Hybrid Consensus: Efficient Consensus in the Permissionless Model

Rafael Pass Elaine Shi

CornellTech, Cornell, Initiative for CryptoCurrency and Contracts (IC3)*

Abstract

Consensus, or state machine replication is a foundational building block of distributed systems and modern cryptography. Consensus in the classical, permissioned setting has been extensively studied in the 30 years of distributed systems literature. Recent developments in Bitcoin and other decentralized cryptocurrencies popularized a new form of consensus in a "permissionless" setting, where anyone can join and leave dynamically, and there is no a-priori knowledge of the consensus nodes. Despite this exciting breakthrough, today's permissionless consensus protocols, often referred to as "blockchains", are known to have terrible performance, which has resulted in heated, and at times acrimonious debates in the community.

First, we show that unfortunately a performance loss is inherent for any protocol that secures against at least 1/3 corruptions in hashpower. Specifically, we formally define a new performance measure called responsiveness, and show that any *responsive* permissionless consensus protocol cannot tolerate 1/3 or more corruptions in hashpower.

Next, we show a tightly matching uppper bound. Specifically, we propose a new permissionless consensus protocol called hybrid consensus, that is responsive and secures against up to 1/3 corruptions in hashpower. Hybrid consensus's idea is to bootstrap fast permissionless consensus by combining an inefficient blockchain protocol with a fast permissioned consensus protocol. Hybrid consensus uses the blockchain not to agree on transactions, but to agree on rotating committees which in turn execute permissioned consensus protocols to agree on transactions. While the high-level idea is intuitive, formally instantiating and reasoning about the protocol exposed a multitude of non-trivial technical subtleties and challenges.

1 Introduction

The distributed systems and cryptography literature traditionally has focused on protocols whose participants are known *a priori*. Bitcoin's rapid rise to fame represents an exciting breakthrough: Bitcoin empirically demonstrated that by leveraging assumptions such as proofs-of-work, non-trivial secure applications can be built on top of a fully decentralized network where nodes join and leave freely and dynamically, and there is no pre-established trust between participants. In the remainder of the paper, we will refer to the two network settings as the *permissioned* setting and the *permissionless* setting respectively.

Informally speaking, Bitcoin's core consensus protocol, often referred to as Nakamoto consensus [46], realizes a "replicated state machine" abstraction, where nodes in a permissionless network reach agreement about a set of transactions committed as well as their ordering. Since the protocol relies on chaining of blocks of transactions, it is often referred to as the "blockchain".

*http://www.initc3.org

Achieving consensus in the traditional permissioned model turns out to be a classical distributed systems problem, and there is a long line of research that seeks to design and optimize Byzantine consensus protocols [18, 23, 42]. The fact that we can obtain consensus in a permissionless model (relying on proofs-of-work) was the novel contribution of Bitcoin. In a sense, Bitcoin popularized a new model of distributed systems that was rarely considered in 30 years of classical distributed systems literature.

Known permissionless consensus protocols such as Bitcoin's Nakamoto consensus [46], however, come at a cost. Since identities of nodes are not known *a priori*, it is imperative to defend against a Sybil attack where an attacker makes up arbitrarily many identities to outvote honest nodes. The Bitcoin protocol critically relies on proofs-of-work to roughly enforce the idea of "one vote per hashpower". Unfortunately, Bitcoin is known to have terrible performance. As Croman et al. [19] point out, the Bitcoin network can sustain at most 7 tx/sec, at a transaction confirmation time of 10+ min (*c.f.* a main-stream payment processor such as Visa handles an average rate of $2,000$ tx/sec, and a peak rate of $59,000$ tx/sec). Further, each confirmed transaction costs roughly \$1 to \$6 if we were to amortize the network's total electricity consumption over all transactions being confirmed — today, this cost is in some sense being subsidized by the speculators of Bitcoin.

This naturally raises an important question.

Is it possible to design an efficient consensus protocol in the permissionless model?

We formally explore this important question in this paper.

1.1 Our Results and Contributions

Understanding the limits: performance vs. security. To understand this formally, let us first try to understand why the Nakamoto consensus protocol [46] (adopted by Bitcoin) is inefficient. As Pass et al. [47] point out, the Nakamoto consensus protocol crucially relies on a-priori knowledge of an upper bound of the network's delay (henceforth denoted Δ) to parametrize its puzzle difficulty, and the protocol's transaction confirmation time is roughly $O(\lambda\Delta)$ to achieve $\exp(-O(\lambda))$ security failure — one way to think about this is that the block interval needs to be $O(\Delta)$ to achieve security against any constant fraction of corruption (in hashpower), and one must wait for $O(\lambda)$ blocks to obtain $\exp(-O(\lambda))$ security failure. Nakamoto is clearly inefficient since the a-priori parameter Δ needs to be set conservatively upfront to ensure the security of the protocol; and the transaction confirmation time suffers from looseness in the estimate Δ. While there are other possible metrics of efficiency, for now, we will focus on this one.

Therefore, one natural question to ask is whether we can have a protocol whose transaction confirmation time depends on only the network's actual performance, but not any a-priori known upper bound. We formally define a performance metric called *responsiveness*[1] that captures this intuition: a protocol is said to be responsive, if its transaction confirmation time depends only on the network's actual delay δ, but not on any a-prior known upper-bound Δ (or simply no a-priori upper bound is known). In particular, in practice the actual δ is often (much) smaller than the upper bound Δ. In this case, responsiveness will be a useful measure of performance.

[1]Note that responsiveness is not to be mistaken with liveness which was defined by Garay et al. [28] and Pass et al. [47] in the permissionless setting. Liveness requires that transactions be confirmed within a bounded amount of time indicated by a liveness parameter T; whereas responsiveness in turn requires that T depend only on the network's actual delay δ but not the (possibly loose) a-priori upper bound Δ.

Theorem 1. *(informal.) No secure and responsive consensus protocol can tolerate* $1/3$ *or more corruptions, even when the adversary is constrained to static corruptions — and this holds in both the classical permissioned setting (even when PKI is assumed), as well as in the permissionless setting with proofs-of-work (where corruptions are stated in terms of hashpower).*

The bad news is that we show that no *responsive* consensus protocol can tolerate $1/3$ or more corruptions, even when the adversary is constrained and can only statically corrupt parties. This lower bound result holds both in the classical permissioned setting (even when PKI is assumed), as well as in the permissionless setting with proofs-of-work — in this setting corruptions are counted in terms of hashpower and not the number of nodes. To put this in perspective, observe that Nakamoto is not responsive, but can tolerate up to $1/2$ corruptions in hashpower [47].

To intuitively understand this lower bound, recall that Dwork et al. [23] prove that in the partially synchronous model with an unknown network delay, no classical permissioned consensus protocol (even with PKI) can tolerate $1/3$ or more Byzantine corruptions. It is not hard to see that their proof still works for partial synchrony with a known upper bound Δ of the network's delay, but requiring the protocol to be responsive (in the classical permissioned setting even with PKI). It turns out that with appropriate modifications, Dwork et al.'s $1/3$ lower bound [23] extends to the permissionless setting with proofs-of-work as Sompolinsky points out [1]. We further generalize Sompolinsky's observation to the permissionless setting with proofs-of-work where the delay upper bound Δ is known to the protocol, but the protocol is required to be responsive. In other words, our lower bound can be viewed as a further generalization of Dwork et al.'s lower bound and Sompolinsky's observation. The formal proof of this lower bound is presented in Section 9.

A responsive protocol with (almost) optimal resilience. The next obvious question is the following: suppose we are willing to relax the model and assume only $< 1/3$ corruptions in hashpower, can we have a responsive consensus protocol in the permissionless setting? We answer this question in the positive.

Theorem 2. *(informal.) There exists a responsive permissionless consensus protocol that is secure against* $1/3 - \epsilon$ *corruptions in hashpower against a mildly adaptive adversary.*

To this end, we propose hybrid consensus. Hybrid consensus provides "efficiency bootstrapping" for permissionless consensus, much as the well-known hybrid encryption and OT-extension are "efficiency bootstrapping" constructions. Since classical permissioned consensus [12, 16, 18, 23, 37, 38,42,45] has been studied and optimized for decades and have been shown to achieve *responsiveness* against $1/3$ corruptions, our idea is to use a slow blockchain protocol (called snailchain) such as Nakamoto consensus [28, 46] to bootstrap fast permissioned byzantine consensus, the end result being a scalable consensus protocol in the permissionless model. For this reason, we call our protocol "hybrid consensus".

Hybrid consensus is the first known *responsive* permissionless consensus protocol, even heuristically. We formally prove that hybrid consensus achieves security against a malicious (i.e., Byzantine) adversary with the following capabilities: 1) wields roughly or $\frac{1}{3}$ fraction of the total computation power; 2) can corrupt nodes adaptively but corruptions take a while to be effective; and 3) and can reorder messages during transmission, and delay messages up to a bound of δ time steps.

Advantages of hybrid consensus. Besides the aforementioned theoretical significance of hybrid consensus, we also note some of its practical benefits.

Scheme	TX conf. time	Processing/tx	% honest
Nakamoto [46], BitcoinNG [25]	$\Theta(\lambda\Delta)$	$O(n)$	$\sim \frac{1}{2}$
Fruitchain [48] (concurrent)	$\Theta(\lambda\Delta)$	$O(n)$	$\sim \frac{1}{2}$
Hybrid consensus over Nakamoto [46]	Opt: $O(\delta)$ Worst: $O(\lambda\delta)$	$O(\lambda)$	$\sim \frac{3}{4}$
Hybrid consensus over Fruitchain [48]	Opt: $O(\delta)$ Worst: $O(\lambda\delta)$	$O(\lambda)$	$\sim \frac{2}{3}$

Table 1: **Summary of our results.** n denotes the total number of nodes (assuming all nodes have equal hashpower); Δ denotes a pre-determined upper-bound on the network's transmission delay; δ denotes the actual delay of the network; λ is the security parameter for attaining $2^{-\lambda}$ security failure.

