication did not discuss the well-known issue of costless simulation or clearly state the implicit assumptions they make to circumvent the lower bound we prove in this paper.

In their subsequent versions, they adopted the erasure model and rely on honest nodes' capability to safely erase secrets from memory to achieve *adaptive* security (and implicitly, by adopting erasures one could defend against the costless simulation). The newer versions of Algorand (released *after* the Snow White) also started to adopt a similar look-back idea (first described by Snow White) to secure against the adaptive chosen-key attack mentioned earlier. The recent versions also provided more thorough mathematical proofs of this approach.

Comparison with Ouroboros and Ouroboros Praos. Snow White was publicly released in September 2016. A closely related work (independent and concurrent from our effort) known as Ouroboros [13] was release about 10 days prior to Snow White. Ouroboros Praos is an improvement over Ouroboros published in 2017 [9].

The Ouroboros version that was released around the same time as Snow White focused on proving the underlying permissioned consensus building block secure, and there is only a short paragraph containing a proof sketch of their full proof-of-stake system (and this proof sketch has been somewhat expanded to a few paragraphs in later versions). In comparison, our Snow White paper adopts a permissioned consensus building block whose security was formally proven secure in a related paper [21]—the full-length of our technical sections are dedicated to a *thorough* treatment of the security of the end-to-end proof-of-stake system.

A notable difference between Snow White and Ouroboros seems to be that their formal treatment does not seem to capture a truly decentralized environment (necessary for decentralized cryptocurrency applications) where nodes may join the system late and not from the very start—had they done so, they would have encountered the well-known costless simulation issue, which, as we show, is impossible to defend against without extra setup assumptions (and indeed, we introduce a reasonable setup assumption to circumvent this lower bound).

A subsequently improved work, called Ouroboros Praos [9], extends the VRF approach described first by Algorand [6] and Dfinity [12] for random block-proposer election. Similar to the newer versions of Algorand, Ouroboros Praos [9] also started adopting an erasure model to achieve adaptive security (and implicitly, defend against costless simulation[3]).

Neither Ouroboros nor Ouroboros Praos adopts an underlying consensus mechanism that provably provides support for sporadic participation. Finally, the improved version Ouroboros Praos [9] started adopting a look-back mechanism that appears to be inspired by Snow White to for committee rotation and random block-proposer selection.

Comparison with Ethereum's Proof-of-Stake Design. Ethereum began proof-of-stake explorations several years ago. Their design has undergone several versions. At the time of the writing, Ethereum was aiming to do "hybrid proof-of-stake", i.e., use Casper as a finality gadget on top of their existing proof-of-work blockchain.

In the past year 2018, conversations with Ethereum core researchers suggest that Ethereum is considering replacing their proof-of-work blockchain with a proof-of-stake blockchain similar to Snow White. Their committee election and random block proposer selection algorithm seems to be improvement of Snow White. Specifically, they would like to adopt an economically secure coin toss protocol for randomness generation (commonly known as RANDAO). This specific protocol is also subject to adversarial bias much like our randomness seed generation (although biasing attacks may lead to economic loss). Thus they rely on exactly the same observation that was proposed in our paper: although the adversary can bias the randomness sufficiently to control a few block proposer slots, he cannot consistently get an advantage over a large number of slots. Interestingly, Ethereum has several practical optimizations that improve the concrete security parameters of the above analysis [1].

2 Snow White's Core Consensus Protocol

We focus on an intuitive exposition of our scheme in the main body. In the online full version [8], we present formal definitions, a formal description of the protocol, as well as the full proofs. We stress that formalizing the end-to-end security of a proof-of-stake system is a significant effort and this leads to our choice of presentation.

2.1 Background: Sleepy Consensus and Sleepy Execution Model

Sleepy Execution Model and Terminology. We would like to adopt an execution model that captures a decentralized environment where nodes can

[3] Snow White's approach of combining checkpointing and "bootstrapping through social consensus" to defend against costless simulation is simpler and more practical in real-world implementations (than relying on VRFs and erasure [6,9]). Notably, our usage of checkpointing and "bootstrapping through social consensus" already exists in real-world cryptocurrencies.

spawn late in time, and can go to sleep and later wake up. In such a model, the protocol may not have a way to anticipate the number of players at any time.

We thus adopt the sleepy model of execution proposed by Pass and Shi [21]. Nodes are either *sleepy* (i.e., offline) or *awake* (i.e., online and actively participating). For simplicity, we also refer to nodes that are awake and honest as *alert*; and all corrupt nodes are assumed to be awake by convention.