- **Latency.** Recall that Nakamoto has a block interval of 10 minutes. To attain $1 - 2^{-\lambda}$ security (e.g., important for high-valued transactions), one must wait for $O(\lambda)$ blocks thus incurring an even longer transaction confirmation time. As Pass et al. [47] show, for a realistic network delay bound $\Delta = 10$ seconds as suggested by measurement studies [21], a 10-minute block interval is necessary to resist a 49.57% attack, and roughly a block interval of 30 seconds is necessary to resist a 1/3 attack.

  By contrast, hybrid consensus achieves a transaction confirmation time of $O(\delta)$ in the optimistic case, and $O(\lambda\delta)$ in the worst-case (i.e., under adversarial attacks) where δ is the network's actual delay. Note that unlike Nakamoto, our optimistic-case performance is independent of the security parameter λ.

  Pragmatically speaking, the following factor also potentially makes Δ much bigger than δ which speaks in favor of hybrid consensus. In practice, block propagation in an open-enrollment, permissionless setting is achieved through an overlay network; and thus Δ must be parametrized to be an upper bound of the overlay network's delay. Therefore, Δ must not only account for multiple hops of direct IP links, but also account for potentially adversarial attacks targeted at the overlay routing protocol. By contrast, if hybrid consensus is deployed, once committee members are discovered, they can possibly communicate with each other as well as other nodes over direct IP links — in this case, the optimistic transaction confirmation time can depend on the δ which is the latency of these direct IP links.

- **Transaction processing.** Consider smart contract applications whose most popular embodiment is Ethereum [54]. Today's blockchain protocols require *all miners* to execute the smart contract program for each transaction, thus incurring a linear in n processing cost — this makes existing blockchain protocols unscalable to large deployments. Hybrid consensus reduces the transaction processing cost to $O(\lambda)$, since regardless of how large n is, only a small committee of size $O(\lambda)$ need to execute the smart contracts and process the transactions.

- **Throughput and lower cost per confirmed transaction.** Besides being responsive, known permissioned consensus protocols, which have been studied and optimized for decades, have also been empirically demonstrated to achieve relatively high throughout. For example, Croman et al. recently show that with about 100 PBFT [18] nodes deployed across multiple data centers on Amazon AWS, one can possibly achieve a transaction throughput of $10,000+$ tx/sec, and transaction confirmation time on the order of seconds.

We therefore expect that hybrid consensus should improve the throughput of permissionless consensus in comparison with known protocols such as Nakamoto. Although hybrid consensus does not eliminate the proof-of-work — in fact, we prove that absent extra setup assumptions, any secure permissionless consensus protocol must perform proofs-of-work infinitely often (see Section 9) — the electricity consumed can now be amortized over a larger set of transactions effectively reducing the cost per confirmed transaction.

Security tradeoff. Hybrid consensus tolerates only 1/3 corruptions (in hashpower) as opposed to Nakamoto's 1/2 in exchange for responsiveness. We have argued that such a tradeoff is inevitable.

Besides the tradeoff in the protocol's resilience, hybrid consensus also makes a reasonable relaxation in the corruption model. While Nakamoto resists a fully adaptive adversary, obviously any protocol that down-selects to a size-$O(\lambda)$ committee cannot tolerate fully adaptive attacks since the adversary can simply observe who get elected in the committee and then target the committee members.

Instead, hybrid consensus defends against *mildly adaptive* corruptions: the adversary is allowed to observe the protocol's interactions and then decide who to corrupt, but corruptions take a while to be effective. This appears a reasonable assumption in practice since it takes a while to infect a machine with a malware, e.g., through zero-day exploits or social engineering attacks. We stress that since our 1/3 lower bound holds even for static adversaries, hybrid consensus is a tightly matching upper bound in this sense. We leave it as an open research question whether there exists a responsive protocol in the permissionless setting that secures against fully adaptive adversaries.

Comparison with closely related works. Although the idea of combining permissionless consensus and permissioned consensus has been discussed in the community (e.g., the recent work by Decker et al. [20] and the concurrent and independent work ByzCoin [35]), to the best of our knowledge, no prior work has provided a formal treatment. As our work shows, combining permissioned and permissionless protocols is non-trivial both in terms of construction and in terms of proving security. Without a formal analysis, it is not clear what earlier/concurrent approaches achieve. For example, in the concurrent work Byzcoin [35], participants may not agree on the PBFT committee with constant probability, thus breaking consensus. In both Decker et al. [20] and Byzcoin [35], consensus can be broken with probability 1 in the worst case if the adversary controls more than $\frac{1}{4}$ of the computation power, since Nakamoto's chain quality suffers from a loss resulting from a selfish mining attack (although Byzcoin actually claims resilience to a $\frac{1}{3}$ attack!). Further, Decker et al. [20] does not achieve responsiveness. We defer a more detailed comparison to Section 1.4.

Therefore, our work is distinct in the following senses: 1) we provide the first provably secure protocol that achieves *responsiveness* (i.e., transaction confirmation time depends on actual network performance, not an a-priori upper bound on the network's delay) in the permissionless model with proofs-of-work; 2) we are the first to formalize the precise model and security guarantees.

1.2 Intuition and Overview

To aid understanding, we give an intuitive, informal summary of our technical roadmap. We build on the formal snailchain abstractions proposed by Garay et al. [28] and Pass et al. [47]. There are two possible realizations of such a snailchain abstraction, the original Nakamoto consenus [28, 46, 47], and the more recent Fruitchain [48] protocol. We can use either snailchain instantiation for hybrid consensus, however there is an important difference (which is overlooked by earlier works such as Byzcoin [35] leading to incorrect claims about their protocol's resilience), namely, we can only achieve security against $1/4 - \epsilon$ corruptions (in hashpower) if we adopted Nakamoto as the underlying snailchain; however, with Fruitchain as the underlying snailchain, we can achieve security against $1/3 - \epsilon$ corruptions. For the remainder of the paper, we will first use Nakamoto as the underlying snailchain since most readers are more familiar with Nakamoto — and later we will show how to use Fruitchain as a drop-in replacement of Nakamoto in hybrid consensus which immediately allows us to resist a $1/3-\epsilon$ attack. This approach, in fact, demonstrates the compelling advantage of modular protocol design and composition.

Imprecisely speaking, a snailchain satisfies the following properties. Note that a formal description of these properties are presented in Section 4.1.

- *Consistency.* All honest nodes' chains agree with each other except for the trailing λ blocks where λ is the security parameter. Further, a node's chain agrees with its future self.
- *Chain quality.* Among any consecutive λ blocks in an honest node's chain, a sufficient fraction of the blocks are mined by honest miners.
- *Chain growth.* Honest nodes' chains grow at a steady rate, neither too fast nor too slow.

Warmup: electing a static committee. Fundamentally, a blockchain such as Nakamoto consensus (henceforth denoted snailchain) relies on proofs-of-work puzzles such that nodes can establish Sybil-resilient identities. Our first idea is to leverage snailchain to elect a static committee. To do this, honest nodes run the blockchain for $\mathsf{csize} + \lambda$ blocks where $\mathsf{csize} = \Theta(\lambda)$ denotes the targeted committee size, and λ denotes a security parameter. At this moment, an honest node would remove the trailing, unstablized λ blocks from its local chain, and call the miners of the first csize blocks the BFT committee[2].

Roughly speaking, such a protocol can be proven secure under a static corruption model due to the following.

- Due to the consistency property of snailchain, all honest nodes agree on the same BFT committee. We stress that it is important to remove the trailing λ unstable blocks since otherwise honest nodes will have differing opinions on who should be the BFT committee (e.g., due to possible forks in the snailchain) — in this case we cannot guarantee the protocol secure[3].
- Due to the chain quality property of snailchain, with appropriate overall parameters, we can ensure that more than 2/3 of the committee members are honest which is sufficient to ensure the security of the permissioned BFT protocol.
- Due to the chain growth property of snailchain, it will not take too long for the BFT committee to be formed.

[2]If multiple blocks are mined by the same miner, that miner can simply act as multiple virtual nodes in the BFT protocol — for this reason, the protocol works for $n < \lambda$ as well (and so does our final scheme hybrid consensus).

[3]Note that in comparison, the concurrent and independent work Byzcoin [35] does not explicitly make this observation, and therefore their protocol and security guarantees appear under-specified.

Finally, committee members sign any transaction committed as well as its sequence number. For any node that was not elected as a committee member, it can simply count $\lceil \frac{1}{3}|\mathsf{csize}|\rceil$ number of signatures from committee members for deciding its own output log. Since more than 2/3 of the committee members are honest, there is at least one honest committee member if at least $\lceil \frac{1}{3}|\mathsf{csize}|\rceil$ vouch for the same transaction and its sequence number (i.e., its relative position within the output log). It is not hard to see that responsiveness holds for this simple protocol assuming that the BFT protocol employed is responsive.

An interesting paradox. Despite the apparent simplicity of this proposal, this scheme is nonetheless thought provoking. One intriguing apparent paradox is the following: *since we do not rely on the* snailchain *to agree on transactions any more, why not stop running the* snailchain *after the committee is elected?* Although this proposal may seem tempting initially, one quickly realizes that it is not secure. In particular, for any protocol that stops performing proofs-of-work after bounded polynomial amount of time, an adversary can always create a simulated execution that is identically distributed as the real execution, such that a node that spawns late is unable to distinguish the real execution from the simulated one. We formalize this lower bound in Section 9, and show that any secure permissionless consensus protocol must call proofs-of-work infinitely often (even for static security and the synchronous model).

Mildly agile adversaries and rotating committees. The aforementioned scheme with a static committee fails to be secure against an adaptive adversary, since the adversary can simply corrupt the committee once it is elected. Unfortunately, any scheme that down-selects from n nodes to a λ-sized committee must be vulnerable to such adaptive attacks.

In reality, however, corruption of a node is typically not instant, since it takes a while to infect an otherwise clean host. We therefore define a slightly relaxed and nonetheless realistic corruption model henceforth referred to as τ-agile corruptions. In this model, roughly speaking, an adversary can issue "target corrupt" instructions to nodes; however, a node that receives "target corrupt" does not become corrupt until τ time steps later.

We show a positive result under this τ-agile corruption model. Our key idea is to rely on rotating committees. When an honest node's chain reaches $R \cdot \mathsf{csize} + \lambda$ in length, the R-th committee is elected by first removing the trailing λ number of blocks, and then from this pruned chain, we elect the last csize blocks' miners as the committee. The idea is that if an adversary targets a committee member once he mines a block (that will allow him to be included in a committee), it will be too late. With an appropriate τ, by the time the node actually becomes corrupt, the committee's term will have been ended, and the next committee will have already taken over!

Henceforth for convenience, we will say that each committee serves for a *day*, and outputs a *daily log*. Our hybrid consensus protocol essentially outputs the concatenation of these daily logs.

Defending against retroactive attacks. Given long enough time, an adversary can eventually corrupt sufficiently many BFT committee members, and at this point, the adversary can forge BFT committee members' signatures over any message of its choice. Therefore, signatures from BFT committee members are of no worth after a long enough time. Under the possibility of such retroactive attacks, nodes that spawn late cannot rely on counting committee members' signatures for deciding logs that are too ancient.

To address this challenge, we rely on an *on-chain stamping* technique. When honest BFT committee members terminate their BFT instances, they would sign a hash of the daily log and propose the tuple as a transaction to the underlying snailchain. We prove that under appropriate parameter choices, sufficiently many honest BFT committee members' daily log hashes get stamped on snailchain in a timely manner (in particular before the nodes actually become corrupt). In this way, nodes who spawn late can recover the correct hash of past daily logs from the underlying snailchain (instead of counting signatures from ancient committee members who may by then be corrupt). We stress that the on-chain stamping is only for late-joining nodes to recover historical daily logs. Transaction confirmation online requires only counting signatures from the present committee, and need not wait for the on-chain stamping to take place.

Chain quality and tolerated corruption. It would be ideal if mining were a random lottery, where for each block, nature draws a winner at random. If this were indeed the case, we could achieve perfect chain quality, i.e., roughly $1-\alpha$ chain quality if α fraction of nodes are corrupt. Unfortunately, several previous works [26, 28, 47] have shown that Nakamoto consensus cannot be thought of as a perfect lottery due to a selfish mining attack. When honest nodes mine a block, they announce the block immediately, but corrupt nodes need not follow this rule. In a selfish-mining attack, roughly speaking, when corrupt nodes mine a block B^* off the currently longest chain, they withhold the block B^* from the public and continue to mine on its own private fork. If at some point honest nodes happen to mine a new block B off chain, at this moment the adversary immediately releases the block B^*, and combined with a network rushing attack, the block B^* will get to other nodes more quickly than B. In this manner, the adversary has successfully erased the work of honest nodes — and in fact every time corrupt nodes mine a block, they have an opportunity to perform such a block withholding attack and erase honest nodes' work. Consequently, Nakamoto consensus would require roughly 3/4 overall honest to achieve 2/3-chain quality (and 2/3-chain quality is needed to ensure that 2/3 of the BFT committee are honest).

The loss of resilience arising from chain quality loss can be mostly avoided, if instead of Nakamoto consensus, we adopt Fruitchain [48] as the underlying blockchain protocol. As shown by Pass and Shi [48], Fruitchain provably defends against such a selfish-mining attack, and therefore attains almost perfect chain quality, i.e., it achieves roughly $(1-\alpha)$-chain quality with any $\alpha < 1/2$ corruption under typical parametrizations. Therefore, hybrid consensus over Fruitchain requires only 2/3 overall honest hashpower (approximately) to achieve security.

A note on responsiveness. It may seem counter-intuitive that since our scheme is responsive, why do we still need a-priori knowledge of Δ, an upper bound of the network's delay? In particular, can we simply choose the parameter Δ to be infinity? Upon closer examination, our agility parameter τ and the protocol's warmup time T_{warmup} will both depend on Δ. If we choose a bigger Δ, the underlying snailchain would adopt a more difficult puzzle and hence tolerate a higher fraction of corrupt nodes; but on the other hand, we are trading off agility and the protocol's warmup time.

The standard notion of partial synchrony was defined by Dwork et al. [23] where the protocol does not know the network's delay. Dwork et al. [23] pointed out that if the protocol knew the network's delay, it could simply wait for the delay parameter and treat the delay parameter as a synchronous round. We stress that our protocol does not wait for the network delay parameter as a synchronous round (which inherently would not meet our responsive goal). We stress that Dwork et al. [23]'s definition of partial synchrony does not rule out the meaningfulness of our model where the protocol knows an a-priori upper bound on the network's delay but aims to achieve responsiveness.

1.3 Numerous Technical Challenges

While the aforementioned idea seems intuitive at first sight, realizing it formally reveals numerous technical subtleties/challenges which we describe below.

Committee switchover: technical subtleties and contributions. Roughly speaking, when honest nodes' chain lengths reach $R \cdot \mathsf{csize} + \lambda$, the $(R-1)$-th BFT committee should end and the R-th BFT committee should take over. How to perform committee switchover turns out to be non-trivial. The most naive approach would be to have non-overlapping committees, where the new committee waits for the previous daily log hash to be stamped on snailchain before starting its BFT instance. In this naive approach, there is a liveness gap at the end of each day, during which the old committee is no longer committing transactions but the new committee is blocked waiting for the stamping to complete. Note that this is already better than just a snailchain, since we only need to wait once per day, and besides this short window, we achieve responsiveness. However, we would like a solution that always guarantees responsiveness. To this end we allow transient *overlap* of two (or more) committees.

In hybrid consensus, when honest nodes chain length reaches $R \cdot \mathsf{csize} + \lambda$, the old committee (i.e., $(R-1)$-th committee) initiate a termination procedure; and a new committee (i.e., R-th committee) initiate a new BFT instance and immediately start committing transactions. As termination is not an instant operation, the previous committee members may continue to output transactions until its termination procedure completes. For this reason, there may be a short overlapping window when two (or more) BFT instances run concurrently. During this short window, although the latter BFT instance may be committing transactions, the committed transactions are withheld and not written into the output LOG, until the previous day's final log is ready and written to LOG. This ensures that the latter daily log always follows the previous daily log in the output LOG.

Due to such concurrently executing BFT instances, we need that the underlying BFT protocol be concurrently composable. Unfortunately, Lindell et al. [40] demonstrate the impossibility of concurrently composable Byzantine agreement! We circumvent this impossibility by observing that due to the way hybrid consensus bootstraps from the snailchain, nodes have common knowledge of committee members' freshly chosen public keys — and in such a setting, concurrent security is indeed possible.

Adversarial selective opening of committees. One technical challenge we encountered is that in the protocol, honest nodes generate public keys and associate them with blocks to be mined. Recall that the committee is formed by taking the miners of consecutive csize blocks. It is important to note that the choice of the committee is subject to influence by the adversary, since the adversary can launch block withholding or selfish-mining attacks to erase an honest node's block if it does not like the specific public key. We refer to such an attack as "selective opening of committees".

Unfortunately as we explain later, the most natural property-based security definition is insufficient for proving security under such selective opening attacks. Therefore, we define a strengthened notion of security which requires a blackbox reduction that converts any adversary attacking the BFT's consistency or liveness properties to an adversary that breaks the security of the underlying signature scheme. Informally, the security notion says that there is a p.p.t. reduction $\mathcal{B}$ such that for any p.p.t. adversary $\mathcal{A}$, for any set of public keys, if $\mathcal{A}$ can break the BFT's consistency or liveness properties for these public keys, then $\mathcal{B}$ which makes blackbox calls to $\mathcal{A}$ can forge a signature on behalf of an honest node which has one of these public keys (later in the formal definition, this adversary $\mathcal{A}$ is actually the environment $\mathcal{Z}$). We formally show that this strengthened security notion is sufficient for handling the selective opening attack.

Although we need a strengthened security notion for the underlying BFT protocol, we note that well-known existing constructions such as PBFT [18] would naturally satisfy this stronger notion of security.

Formal framework for composing consensus protocols. Finally, our work demonstrates a framework for composing consensus protocols. Due to growing interests in cryptocurrencies and their applications, there is a clear and increasing appetite from the community for mixing and composing multiple consensus protocols to enable new applications — therefore our approach can be of independent interest.

To achieve this, we rely on a Universal Composition-like formal framework that enables composition of cryptographic protocols. We use the UC framework [13,14,17] in a way such that we define formal properties of protocols rather than relying on ideal behavioral specifications. A protocol's security requirements are defined specifically in terms of properties over honest nodes' outputs to the environment $\mathcal{Z}$. All of these security properties will be preserved when hybrid consensus is sequentially or concurrently composed with any other protocol. Additionally, we adopt a modular composition approach when presenting our hybrid consensus protocol. Precisely formalizing the security properties of subprotocols and building blocks turns out to be challenging, and exposes numerous subtleties which we shall discuss in the technical parts of the paper.

1.4 Related Work

Scaling decentralized consensus. The scalability of Bitcoin and decentralized, permissionless cryptocurrencies is a highly visible issue, and has resulted in heated, and at times acrimonious debates in the community [19,53]. The cryptocurrency community have proposed various incremental patches to alleviate the scalability pressure in the near term, including adjusting the block size and others [5,24,29,30,55].

Eyal et al. propose BitcoinNG [25], where a slow snailchain protocol is used to elect a single leader every epoch, and the leader is in charge of incorporating and linearizing transactions during its appointment. In essence, BitcoinNG can be viewed as pipelining block transmission by breaking it up and spreading block transmission over time — effectively reducing upper-bound on the worst-case delay Δ in Nakamoto consensus. BitcoinNG still requires nodes to wait for $\Theta(\lambda)$ blocks in the underlying snailchain (referred to as key blocks in BitcoinNG) to stabilize for transactions to be confirmed where λ is the security parameter. In comparison, hybrid consensus is responsive, and the transaction confirmation time in the optimistic case is only $O(\delta)$ where δ is the network's actual delay, not the a priori known upper bound Δ. BitcoinNG did not give a formal treatment of their protocol, but it is conceivable that their protocol can be proven to realize a permissionless consensus abstraction.

Side-chain [6] is another notable effort at addressing Bitcoin's scalability painpoint. Side-chain's idea is to support consensus protocols off the main Bitcoin blockchain, and the currency in the side chain is pegged to Bitcoin. The side-chain protocol lacks formal guarantees, and the protocol description and implementation remain somewhat incomplete.