Messages delivered by an alert node is guaranteed to arrive at all other alert nodes within a maximum delay of Δ, where Δ is an input parameter to the protocol. A sleepy node captures any node that is either offline or suffering a slower than Δ network connection. A sleepy node can later wake up, and upon waking at time t, all pending messages sent by alert nodes before $t - \Delta$ will be immediately delivered to the waking node.

We allow the adversary to *dynamically spawn* new nodes, and newly spawned nodes can either be honest or corrupt. Further, as we discuss later, we allow the adversary to declare corruptions and put alert nodes to sleep in a mildly adaptive fashion.

For readability, we defer a detailed presentation of the formal model to our online full version [8].

The Sleepy Protocol as a Starting Point. Classical consensus protocols must count sufficiently many votes to make progress and thus the protocol must know a-priori roughly how many nodes will show up to vote. Since Pass and Shi's Sleepy consensus protocol is the only protocol known to provide consensus under sporadic participation, i.e., the protocol need not have a-priori knowledge of the number of players at any time. We thus consider Sleepy as a starting point for constructing our notion of robustly reconfigurable consensus. We now briefly review the Sleepy consensus protocol as necessary background.

Sleepy is a blockchain-style protocol but without proof-of-work. For practical considerations, below we describe the version of Sleepy instantiated with a random oracle (although Pass and Shi [21] also describe techniques for removing the random oracle). Sleepy relies on a random oracle to elect a leader in every time step. The elected leader is allowed to extend a blockchain with a new block, by signing a tuple that includes its own identity, the transactions to be confirm, the current time, and the previous block's hash. Like in the Nakamoto consensus, nodes always choose the longest chain if they receive multiple different ones. To make this protocol fully work, Sleepy [21] proposes new techniques to timestamp blocks to constrain the possible behaviors of an adversary. Specifically, there are two important blockchain timestamp rules:

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

All aforementioned timestamps can be adjusted to account for possible clock offsets among nodes by applying a generic protocol transformation [21].

2.2 Handling Committee Reconfiguration

As mentioned, our starting point is the Sleepy consensus protocol, which assumes that all consensus nodes know each other's public keys; although it may not be known a-priori how many consensus nodes will show up and participate.

We now discuss how to perform committee reconfiguration such that the consensus committee tracks the latest stake distribution. To support a wide range of applications, our Snow White protocol does not stipulate how applications should select the committee over time. Roughly speaking, we wish to guarantee security as long as the application-specific committee selection algorithm respects the constraint that there is honest majority among all awake nodes. Therefore, we assume that there is some application-specific function $\mathsf{elect\_cmt}(\mathit{chain})$ that examines the state of the blockchain and outputs a new committee over time. In a proof-of-stake context, for example, this function can roughly speaking, output one public key for each currency unit owned by the user. In Sect. 3, we discuss in a proof-of-stake context, how one might possibly translate assumptions on the distribution of stake to the formal requirements expected by the consensus protocol.

Strawman Scheme: Epoch-Based Committee Selection. Snow White provides an epoch-based protocol for committee reconfiguration. To aid understanding, we begin by describing a strawman solution. Each T_{epoch} time, a new epoch starts, and the beginning of each epoch provides a committee reconfiguration opportunity. Let $\mathsf{start}(e)$ and $\mathsf{end}(e)$ denote the beginning and ending times of the e-th committee. Every block in a valid blockchain whose time stamp is between $[\mathsf{start}(e), \mathsf{end}(e))$ is associated with the e-th committee.

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

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

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

Preventing an Adaptive Key Selection Attack. Unfortunately, the above scheme is prone to an adaptive key selection attack where an adversary can break consistency with constant probability. Specifically, as the random oracle H is chosen prior to protocol start, the adversary can make arbitrary queries

to H. Therefore, the adversary can spawn corrupt nodes and seed them with public keys that causes them to be elected leader at desirable points of time. For example, since the adversary can query H, it is able to infer exactly in which time steps honest nodes are elected leader. Now, the adversary can pick corrupt nodes' public keys, such that every time an honest node is leader, a corrupt node is leader too—and he can sustain this attack till he runs out of corrupt nodes. Since the adversary may control up to $\Theta(n)$ nodes, he can thus break consistency for $\Omega(n)$ number of blocks.

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

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

For security, we require that

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

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

Indeed, if each nonce is used only to select the leader in a small number of time steps (say, $O(1)$ time steps), such adversarial bias would indeed have been

detrimental—in particular, by enumerating polynomially many possibilities, the adversary can cause itself to be elected with probability almost 1 (assuming that the adversary controls the last block of the prefix).