Various other approaches [8, 36, 49–52] have been proposed recently to attain consensus in different variants of "decentralized" settings with varying trust assumptions.

Closely related works. Although recent or concurrent works have attempted at a superficially similar idea of combining permissionless and permissioned consensus [20,35,41], none of these works have fully formulated and precisely enunciated their security guarantees. Furthermore, existing protocols either do not achieve responsiveness [20,35], or assume very different setup assumptions [41].

In recent work, Decker et al. propose PeerCensus [20], where they combine a blockchain and classical BFT to strengthen the probabilistic consistency guarantee of snailchain. Although their protocol bears a superficial resemblance to hybrid consensus, they do not achieve responsiveness because their protocol requires solving proof-of-work puzzles to create the next block, and the puzzle's difficulty needs to be parametrized knowing an upper bound of the network's delay to attain a reasonable chain quality which appears necessary for their permissioned BFT protocol to be secure. Decker et al. also did not provide formal security analysis — since their protocol bootstraps from snailchain, it would tolerate at most $1/4$ corruption under a rushing attack.

The concurrent and independent works ByzCoin [35] and SCP [41] also bear superficial resemblance to hybrid consensus, but again our approach is fundamentally different. SCP assumes a fundamentally different trust model — SCP does not rely on proof-of-work puzzles, and in light of our lower bound in Section 9, they must assume a fundamentally different trust model. Upon more careful examination, the SCP paper implicitly assumes that the network can somehow agree on an initial directory committee. The SCP paper suggests that they can achieve this by having early nodes announce their identities, but since this would require consensus to start with, it appears that this directory committee should be regarded as an extra trusted setup assumption. Therefore, SCP is not in the permissionless model. The SCP paper also did not fully formalize their approach and precisely what formal guarantees they attain is not clear.

The independent work of Byzcoin [35] does not achieve responsiveness. ByzCoin's idea is to improve BitcoinNG by implementing BitcoinNG's leader with a PBFT committee. Their protocol appears underspecified, but it appears that they elect the committee using the miners of the last k blocks. Note that such a committee is not necessarily globally consistent, and the ByzCoin protocol does not fully specify how they handle possibly inconsistent views of the committee. Similarly, ByzCoin also did not formalize their approach. Even if their approach (or some modification thereof) could somehow be proven secure through likely highly non-trivial ways, the ByzCoin protocol *does not achieve responsiveness* just like BitcoinNG, since nodes still need to wait for $\Theta(\lambda)$ number of key blocks to stabilize for the transaction to be confirmed with all but $2^{-\lambda}$ probability, and the block interval needs to be parametrized as $c\Delta$.

In conclusion, *hybrid consensus* is the first to show how to combine permissioned and permissionless consensus in a formally correct manner, as well as the first to achieve responsiveness in the permissionless model.

Permissioned consensus. Consensus protocols in the permissioned model have been extensively investigated by the community in the past 30 years [11,12,18,23,27,33,37–39]. These works typically consider three different models: 1) the synchronous model [22] where protocols proceed in rounds, and messages delivered in one round are guaranteed to arrive at the recipient at the beginning of the next round; 2) the partial synchrony model [23] where the network has a bounded delay parameter but the protocol does not know this delay; and 3) the asynchronous model [11,12] where the network's delay may grow unbounded.

Our network model is akin to the standard notion of partial synchrony [23] but not the same. Although we allow the protocol to know an a-priori upper bound on the network's delay, we aim to achieve responsiveness. We stress that any protocol that waits for the network delay and treats the delay as a synchronous round inherently cannot be responsive. If the protocol did wait for the network delay as a synchronous round, this would indeed translate to the synchronous model, however, our responsiveness requirement makes the design of protocols non-trivial in our network model.

Earlier works on permissioned consensus have also considered group reconfiguration. For example, Vertical Paxos [38] and BFT-SMART [9] allow nodes to be reconfigured in a dynamic fashion. These works consider group reconfiguration for a related but somewhat different purpose. It would be interesting to investigate whether these techniques can be adapted to our setting to perform the switchover of committee members. We point out, however, that earlier group reconfiguration techniques do not prove security under the selective opening attack (in fact, most of these works do not adopt a cryptographically sound framework of reasoning). If we are to adapt these techniques, a new, cryptographically sound treatment is necessary.

Distributed systems and cryptography. Consensus and distributed systems interact closely with cryptography such as multi-party computation (MPC). On one hand, multi-party computation (MPC) essentially relies on broadcast or distributed consensus primitives to achieve consistency and potentially livenss, often referred to as guaranteed output in the MPC context. On the other hand, distributed consensus protocols often make use of cryptography to ensure security. For example, the *authenticated Byzantine* model [22] makes use of digital signatures, and cryptographers refer to this setup assumption as the *public-key infrastructure* [13, 14, 17].

On the other hand, the *non-authenticated Byzantine* model [39] in distributed systems is actually referred to as the *authenticated channels* model by cryptographers [13, 14, 17]. When protocols employ computationally secure cryptographic primitives, implicitly we assume that the network's delay must be polynomially bounded in the security parameter (but can be an unbounded polynomial in the asynchrony case), since we cannot guarantee security for protocols that run exponentially long. When distributed consensus protocols make use of computationally secure primitives, a best practice is to rely on computational reductions to prove the security of the protocols — it has become well-understood that modeling cryptography as the most natural blackbox without careful scrutiny can be error-prone and flawed [2–4, 7, 10, 15, 31, 32, 43, 44].

Our paper demonstrates such an approach where we adopt the protocol composition frameworks [13, 14, 17] developed by the cryptographers to reason about distributed systems protocols — we show that doing so is necessary in particular through the handling of the selective opening attack. Such issues can easily be overlooked if we did not adopt a formal, cryptographically sound framework of reasoning.

2 Technical Roadmap

Before presenting our formalism, we first give an informal technical roadmap to aid understanding.

2.1 Execution Model

We consider a model where nodes are Interactive Turing Machines (ITM). The execution of the ITM system proceeds in atomic units called time steps. In each time step, each node receives messages, performs a polynomial amount of computation, and then sends messages to other nodes.

Proofs-of-work. We assume that our ITM system is augmented with a proof-of-work. There is a random oracle denoted by a pair $(\mathsf{H}, \mathsf{H.ver})$. Without loss of generality, we assume that each node can query H only once in each time step, but can query $\mathsf{H.ver}$ an unbounded polynomial number of times. In practice, if a node has more than unit computation power, it can simply be considered as a set of multiple nodes.

Network assumptions. We assume a partially synchronous model, where any message sent by an honest node is guaranteed to arrive at all honest nodes within δ time steps. The adversary is allowed to reorder messages subject to the above constraints.

Our protocol needs to know a possibly loose upper bound of δ to parametrize the scheme (in particular, to parametrize the puzzle difficulty of the underlying snailchain). We henceforth use the notation Δ to denote this pre-determined upper bound. Our protocol achieves *responsiveness*: even though we use the a-priori upper bound Δ as an input parameter, our protocol achieves transaction confirmation time that depends on the network's actual delay δ, not the possibly loose upper bound Δ. This requirement makes our setting fundamentally different than the synchronous model — since if the protocol simply takes Δ time steps to be a synchronous round, the protocol would not be responsive.

Open enrollment and mildly agile corruption. Although we allow the adversary to adaptively decide which nodes to corrupt, corruption does not take place instantly. In our model, when the adversary issues a "target corrupt" instruction to a node, it takes τ time for the node to actually become corrupt. Once a node actually becomes corrupt, the adversary can kill the node. Finally, new nodes can spawn at any time.

2.2 Definition: Permissionless Consensus

We would like to realize a state machine replication abstraction in the permissionless model — henceforth we refer to this abstraction as *permissionless consensus*. In a permissionless consensus protocol, each node outputs a LOG in every time step — this LOG represents the set of committed transactions. Two important security requirements, namely, *consistency* and *liveness* must be guaranteed with overwhelming probability.

- *Consistency*: Consistency includes the following:
  - *Common prefix.* Suppose that an honest node i outputs LOG to $\mathcal{Z}$ at time t, and an honest node j (same or different) outputs LOG' to $\mathcal{Z}$ at time t', it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that a node i is honest at time t and $t' \geq t$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.

- *Liveness*: Suppose that transactions TXs is input to an honest node at time $t \geq T_{\mathrm{warmup}}$. Then, if any node that is honest at time $t' \geq t + T_{\mathrm{confirm}}$ outputs LOG at time t', it holds that $\mathsf{TXs} \subseteq \mathsf{LOG}$.

Intuitively, liveness says that transactions get included in honest nodes' LOGs within T_{confirm} time. There are two liveness parameters T_{confirm} and T_{warmup}. T_{warmup} is the protocol's warmup time; and T_{confirm} is the maximum wait time for a transaction (proposed after T_{warmup}) to be confirmed.

2.3 Building Blocks

We make use of two main building blocks, a permissioned BFT protocol with a strengthened notion of security, and a slow blockchain denoted snailchain. We now informally define these abstractions.

Underlying snailchain. We assume an underlying snailchain that satisfy *consistency*, *chain quality*, and *chain growth*. An intuitive description of these properties have been presented in Section 1.2. These properties will be formalized in Section 4.1.

Permissioned BFT. Permissioned BFT protocols have been extensively studied in the distributed systems literature. Typically, known permissioned BFT [18, 23, 42] satisfy exactly the same consistency and liveness guarantees as defined earlier — but here for the permissioned setting.

There is, however, an even more interesting technical subtlety in formalizing the permissioned BFT abstraction. A property-based security definition turns out to be insufficient due to a selective opening attack. In particular, the adversary is allowed to first look at nodes' public keys, and then adaptively influence the way the committee is chosen. In Section 4.2, we argue that there exists a (somewhat contrived) permissioned BFT protocol that is provably secure under property-based definitions, but would be completely broken if subject to adversarial selective opening.

As a result, we define a strengthened security notion for our underlying permissioned BFT building block. Not only do we require that the aforementioned consistency and liveness properties be satisfied with overwhelming probability, we need the following stronger statement:

> There exists a p.p.t. reduction $\mathcal{B}$ such that given any p.p.t. adversary that can break the BFT's security properties over any set of public keys, the reduction $\mathcal{B}$ which makes blackbox calls to this adversary can forge a signature on behalf of an honest party.

We defer the discussion of the formal definitions and technical details to Section 4.2. It is not hard to see that the augmented PBFT protocol also naturally satisfies this strengthened security notion.

2.4 Hybrid Consensus Overview

In this section, we describe our construction informally and in an intuitive manner. Formalizing the construction takes a fair amount of effort and will be described later in Section 5. We present notations in Table 2. For convenience, in this paper, we first describe our hybrid consensus protocol assuming Nakamoto as the underlying snailchain. Later in Section 6, we will argue that we can use Fruitchains [48] (almost) as a drop-in replacement for Nakamoto, and obtain better parameters.

Variable	Meaning
tx	a transaction
ℓ	sequence number of a transaction within each BFT instance
LOG	the totally ordered log each node outputs, LOG is always populated in order
log	log of one BFT instance, referred to as daily log
$\mathsf{log}[\ell : \ell']$	transactions numbered ℓ to ℓ' in log
$\mathsf{log}[: \ell]$	$\mathsf{log}[1 : \ell]$
λ	security parameter
α	adversary's fraction of hashpower
δ	network's maximum actual delay
Δ	a-priori upper bound of the network's delay (typically loose)
csize	committee size, our protocol sets $\mathsf{csize} := \lambda$
th	$\mathsf{th} := \lceil \mathsf{csize}/3 \rceil$, a threshold
$\mathsf{lower}(R), \mathsf{upper}(R)$	$\mathsf{lower}(R) := (R-1)\mathsf{csize} + 1$, $\mathsf{upper}(R) := R \cdot \mathsf{csize}$
chain	a node's local chain in the underlying $\mathsf{snailchain}$ protocol
$\mathsf{chain}[: -\lambda]$	all but the last λ blocks of a node's local chain
$\text{MinersOf}(\mathsf{chain}[s : t])$	the public keys that mined each block in $\mathsf{chain}[s : t]$. It is possible that several public keys belong to the same node.
$\{\mathsf{msg}\}_{\mathsf{pk}^{-1}}$	a signed message msg, whose verification key is pk
T_{bft}	liveness parameter of the underlying BFT scheme

Table 2: Notations

Recall that earlier, as a warmup exercise, we described a protocol which elects a static BFT committee by running $\mathsf{snailchain}$ for $\mathsf{csize} + \lambda$ blocks and electing the miners of the first csize blocks as the BFT committee. The drawback of this protocol is the following: if the adversary can now adaptively target the set of committee members to corrupt, it can break the security of the protocol by corrupting only a small number of nodes.

This problem can be mitigated through rotating committees, and our resulting protocol can be proven secure against a mildly agile adversary that takes a while to corrupt nodes. In other words, if the adversary targets the committee members to corrupt, by the time the committee members actually become corrupt, it is already too late since the next committee will have taken over. On the other hand, if the adversary corrupts aimlessly without knowing a priori which random committee members will be selected, it can only successfully corrupt $< \frac{1}{3}$ committee members when the random choices are announced. Our formal protocol description is given in Section 5.3 but here we explain the intuition first.

Rotating committees. As soon as a node collects $R \cdot \mathsf{csize} + \lambda$ blocks in the underlying $\mathsf{snailchain}$, it enters the R-th "day". Note that all nodes may perceive the day start slightly out of sync, but due to the consistent length property of $\mathsf{snailchain}$ (which is part of chain growth), all nodes perceive the day start very close from each other except with $\mathsf{negl}(\lambda)$ probability.

On each day, a new committee is in charge. For an honest node with a local chain (of the underlying $\mathsf{snailchain}$), he defines set of committee members for day R as

$$\mathsf{comm}_R := \text{MinersOf}(\mathsf{chain}[(R-1)\mathsf{csize} + 1 : R \cdot \mathsf{csize}])$$

It follows directly that all honest nodes agree on the same comm_R due to the common prefix and common self-prefix properties of snailchain. As mentioned earlier, it is possible that several public keys in $\text{MinersOf}(\mathsf{chain}[(R-1)\mathsf{csize}+1 : R\cdot\mathsf{csize}])$ belong to the same node. In this case, the same node will act as multiple nodes, each with a different public key, in the BFT consensus.

Daily operations. We now describe the daily operations of both committee members and non-members.

- *Committee members.* On each day R, the R-th committee will run a BFT instance. A committee member will continue running the BFT protocol to commit transactions until it receives a "stop" instruction at which point a special stopping procedure is invoked. Therefore, committee members will output committed transactions gradually over time. Committed transactions will populate a node's *daily log* denoted log_R.

  Whenever an honest committee member adds a new transaction tx to its log_R, it will sign the tuple (R, ℓ, tx) where R denotes the current day and ℓ denotes the sequence number of tx within the day. The honest committee member then gossips the signed tuple to the network.

- *Committee non-members.* Non-members hear signed transactions from the network. Whenever a non-member hears that a tuple (R, ℓ, tx) has been signed by more than $\frac{1}{3}$ fraction of comm_R members, he adds the tx to its log_R:

  $$\text{if } \mathsf{log}_R[\ell] \text{ is not populated } : \quad \mathsf{log}_R[\ell] := \mathsf{tx}$$

  Observe that a committee non-member can write to its $\mathsf{log}_R[\ell]$ out of order since messages may be received out of order. However, a transaction cannot be processed until all preceding transactions have been committed. Later, when we define each node's output LOG, we enforce that transactions are always written to LOG in sequential order — and this can be achieved if committee non-members output the longest contiguous prefix of log_R to its LOG.

Committee switchover. Whenever a node enters a new day denoted $R + 1$, it performs a committee switchover procedure as follows. Below R denotes the previous day.

- *Member of the previous committee.* If a node is a member of the R-th committee denoted comm_R, it inputs a special, signed `stop` transaction to the previous BFT— a node may run multiple BFT virtual nodes, in which case one signed `stop` transaction is input to each BFT virtual node. When the BFT's log collects sufficiently many of these `stop` transactions signed by distinct committee member public keys, the log is finalized and all later transactions are ignored. At this moment, we say that the previous BFT has terminated. When the previous BFT has terminated, a member of comm_R will sign the tuple $(R, |\mathsf{log}_R|)$ and gossip the signed tuple to the network. This allows non-members of comm_R to determine when log_R ends.

  Further, an honest committee member signs $(R, \mathsf{hash}(\mathsf{log}_R))$ where hash is collision-resistant, and proposes the signed tuple to the underlying snailchain— we henceforth refer to this action as *stamping*. As we explain later, timely stamping secures against an adversary that can retroactively corrupt old committee members in the future.

  At this point, the honest comm_R member outputs "done".

- *Non-member of the previous committee.* If the node is not a committee member of the BFT instance for day R, it waits for more than $\frac{1}{3}$ fraction of comm_R members to vouch for a tuple (R, ℓ). When this happens, it knows that ℓ is the final sequence number of log_R. Therefore, it simply waits for all of $\mathsf{log}_R[:\ell]$ to be populated before outputting "done".

Whenever a node (either member of non-member of comm_R) outputs "done", its log_R is said to be *final*. We note that honest comm_{R+1} members start the new BFT instance for day $R+1$ as soon as they perceive the start of day $R+1$, and without waiting for their log_R to be final. This ensures that all comm_{R+1} members start the new BFT instance within a short duration from each other (whereas waiting for log_R to be final will incur extra drift in the start time of the next BFT instance).

Output log. Nodes need to collect their daily logs into a final log denoted LOG — and this final log must satisfy the properties defined in Section 3.2. In particular, this final log LOG outputs transaction in increasing order, since one may not be able to process a transaction until all preceding transactions have been accumulated. As we pointed out, committee non-members may write to its daily log log_R out of order. Further, when the BFT instance on the R-th day is started, the previous BFT instance may not have fully completed, and therefore log_R would have to wait for log_{R-1} to be final.

Therefore, to output the final log LOG in order, we simply define LOG to contain:

- A maximal, consecutive sequence of daily logs log_1, log_2, ..., log_{r-1} all of which must be *final*.
- The longest contiguous prefix of the daily log log'_r.

Bootstrapping. Nodes that join late must perform a bootstrapping procedure to catch up. First, a bootstrapping node will need to identify a maximal underlying chain for the $\mathsf{snailchain}$ protocol. This allows the node to securely determine what each committee has been so far.

A bootstrapping node also needs to catch up on its local LOG. This has to be dealt with care, since a bootstrapping node cannot simply rely on collecting enough signatures from each old committee any more — in particular, keep in mind that the adversary can retroactively corrupt old committees. Old committee members that later become corrupt can sign arbitrary messages. Therefore, informally, signatures from committee members are only trustworthy if they are timely. Signatures from very old committee members are no longer trustworthy.

To allow late joining nodes to catch up securely, recall that honest committee members stamp (R, h) tuples to the blockchain as soon as their reign is over where $h = \mathsf{hash}(\mathsf{log}_R)$ is a hash of the final daily log. In this way, bootstrapping nodes can recover the correct hashes of old daily logs from $\mathsf{snailchain}$. Afterwards, the bootstrapping node can simply query for daily logs that match these hashes to populate its local LOG.

Note that an adversary can still corrupt old committee members and make them stamp arbitrary things to $\mathsf{snailchain}$. However, an adversary cannot revert what the correct values an honest committee member has already stamped, if retroactive corruptions can only take place sufficiently late. Therefore, if a committee member stamped conflicting tuples on $\mathsf{snailchain}$, a bootstrapping node simply suppresses all later occurrences. Our security theorem parametrizes τ as in τ-agility appropriately, such that more than $\frac{2}{3}$ fraction of any committee must remain honest till all tuples are stably stamped on $\mathsf{snailchain}$ (see Lemmas 3 and 4).

2.5 Modular Protocol Composition and Formal Reasoning

To aid formal reasoning and presentation, our protocols are described through a modular composition approach.

Daily offchain consensus. We first construct an intermediate abstraction called DailyBFT which describes what committee members and non-members do respectively to agree on each day's daily log. Our hybrid consensus protocol will fork one instance of DailyBFT$[R]$ for each day where R is the day number as well as the unique session identifier for the DailyBFT instance. Hybrid consensus then concatenates the daily logs output by these DailyBFT instances.