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

Reasoning About Security Under Adversarially Biased Hashes. Formalizing this above intuition is somewhat more involved. Specifically, our proof needs to reason about the probability of bad events (related to chain growth, chain quality, and consistency) over *medium-sized* windows such that the bad events depend only on $O(1)$ number of hashes (determined by the nonces used to seed them). This way, we can apply a union bound that results in polynomial security loss. If the window size is too small, it would not be enough to make the failure probability negligible; on the other hand, if the window were too big, the blowup of the union bound would be exponential. Finally, we argue if no bad events occur for every medium-sized window, then no bad events happen for every window (as long as the window is not too small). We defer the detailed discussions and formal proofs to our online full version [8].

2.3 Handling Mildly Adaptive and Posterior Corruptions

We now consider how to defend against an adversary that can adaptively corrupt nodes after they are spawned. In this paper, we will aim to achieve security against a *mildly adaptive adversary*. Specifically, a mildly adaptive adversary is allowed to dynamically corrupt nodes or make them sleep, but such `corrupt` or `sleep` instructions take a while to be effective. For example, in practice, it may take some time to infect a machine with malware. Such a "mildly adaptive" corruption model has been formally defined in earlier works [20], where they call it the τ-agile corruption model, where τ denotes the delay parameter till `corrupt` or `sleep` instructions take effect. Intuitively, as long as τ is sufficiently large, it will be too late for an adversary to corrupt a node or make the node sleep upon seeing the next epoch's hash. By the time the `corrupt` or `sleep` instruction takes effect, it will already be well past the epoch.

The main challenge in handling mildly adaptive corruptions is the threat of a *history rewriting* attack when *posterior corruption* is possible: members of past committees may, at some point, have sold their stake in the system, and thus they have nothing to lose to create an alternative version of history.

We rely on a *checkpointing* idea to provide resilience to such posterior corruption—as long as there is no late joining or rejoining (we will discuss how to handle late joining or rejoining later). Checkpointing is a technique that has

been explored in the classical distributed systems literature [5] but typically for different purposes, e.g., in the case of PBFT [5] it was used as an efficiency mechanism. Suppose that we can already prove the consistency property as long as there is no majority posterior corruption. Now, to additionally handle majority posterior corruption, we can have alert nodes always reject any chain that diverges from its current longest chain at a point sufficiently far back in the past (say, at least W time steps ago). In this way, old committee members that have since become corrupt cannot convince alert nodes to revise history that is too far back—in other words, the confirmed transaction log stabilizes and becomes immutable after a while.

2.4 Late Joining in the Presence of Posterior Corruption

Indeed, the above approach almost would work, if there are no late spawning nodes, and if there are no nodes who wake up after sleeping for a long time. However, as mentioned earlier, handling late joining is important for a decentralized network.

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

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

Unfortunately, it turns out that this is impossible without additional trust—in fact, we can formalize the aforementioned attack and prove a lower bound (see our online full version [8]) which essentially shows that in the presence of majority posterior corruption, a newly joining node has no means of discerning a real history from a simulated one:

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

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

2.5 Putting It Altogether: Informal Overview of Snow White

In summary, our protocol, roughly speaking, works as follows. A formal description of the protocol, the parameter choices and their relations, and proofs of security are deferred to our online full version [8].