In a DailyBFT instance, each elected committee member spawns one or more BFT virtual nodes, depending on how many of its public keys were included in the committee. If a node has not been elected as the committee, it would count signatures from committee members to decide its daily log.

We formalize and prove the security properties of DailyBFT: Below are a few things to keep in mind when reading the detailed formalism presented in Section 5.2.

- While the lower-level BFT building block states its security properties (i.e., consistency and liveness) for committee members only, in DailyBFT, these security properties are extended to non-committee members as well.

- The lower-level BFT building block assumes that all committee members are spawned before the BFT instance starts. In DailyBFT, however, these security properties need to extend to committee non-members who potentially spawned later (but not too late).

- On the other hand, DailyBFT does not guarantee security (i.e., consistency and liveness) for nodes that join too late, since committee members may become corrupt far out in the future, at which point committee members can sign arbitrary tuples, and thus late joining nodes cannot rely on counting signatures to decide their daily logs any more. We defer it to hybrid consensus to deal with this attack, by having late joining nodes recover ancient daily logs by examining daily log hashes stamped on the snailchain.

- DailyBFT offers a `keygen` abstraction: upon every `keygen` query, DailyBFT generates and outputs a new miner public key pk — the hybrid consensus protocol will incorporate pk into the block being mined. Later, DailyBFT will receive input from the environment which set of pks have been selected as committee members. This is where the adversarial selective opening of committee keys is handled. The security proof of DailyBFT therefore makes use of the *strong* security of the BFT protocol, to argue that the BFT protocol, when run inside DailyBFT as a subprotocol, will respect the stated security properties including consistency and liveness — otherwise one could construct a reduction that breaks signature security.

- Finally, in comparison with BFT, DailyBFT additionally implements a termination procedure that satisfies two properties, *timely termination*, and *termination agreement*. Timely termination says that the BFT protocol terminates quickly upon honest nodes receiving `stop` instructions. Termination agreement says that all honest nodes output identical final logs upon termination.

  Termination is realized by having honest BFT virtual nodes input a special, signed `stop` transaction to the underlying BFT. When $\lceil |\mathsf{comm}|/3 \rceil$ `stop` transactions signed by distinct committee member keys have accumulated in the log, all later transactions are ignored and the log is finalized.

Hybrid consensus. We now describe our final product, the hybrid consensus protocol. Hybrid consensus consumes multiple instances of DailyBFT where rotating committees agree on daily logs. Hybrid consensus primarily does the following:

- It manages the spawning and termination of DailyBFT instances effectively using snailchain as a global clock that offers weak synchronization among honest nodes;
- Recall that each DailyBFT instance does not ensure security for nodes that spawn too late, since committee members can become corrupt far out in the future at which point they can sign arbitrary tuples. Therefore, hybrid consensus introduces an on-chain stamping mechanism to extend security guarantees to even nodes that spawn late. Specifically, committee members stamp their signed daily log hash onto snailchain when their BFT instance terminates. Nodes that spawn late will rely on this on-chain stamp to identify and recover ancient daily logs in the past (rather than counting off-chain signatures from committee members).

2.6 Main Theorems

We now state our main theorems whose detailed proofs are presented in Section 8. Our hybrid consensus can be instantiated using either Nakamoto or Fruitchain as the underlying snailchain, resulting in the following theorems.

Theorem 3 (Hybrid consensus over Nakamoto)**.** *For any (arbitrarily small) constant $\epsilon > 0$, let $\alpha = \frac{1}{4} - \epsilon$, then for every n, δ, there exists sufficiently small $\rho_0 := \Theta(\frac{1}{\delta n})$ such that* $\mathsf{HybridConsensus}^{\lambda}$ *with Nakamoto as the underlying* snailchain *and with mining difficulty parameter $\rho < \rho_0$ is secure w.r.t. any p.p.t. $\Gamma^{\mathsf{hc}}_{\rho}(n, \alpha, \delta, \tau)$-admissible $(\mathcal{A}, \mathcal{Z})$, where*

$$T_{warmup} := 8\lambda/3n\rho, \quad T_{confirm} := O(\lambda\delta)$$

Theorem 4 (Hybrid consensus over Fruitchain)**.** *For any (arbitrarily small) constant $\epsilon > 0$, let $\alpha = \frac{1}{3} - \epsilon$, there exists a constant $\eta > 0$ (related to ϵ), a suitable $\kappa = \Theta(\lambda)$, and for every n, δ, there exists a sufficiently small $\rho := \Theta(\frac{1}{\delta n})$, such that* $\mathsf{HybridConsensus}^{\lambda,\eta}$ *over Fruitchain with parameters (ρ, κ), is secure w.r.t. any p.p.t. $\Gamma^{\mathsf{hcfruit}}_{\rho,\eta}(n, \alpha, \delta, \tau)$-admissible $(\mathcal{A}, \mathcal{Z})$, where*

$$1.5\lambda(1 + \frac{1}{\eta})/(1 - 5\eta)n\rho, \quad T_{confirm} := O(\lambda\delta)$$

Note that in both the above theorems, the T_{confirm} parameter is stated for the worst-case transaction confirmation time even when under attack. In the optimistic case, hybrid consensus achieves a transaction confirmation time of $O(\delta)$. Further, although the theorem is stated in terms of the network's actual delay δ, in practice we must predetermine an upper bound estimate (denoted Δ) of δ to parametrize the puzzle difficulty level ρ. As long as Δ is indeed an upper bound on δ, security is guaranteed by the above theorem, and the scheme achieves responsiveness, i.e., the transaction confirmation time does not depend on the upper bound Δ, but the actual network delay δ. As mentioned earlier, if we choose a looser estimate Δ (i.e., a greater value of Δ), the scheme will then be parametrized with a more difficult puzzle — on one hand this allows us to tolerate potentially a higher fraction of corrupt nodes; on the other hand, the agility parameter τ as well as the protocol's warmup time will increase accordingly.

3 Problem Definitions

Strongly negligible functions. All security failures in this paper will be expressed as (exponentially) strongly negligible functions in terms of some security parameter $\lambda \in \mathbb{N}$. We say that a function $\mathsf{negl}(\cdot)$ is strongly negligible, if there exist some constants $c_0 > 0, c_1$, such that for all $\lambda \in \mathbb{N}$, $\mathsf{negl}(\lambda) \leq \exp(-(c_0\lambda + c_1))$. In the remainder of the paper, we simply use the term negligible for simplicity, but all uses of it can be automatically replaced by strongly negligible.

3.1 Formal Model

Execution model. We assume the following execution model:

- *Interactive Turing Machines.* We assume a standard Interactive Turing Machine (ITM) model [13, 14, 17] often adopted in the cryptography literature (but augmented with proof-of-work as explained later). There is an underlying, global clock that increments over time; each clock tick is referred to as an *atomic time step.*

  Nodes can perform unbounded polynomial amount of computation in each atomic time step, as well as send and receive polynomially many messages. Although not explicitly noted in the paper, nodes receive inputs from an environment $\mathcal{Z}$ and send their outputs to an environment.

- *Proof-of-work.* We assume that there is a random oracle denoted by a pair $(\mathsf{H}, \mathsf{H.ver})$. In each atomic time step, each node can make at most one H oracle query, but an unbounded (polynomial) number of $\mathsf{H.ver}$ queries. If there are multiple instances of the blockchain protocol, we assume that each protocol instance has its own independent random oracle. The environment cannot directly query the random oracle, but can query the random oracle through the help of the adversary.

- *Corruption.* At any point of time, the environment $\mathcal{Z}$ can communicate with corrupt nodes in arbitrary manners. This also implies that the environment can see the internal state of corrupt nodes. Corrupt nodes can deviate from the prescribed protocol arbitrarily, i.e., exhibit byzantine faults. All corrupt nodes are controlled by a probabilistic polynomial-time adversary denoted $\mathcal{A}$, and the adversary can see the internal states of corrupt nodes.

  For honest nodes, the environment cannot observe their internal state, but can observe any information honest nodes output to the environment by the protocol definition. Details on corruption models will be described later.

- *Network delivery.* The adversary is responsible for delivering messages between nodes. We assume that the adversary is capable of *delaying* or *reordering* messages, possibly subject to certain restrictions as described below.

τ-agile corruption. In standard adaptive corruption models, whenever the environment wishes to corrupt a node, the corruption takes place instantly. Our protocol is proven secure under a slightly relaxed adaptive corruption model which we refer to as τ-agile corruption. Roughly speaking, τ-agile corruption says that it takes a short while for the environment to actually corrupt a node. More formally, the environment is allowed to corrupt and spawn new nodes according to the following procedures:

- *Delayed corruption.* We assume that the environment can adaptively corrupt a node but with the following restrictions. To corrupt a node i, the environment must issue a "target corrupt" instruction to node i at some point of time denoted t. Node i does not become corrupt immediately, but rather remains honest till $t+\tau$, and becomes corrupt at time $t+\tau$ — at this point, the the corrupt node i communicates arbitrarily with the environment and can deviate arbitrarily from the protocol.

- *Killing a corrupt node.* Once a node actually becomes corrupt, the environment can issue a "kill" instruction to kill the node. A killed node is no longer live. The environment cannot kill honest nodes directly without corrupting them first.

- *Spawning new nodes.* The environment is also allowed to spawn fresh nodes, either honest or corrupt ones. A node spawned at time t_{spawn} is considered live at time t_{spawn}. Spawning a corrupt node is equivalent to increasing the hashpower of the adversary which takes place instantly. If an honest node is spawned, the environment must follow the delayed corruption procedure if it wishes to corrupt this node later. An honest, newly spawned node starts running the main protocol.

We say that a node (that has been spawned and has not been killed) can be in three mutually exclusive states:

1. *Intact*: An honest node that has not received a "target corrupt" instruction.

2. *Pre-corrupt*: An honest node that has received a "target corrupt" instruction, but has not become corrupt yet.

3. *Corrupt*: A node that is either spawned to be corrupt, or spawned to be honest, but then received a "target corrupt" instruction and actually became corrupt.

Both intact nodes and pre-corrupt nodes are considered *honest*.