- First, there is a random oracle H that determines if a member of the present committee is a leader in each time step. If a node is leader in a time step t, he can extend the blockchain with a block of the format $(h_{-1}, \mathsf{txs}, \mathsf{time}, \mathsf{nonce}, \mathsf{pk}, \sigma)$, where h_{-1} is the previous block's hash, txs is a set of transactions to be confirmed, nonce is a random seed that will be useful later, pk is the node's public key, and σ is a signature under pk on the entire contents of the block. A node can verify the validity of the block by checking that (1) $\mathsf{H}^{\mathsf{nonce}_e}(\mathsf{pk}, \mathsf{time}) < D_p$ where D_p is a difficulty parameter[4] such that the hash outcome is smaller than D_p with probability p, and nonce_e is a nonce that is reselected every epoch (we will describe how the nonce is selected later); (2) the signature σ verifies under pk; and (3) pk is a member of the present committee as defined by the prefix of the blockchain.
- A valid blockchain's timestamps must respect two constraints: (1) all timestamps must strictly increase; and (2) any timestamp in the future will cause a chain to be rejected.
- Next, to defend against old committees that have since become corrupt from rewriting history, whenever an alert node receives a valid chain that is longer than his own, he only accepts the incoming chain if the incoming chain does not modify blocks too far in the past, where "too far back" is defined by the parameter κ_0.
- Next, a newly joining node or a node waking up from long sleep must invoke a secure bootstrapping mechanism such that it can identify the correct version of the history to believe in. One mechanism to achieve this is for the (re)spawning node to contact a list of nodes the majority of whom are alert.
- Finally, our protocol defines each contiguous T_{epoch} time steps to be an epoch. At the beginning of each epoch, committee reconfiguration is performed in the following manner. First, nodes find the latest prefix (henceforth denoted $chain_{-2\omega}$) in their local chain whose timestamp is at least 2ω steps ago. This prefix $chain_{-2\omega}$ will be used to determine the next committee—and Snow White defers to the application-layer to define how specifically to extract the next committee from the state defined by $chain_{-2\omega}$. Next, nodes find the latest prefix (denoted $chain_{-\omega}$) in their local chain whose timestamp is at least ω steps ago. Given this prefix $chain_{-\omega}$, we extract the nonces contained in all blocks, the resulting concatenated nonce will be used to seed the hash function H for the next epoch.

[4] As we discuss in our online full version [8], in practice, the next committee is read from a stabilized prefix of the blockchain and we know its total size a-priori. Therefore, assuming that an upper bound on the fraction of awake nodes (out of each committee) is known a-priori, we can set the difficulty parameter D_p accordingly to ensure that the expected block interval is sufficiently large w.r.t. to the maximum network delay (and if the upper bound is loose, then the confirmation time is proportionally slower). Although on the surface our analysis assumes a fixed expected block interval throughout, it easily generalizes to the case when the expected block interval varies by a known constant factor throughout (and is sufficiently large w.r.t. to the maximum network delay).

Resilience Condition. In the online full version [8], we will give a formal presentation of our protocol and prove it secure under the following resilience condition. We require that the majority of the committee remain honest not only during the time it is active, but also for a short duration (e.g., a handoff period) afterwards. In particular, even if the entire committee becomes corrupt after this handoff period, it should not matter to security.

In other words, we require that *for any committee, the number of alert committee members that remain honest for a window of W outnumber the number of committee members that become corrupt during the same window.* In particular, we will parametrize the window W such that it incorporates this short handoff period after the committee becomes inactive. Somewhat more formally, we require that there exists a constant $\psi > 0$ such that for every possible execution trace view, for every $t \leq |\mathsf{view}|$, let $r = \min(t + W, |\mathsf{view}|)$,

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

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

3 From Robustly Reconfigurable Consensus to PoS

We now discuss how to apply our core consensus protocol in a proof-of-stake (PoS) application. There are two challenges: (1) in a system where money can switch hands, how to make the committee composition closely track the stake distribution over time; and (2) how to distribute fees and rewards to ensure incentive compatibility.

3.1 Base Security on Distribution of Stake

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

Since our core consensus protocol allows a committee reelection opportunity once every epoch, it is possible that the distribution of the stake in the system lags behind the committee election. However, suppose that this is not the case, e.g., pretend for now that there is no money transfer, then it is simple to translate the assumptions to distribution on stake. Imagine that the application-defined $\mathsf{elect\_cmt}(\mathit{chain})$ function will output one public key for each unit of currency as expressed by the state of *chain*. If a public key has many units of coin, one could

simply output the public key pk along with its multiplicity m—and the strings $\mathsf{pk}||1, \ldots, \mathsf{pk}||m$ may be used in the hash query for determining the leader. Snow White's core consensus protocol does not care about the implementation details of $\mathsf{elect\_cmt}(chain)$, and in fact that is an advantage of our modular composition approach. In this way, our Snow White protocol retains security as long as the at any point of time, more stake is alert and will remain honest sufficiently long than the stake that is corrupt. Here when we say "a unit of stake is alert (or honest, corrupt, resp.)", we mean that the node that owns this unit of stake is alert (or honest, corrupt, resp.).