Henceforth, whenever we say that "an honest node i performs certain actions at time t", we mean that the node i is honest at time t. For example, if we say that an honest node outputs a message to the environment $\mathcal{Z}$ at time t, we implicitly mean that the node is honest at time t (but may become corrupt later). Alternatively, if we say that an honest node performs an action, it means that the node is honest at the time it performs the action, although it may become corrupt sometime in the future.

Fully adaptive corruption and static corruption. We note that 0-agile corruption is equivalent to the *fully adaptive* corrupt model where the environment $\mathcal{Z}$ can corrupt nodes instantly. Under a fully adaptive corruption model, a node is intact iff it is honest.

We also define static corruption in the permissionless model — static corruption is a weaker corruption model and is only used for the purpose of proving our lower bounds — note that assuming a weaker corruption model yields stronger lower bounds. We assume that in the static corruption model, environment can spawn honest or corrupt nodes at any time. However, once an honest node is spawned, the environment is unable to corrupt it later on.

Gossip network model. We assume that all messages sent by honest nodes are spread over a gossip network and can eventually be heard by all other honest nodes. For nodes that join at time T, it will receive all messages sent by honest nodes after time T. An honest node need not know the identities of other nodes in the network to gossip a message to all other nodes.

The adversary cannot drop or modify messages by honest nodes, but is allowed to reorder or delay messages subject to certain restrictions. The adversary may selectively deliver a message to a subset but not all of the honest nodes.

We assume that the identity of a message's sender is unknown. Messages can be signed, but an honest node does not know the correspondence between public keys and physical identities of nodes.

We define the following types of gossip networks which impose different restrictions on the adversary's ability to delay and reorder messages:

- *Synchronous model.* In the synchronous model, messages gossiped by an honest node at time t are guaranteed to arrive at all honest nodes, possibly out of order, in time step $t+1$. Moreover, historical messages are delivered to newly spawned nodes instantly.

- *δ-partially synchronous model.* In a δ-partially synchronous model, messages gossiped by an honest node at time t are guaranteed to arrive at all honest nodes, possibly out of order, by time $t+\delta$. Moreover, historical messages are delivered to newly spawned nodes instantly.

  More formally, suppose an honest node gossips a message in time step $t \leq t^*$, then if a node i is honest in time step $t^* + \delta$, then it must have received the message.

Note that in practice, honest nodes can implement a historical transcript retrieval service — this way, a node can obtain a copy of the entire historical transcript when spawning a consensus instance. It is not hard to see that if any honest node remains honest and live for at least 3δ time, no historical transcript will be lost.

As mentioned later in Section 3.2, we allow our protocol to know a possibly loose upper bound Δ on the network's delay, since the underlying snailchain must know such an upper bound for parametrizing the mining difficulty. However, we require that the protocol be *responsive*, i.e., its actual performance must depend only on the network's actual δ value, not the loose upper bound Δ.

Randomized protocol execution and probability space. Let Π be some protocol, let $\mathcal{A}, \mathcal{Z}$ be probabilistic polynomial-time (or p.p.t. for short) algorithms, and let $\lambda \in \mathbb{N}$. Let $\textsc{exec}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$ be a random variable denoting the joint view of all nodes (i.e., all their inputs, random coins, and messages received, including those from random oracles) in the above execution.

Let property be a function that takes as input a fixed view and outputs either 0 or 1. Throughout the paper, whenever we say that some property holds for $\textsc{exec}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$ with probability p, we formally mean that

$$\Pr\left[\mathsf{view} \xleftarrow{\$} \text{EXEC}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{property}(\mathsf{view}) = 1\right] = \mathsf{p}$$

where probability is taken over all random coins of $\mathcal{A}$, $\mathcal{Z}$, all honest nodes, and all random oracles.

Compliant execution. We often impose constraints on the adversary $\mathcal{A}$ and $\mathcal{Z}$ to prove security properties of protocols. We therefore define what we consider as compliant executions, in terms of constraints on the pair $(\mathcal{A}, \mathcal{Z})$.

Definition 1 ((n, δ, τ)-valid $(\mathcal{A}, \mathcal{Z})$)**.** *We say that the pair $(\mathcal{A}, \mathcal{Z})$ is (n, δ, τ)-valid w.r.t. protocol* Π *if $\mathcal{A}$ and $\mathcal{Z}$ are probabilistic polynomial-time algorithms such that for every $\lambda \in \mathbb{N}$, the following properties hold with probability* 1 *for* $\text{EXEC}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$*:*

1. At any point of time, the number of live nodes[4] is n;
2. $\mathcal{A}$ delays messages from honest nodes in at most δ time steps;
3. Once an honest node receives input "target corrupt" from the environment, it takes at least τ time before the node becomes corrupt.

Definition 2 ($(n, \alpha, \delta, \tau)$-valid $(\mathcal{A}, \mathcal{Z})$)**.** *We say that the pair $(\mathcal{A}, \mathcal{Z})$ is $(n, \alpha, \delta, \tau)$-valid w.r.t. protocol* Π*, such that $(\mathcal{A}, \mathcal{Z})$ is (n, δ, τ)-valid as per Definition 1, and moreover, for any $\lambda \in \mathbb{N}$, in any* view *in support of* $\text{EXEC}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$*, it holds that at any time, no more than α fraction of the live nodes are either in corrupt or pre-corrupt states.*

Throughout the paper, although not noted explicitly, all parameters including n, α, δ, and τ are functions in the security parameter λ. Further, for notational simplicity, in this paper we do not explicitly define validity rules for transaction inputs. However, it is not difficult to extend our definitions to incorporate transaction validity rules like Garay et al. [28] and Pass et al. [47].

Our protocols may be secure under choices of parameters including n, α, δ, and τ that satisfy specific constraints. We therefore define the notion of Γ-admissibility where Γ is a function that imposes constraints on parameter choices.

Definition 3 (Γ-admissible)**.** *Let Γ be a binary function in parameters n, α, δ, τ. We say that a p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ is Γ-admissible w.r.t. some protocol* Π *iff there exists $n, \delta > 0$ and $\alpha, \tau \geq 0$ such that*

- $\Gamma(n, \alpha, \delta, \tau) = 1$;
- $(\mathcal{A}, \mathcal{Z})$ is $(n, \alpha, \delta, \tau)$-valid w.r.t. Π by Definition 2;

3.2 Problem Definition: Permissionless Consensus

Our $\mathsf{HybridConsensus}$ protocol realizes a state machine replication abstraction in the permissionless model — henceforth we refer to this abstraction as *permissionless consensus.* In a permissionless consensus protocol, nodes maintain a LOG over time that is a list of transactions; and further,

[4] In principle, it is not difficult to relax this requirement and allow the number of nodes to vary up to a constant factor, but the chain growth parameter of the underlying $\mathsf{snailchain}$ needs to be adjusted accordingly.

consistency and liveness are guaranteed. Our permissionless consensus abstraction is a variant of the "public ledger" abstraction adopted by Garay et al. [28] and Pass et al. [47]. The differences are non-essential. In particular, we essentially incorporate the transaction mempool into our abstraction, such that maintaining the transaction mempool is not left to the caller. By contrast, the public ledger abstraction by Garay et al. [28] and Pass et al. [47] leaves it to the caller to maintain a transaction mempool.

More formally, a permissionless consensus satisfies the following abstractions.

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

Security definitions. Let p.p.t. algorithms $(\mathcal{A}, \mathcal{Z})$ be $(n, \alpha, \delta, \tau)$-valid w.r.t. a permissionless consensus protocol Π. Let T_{warmup}, T_{confirm}, $T_{\text{bootstrap}}$ be polynomial functions in λ, n, α, δ, and Δ. We say that a permissionless consensus protocol Π is secure w.r.t. $(\mathcal{A}, \mathcal{Z})$ with parameters $(T_{\text{warmup}}$, T_{confirm}, $T_{\text{bootstrap}})$, if there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, with $1 - \mathsf{negl}(\lambda)$ probability, the following properties hold for $\text{EXEC}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$:

- *Consistency*: Consistency includes the following properties:
  - *Common prefix.* Suppose that an honest node i outputs LOG to $\mathcal{Z}$ at time t, and an honest node j (same or different) outputs LOG' to $\mathcal{Z}$ at time t', it holds that either $\mathsf{LOG} \prec \mathsf{LOG}'$ or $\mathsf{LOG}' \prec \mathsf{LOG}$. Here the relation $\prec$ means "is a prefix of". By convention we assume that $\emptyset \prec x$ and $x \prec x$ for any x.
  - *Self-consistency.* Suppose that a node i is honest at time t and $t' \geq t$, and outputs LOG and LOG' at times t and t' respectively, it holds that $\mathsf{LOG} \prec \mathsf{LOG}'$.
- *Liveness*: Suppose that the environment $\mathcal{Z}$ inputs TXs to an honest node at time $t \geq T_{\text{warmup}}$. Suppose that some node i spawned at time t_{spawn} and remains honest till $t' \geq \max(t_{\text{spawn}} + T_{\text{bootstrap}}, t + T_{\text{confirm}})$. Let LOG be the output of node i at time t', it holds that any $\mathsf{tx} \in \mathsf{TXs}$ is included in LOG.

  Intuitively, liveness says that transactions input to an honest node gets included in their LOGs within T_{confirm} time. Further, T_{warmup} is referred to as the protocol's warmup time.

Note that the above definitions are with respect to a specific $(\mathcal{A}, \mathcal{Z})$ pair. However, our main theorem later will state the security of the $\mathsf{HybridConsensus}$ protocol for any p.p.t. $(\mathcal{A}, \mathcal{Z})$ as long as they respect certain constraints.

Remark 1. *For our hybrid consensus protocol, $T_{bootstrap} = 0$, i.e., newly spawned nodes are bootstrapped instantly. Therefore we often omit writing the term $T_{bootstrap} = 0$ without risk of ambiguity. However, our problem definition admits a polynomial $T_{bootstrap}$ since this will allow us to prove a stronger lower bound.*

Responsiveness. We say that a permissionless consensus protocol is responsive if the liveness parameter T_{confirm} depends only on the network's actual δ, not on the loose upper bound Δ that is used to parametrize the protocol.