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

The Limited Liquidity Assumption. One approach to thwart such an attack is to limit the liquidity in the system—in fact, Snow White expects that the cryptocurrency layer enforces that money will not switch hands too quickly. For example, imagine that at any point of time, $a = 30\%$ of the stake is alert and will remain honest sufficiently long, $c = 20\%$ is corrupt, and the rest are sleepy. We can have the cryptocurrency layer enforce the following rule: only $\frac{a-c}{2} - \epsilon = 5\% - \epsilon$ of the stake can switch hands during every window of size $2\omega + T_{\text{epoch}} + W$. In other words, if in any appropriately long window, only l fraction of money in the system can move, it holds that as long as at any time, $2l + \epsilon$ more stake is alert and remain honest sufficiently long than the stake that is corrupt, we can guarantee that the conditions expected by the consensus protocol, that is, at any time, more committee members are alert and remain honest sufficiently long, than the committee members that are corrupt.

4 Achieving Incentive Compatibility

4.1 Fair Reward Scheme

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

Since Snow White is a blockchain-style protocol, we also inherit the well-known selfish mining attack [10,18] where a minority coalition can increase its rewards by a factor of nearly 2 in the worst case. Fortunately, inspired by the

recent work Fruitchains [19] we provide a solution to *provably* defend against any form of selfish mining attacks, and ensure that the honest protocol is a coalition-safe ϵ-Nash equilibrium. At a high level, Fruitchains provides a mechanism to *transform any (possibly unfair) blockchain that achieves consistency and liveness into an approximately fair blockchain in a blackbox manner*. Our key observation is that this transformation is also applicable to our non-proof-of-work blockchain—since we realize the same abstraction as a proof-of-work blockchain. Since we apply the essentially same techniques below as Fruitchains, we give an overview of the mechanisms below for completeness and refer the reader to Fruitchains [19] for full details.

Two Mining Processes. Like in Fruitchains [19], we propose to have two "mining" processes piggybacked atop each other. Recall that earlier each node invokes the hash function H in every time step to determine whether it is a leader in this time step. Now, we will use the first half of H to determine leadership, and use the second half to determine if the user mines a "fruit" in this time step. Additionally, we will add to the input of H the digest of a recently stablized block such that any fruit mined will "hang" from a recently stablized block—which block a fruit hangs from indicates the roughly when the fruit was "mined", i.e., the *freshness* of the fruit. Whenever an honest node finds a fruit, it broadcasts the fruit to all peers, and honest nodes will incorporate all outstanding and fresh fruits in any block that it "mines". Note that fruits incorporated in blocks are only considered valid if they are sufficiently fresh. Finally, all valid fruits contained in the blockchain can be linearized, resulting in an ordered "fruit chain".

The formal analysis conducted in Fruitchains [19] can be adapted to our setting in a straightforward manner, giving rise to the following informal claim:

Claim (Approximate fairness [19]*).* Assume appropriate parameters. Then for any (arbitrarily small) constant ϵ, in any $\frac{\kappa}{\epsilon}$ number of consecutive fruits, the fraction of fruits belonging to an adversarial coalition is at most ϵ fraction more than its fair share, as long as, informally speaking, in any committee, alert committee members that remain honest by the posterior corruption window outnumber members that become corrupt by the same window.

We refer the reader to Fruitchains [19] for a formal proof of this claim. Intuitively, this claim holds because the underlying blockchain's liveness property ensures that no honest fruits will ever be lost (i.e., the adversary cannot "erase" honest nodes' work in mining fruits like what happens in a selfish mining attack); and moreover, in any sufficiently long window, the adversary can incorporate only legitimate fruits belonging to this window (and not any fruits ϵ-far into the past or future).

Payout Distribution. Based on the above claim of approximate fairness, we devise the following payout mechanism following the approach of Fruitchain [19]. We will distribute all forms of payout, including mining rewards and transaction fees to fruits rather than blocks. Furthermore, every time payout is issued, it

will be distributed equally among a recent segment of roughly $\Omega(\frac{\kappa}{\epsilon})$ fruits. Like in Fruitchains, this guarantees that as long as at any time, there are more alert committee members that remain honest sufficiently long than corrupt committee members, the corrupt coalition cannot increase its share by more than ϵ no matter how it deviates from the prescribed protocol—in other words, the honest protocol is a coalition-safe ϵ-Nash equilibrium.

4.2 Thwarting Nothing-at-Stake Attacks

Nothing-at-stake refers to a class of well-known attacks in the proof-of-stake context [23], where participants have nothing to lose for signing multiple forked histories. We describe how Snow White defends against such attacks. Nothing-at-stake attacks apply to both signing forked chains in the past and in the present—since the former refers to posterior corruption style attacks which we already addressed earlier, in the discussion below, we focus on signing forked chains in the present.

First, as long as the adversary does not control the majority, our core consensus protocol formally guarantees that signing forked chains does not break consistency. In fact, we incentivize honest behavior by proving that the adversary cannot increase its rewards by an arbitrarily small ϵ fraction, no matter how it deviates from honest behavior which includes signing forked chains.

With ϵ-Nash equilibrium, one limitation is that players can still do a small ϵ fraction better by deviating, and it would be desirable to enforce a stronger notion where players do strictly worse by deviating. We can make sure that nothing-at-stake attackers do strictly worse by introducing a penalty mechanism in the cryptocurrency layer: by having players that sign multiple blocks with the same timestamp lose an appropriate amount of collateral—to achieve this we need that the underlying core consensus protocol achieves consistency, when roughly speaking, the adversary controls only the minority. Even absent such a penalty mechanism, players currently serving on a committee likely care about the overall health of the cryptocurrency system where they still hold stake due to the limited liquidity assumption—this also provides disincentives for deviating.

The holy grail, of course, is to design a provably secure protocol where *any* deviation, not just nothing-at-stake attacks, cause the player to do strictly worse. We leave this as an exciting open question. It would also be interesting to consider security when the attack controls the majority—however, if such a majority attacker can behave arbitrarily, consistency was shown to be impossible [21]. Therefore, it thus remains an open question even what meaningful notions of security one can hope for under possibly majority corruption.

Additional Materials in Online Full Version

In our online full version [8], we present full formalism including definitions, proofs, and lower bound results. We also present simulation and experimental results, and discuss concrete parameters in the online full version.

Acknowledgments. We gratefully acknowledge Siqiu Yao and Yuncong Hu for lending critical help in building the simulator. We thank Lorenzo Alvisi for suggesting the name Snow White. We also thank Rachit Agarwal, Kai-Min Chung, and Ittay Eyal for helpful and supportive discussions.

References

1. Personal communication with Vitalik Buterin, and public talks on sharding by Vitalik Buterin (2018)
2. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D., Brenner, M., Rohloff, K. (eds.) FC 2016. LNCS, vol. 9604, pp. 142–157. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53357-4_10
3. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: extending bitcoin's proof of work via proof of stake. In: Proceedings of the ACM SIGMETRICS 2014 Workshop on Economics of Networked Systems, NetEcon (2014)
4. Bonneau, J., Clark, J., Goldfeder, S.: On bitcoin as a public randomness source. IACR Cryptology ePrint Archive 2015:1015 (2015)
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: OSDI (1999)
6. Chen, J., Micali, S.: Algorand: the efficient and democratic ledger (2016). https://arxiv.org/abs/1607.01341
7. User "cunicula" and Meni Rosenfeld. Proof of stake brainstorming, August 2011. https://bitcointalk.org/index.php?topic=37194.0
8. Daian, P., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, online full version of this paper (2016)
9. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: an adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573 (2017). http://eprint.iacr.org/2017/573
10. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. In: FC (2014)
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. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series: Consensus system. https://dfinity.org/tech
13. 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
14. King, S., Nadal, S.: Ppcoin: peer-to-peer crypto-currency with proof-of-stake (2012). https://peercoin.net/assets/paper/peercoin-paper.pdf
15. Kwon, J.: Tendermint: consensus without mining (2014). http://tendermint.com/docs/tendermint.pdf
16. Maxwell, G., Poelstra, A.: Distributed consensus from proof of stake is impossible (2014). https://download.wpsoftware.net/bitcoin/pos.pdf
17. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)

18. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10211, pp. 643–673. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_22
19. Pass, R., Shi, E.: Fruitchains: a fair blockchain (2016, manuscript)
20. Pass, R., Shi, E.: Hybrid consensus: efficient consensus in the permissionless model (2016, manuscript)
21. Pass, R., Shi, E.: The sleepy model of consensus (2016). http://eprint.iacr.org/2016/918
22. Pass, R., Shi, E.: Rethinking large-scale consensus. In: CSF (2017)
23. Poelstra, A.: Distributed consensus from proof of stake is impossible. https://download.wpsoftware.net/bitcoin/alts.pdf
24. User "QuantumMechanic". Proof of stake instead of proof of work, July 2011. https://bitcointalk.org/index.php?topic=27787.0
25. User "tacotime". Netcoin proof-of-work and proof-of-stake hybrid design (2013). http://web.archive.org/web/20131213085759/www.netcoin.io/wiki/Netcoin_Proof-of-Work_and_Proof-of-Stake_Hybrid_Design
26. Griffith, V., Buterin, V.: Casper the friendly finality gadget. https://arxiv.org/abs/1710.09437