4 Building Blocks

4.1 Underlying Blockchain Protocol $\mathsf{snailchain}$

Our main scheme is of an efficiency bootstrapping nature, where we bootstrap from an underlying, slow blockchain denoted $\mathsf{snailchain}$ to obtain a permissionless consensus protocol with fast transaction confirmation and high throughput.

We assume the underlying slow consensus protocol denoted $\mathsf{snailchain}$ (e.g., Bitcoin's Nakamoto consensus [46]) realizes a "blockchain" abstraction, which can be considered as a special-case permissionless consensus protocol as defined in Section 3.2.

Abstraction. We assume that the $\mathsf{snailchain}$ protocol provides the following input/output abstraction.

Inputs. In each time step, the environment $\mathcal{Z}$ inputs to each honest node $(\mathsf{recs}, \mathsf{pk})$ where recs denotes a set of records and pk denotes a public key.

Outputs. In each time step, honest nodes output to the environment the following:

$$\mathsf{chain} := \{(\mathsf{recs}_i, \mathsf{pk}_i)\}_i$$

Useful notions. We define the following notions that will be useful later.

Local chain. In each time step, an honest node outputs to the environment some chain, for simplicity we refer to this chain as the honest node's local chain in this time step.

Intact and honest blocks. Given chain which denotes an honest node's local chain at some time t, we can define whether each block in chain is intact (or honest resp.) with respect to a prefix of chain. A block $\mathsf{chain}[j] := (\mathsf{recs}, \mathsf{pk})$ is said to be intact (or honest resp.) w.r.t. a prefix $\mathsf{chain}[: j']$ where $j' < j$ if there exists some node i intact (or honest resp.) at some time $t' \leq t$, such that 1) node i output chain' to $\mathcal{Z}$ at time t' such that $\mathsf{chain}[: j'] \prec \mathsf{chain}'$, and 2) $\mathcal{Z}$ input $(\mathsf{recs}, \mathsf{pk})$ to node i at time $t' + 1$. Informally, for an honest party's chain denoted chain, a block $B := \mathsf{chain}[j]$ is intact (or honest resp.) w.r.t. a prefix $\mathsf{chain}[: j']$ where $j' < j$, if earlier there is some honest node who received the block B as input when its local chain contains the prefix $\mathsf{chain}[: j']$.

Security definitions. Similar to earlier works [25, 28, 34, 47], we define the following properties for a $\mathsf{snailchain}$ protocol. In all of the following, the probability is defined over randomness consumed by all honest nodes, the environment $\mathcal{Z}$, as well as the adversary $\mathcal{A}$ in the execution.

Suppose that $(\mathcal{A}, \mathcal{Z})$ is $(n, \alpha, \delta, \tau)$-valid w.r.t. $\mathsf{snailchain}$. Let W_C, W_Q, W_G be polynomial functions in λ. Let Q, G, G' be polynomial functions in $\lambda, n, \alpha, \delta$, and Δ. We say that a $\mathsf{snailchain}$ protocol satisfies *W_C-consistency, (W_Q, Q)-chain quality, and (W_G, G, G')-chain growth* w.r.t. to $(\mathcal{A}, \mathcal{Z})$, if there exists a negligible function negl such that for any $\lambda \in N$, except with $\mathsf{negl}(\lambda)$ failure probability, the following properties hold for $\textsc{exec}[\mathsf{snailchain}](\mathcal{A}, \mathcal{Z}, \lambda)$:

- *Consistency.* For any node i that is honest at time t, and any j (same or different) that is honest at time $t' \geq t$, let chain denote what node i outputs to $\mathcal{Z}$ at time t, and let chain' denote what node j outputs to $\mathcal{Z}$ at time t', it holds that

$$\mathsf{chain}[: -W_C] \prec \mathsf{chain}'$$

- *Chain quality.* Let chain denote what an honest node outputs to $\mathcal{Z}$ at any time t. Then for any $\lambda_1 \geq W_Q(\lambda)$, if $|\mathsf{chain}| \geq \lambda_1$, it holds that for any $i \leq |\mathsf{chain}| - \lambda_1$, at least $\lceil Q\lambda_1 \rceil$ number of blocks in $\mathsf{chain}[i : i + \lambda_1]$ are intact w.r.t. $\mathsf{chain}[: i - 1]$. In other words, at any time, among any $\lambda_1 \geq W_Q(\lambda)$ consecutive window of blocks in an honest node's output chain, at least Q fraction of the blocks are intact w.r.t. the prefix of the window.

- *Chain growth.* In every time step, the following properties hold:

  1. *Consistent length.* Suppose that an honest node outputs chain at time t. It holds that any honest node must output a chain of length at least $|\mathsf{chain}|$ at any $t' \geq t + \delta$.
  2. *Chain growth.* Suppose that an honest node i outputs chain at time t, an honest node j (same or different) outputs chain' at time $t' \geq t$, suppose that $G \cdot (t' - t) \geq W_G(\lambda)$, it holds that $G \cdot (t' - t) \leq |\mathsf{chain}'| - |\mathsf{chain}| \leq G' \cdot (t' - t)$.

  Therefore, intuitively, chain growth says that 1) honest nodes have roughly the same chain length, and 2) honest nodes' chains cannot grow too slowly.

For convenience, we now define a derived property called *liveness.* If i is an honest node at t, let $(\mathsf{recs}_i^t, \_)$ denote what the environment $\mathcal{Z}$ inputs to honest node i at time t. Let rec be some record. We say that $\mathcal{Z}$ *proposes* rec to node i at time t if $\mathsf{rec} \in \mathsf{recs}_i^t$.

We say that a $\mathsf{snailchain}$ protocol satisfies liveness w.r.t. $(\mathcal{A}, \mathcal{Z})$ with liveness parameter T_{snail}, if there exists a negligible function negl such that for any $\lambda \in \mathbb{N}$, with $1 - \mathsf{negl}(\lambda)$ probability, the following holds for $\textsc{exec}[\mathsf{snailchain}](\mathcal{A}, \mathcal{Z}, \lambda)$:

- *Liveness.* Let rec be some record. If for every honest node i, for each $t' = t, t + 1, \ldots$ the environment $\mathcal{Z}$ proposes rec unless rec is already contained in node i's output[5] $\mathsf{chain}[: -\lambda]$, then we have that at time any $t_1 \geq t + T_{\text{snail}}$, if an honest node outputs chain', then rec must be included in $\mathsf{chain}'[: -\lambda]$.

  Intuitively, liveness simply says that if the environment $\mathcal{Z}$ continues to input the same record rec to all honest nodes for T_{snail} amount of time, then rec will get included in all honest nodes' local chain in at most T_{snail} time.

Lemma 1 (Liveness as a derived property)**.** *For any p.p.t. algorithms $\mathcal{A}, \mathcal{Z}$, any $Q > 0, G' \geq G > 0$, any W_C, W_Q, W_G such that $W_C(\lambda) + W_Q(\lambda) + \lambda \geq W_G(\lambda)$ for all λ, if* $\mathsf{snailchain}$ *satisfies W_C-consistency, (W_Q, Q)-chain quality and (W_G, G, G')-chain growth w.r.t. $(\mathcal{A}, \mathcal{Z})$, then* $\mathsf{snailchain}$ *satisfies liveness w.r.t. $(\mathcal{A}, \mathcal{Z})$ with liveness parameter $T_{snail} = (W_C + W_Q + \lambda)/G$.*

Note that for convenience of application later, we define a slightly modified version of liveness in comparison with Garay et al. [28], Pass et al. [47], and Fruitchain [48]. It is straightforward to see that our liveness notion is implied by those adopted in Pass et al. [47] and Fruitchain [48].

Underlying $\mathsf{snailchain}$ is an non-responsive permissionless consensus. It is not hard to see that that our underlying $\mathsf{snailchain}$ abstraction defined in Section 4.1 can be regarded as a special-case instantiation of a "permissionless consensus" protocol. In particular, each node's LOG can be the ordered list of records in $\mathsf{chain}[: -\lambda]$. Such a permissionless consensus protocol is non-responsive since we need to set the expected block interval to be $\Theta(\Delta)$ under typical parameters, where Δ is an a priori upper bound on the network's delay. Therefore $T_{\text{confirm}} := \Theta(\lambda\Delta)$.

[5]In practice, we can perform the following optimization within the Nakamoto protocol: the honest algorithm can suppress a record rec if it is already contained in the longest chain that it tries to extend.

4.1.1 Nakamoto as the underlying snailchain

Garay et al. [28] prove that Nakamoto consensus [46] satisfies variants of the above properties assuming a fully synchronous model, i.e., messages are delivered instantly and cannot be delayed by the adversary. Pass et al. [47] strengthen these properties and prove that Nakamoto consensus satisfies them in a δ-partially synchronous network under appropriate conditions on δ.

Below we restate the main theorem of Pass et al. [47] for the underlying snailchain. Let α and β denote the fraction of corrupt and honest nodes respectively where $\alpha + \beta = 1$, and let ρ denote the probability that a single node mines a valid block in one time step. ρ is closely related to the mining difficulty parameter.

- Let $p := 1 - (1-\rho)^{\beta n}$ denote the probability that some honest node succeeds in mining a block in a single time step.
- Let $q := \alpha n \rho$ denote an upper bound on the expected number of blocks that the adversary can mine in a single time step.
- Let $\gamma := \frac{p}{1+\delta p}$ which can be thought of as a version of p discounted by the network's delay δ.

Definition 4 (Admissible parameters for snailchain $\Gamma_\rho^{\mathsf{snail}}$)**.** *We define* $\Gamma_\rho^{\mathsf{snail}}(n, \alpha, \delta, \tau) = 1$ *iff the following holds:*

- $n > 0$, $\delta > 0$, $\tau \geq 0$ are all polynomial functions in λ; $\alpha > 0$ is a constant;
- There exists a constant $\eta > 0$ such that $p(1 - (2\delta + 2)p) \geq (1 + \eta)q$.

Theorem 5 (Nakamoto as the underlying snailchain [47])**.** *For any constants* $\eta_0, \eta_1, \eta_2, \eta, \rho > 0$*, let* $Q = 1 - (1+\eta_0)\frac{q}{\gamma}$*, let* $G = \gamma/(1+\eta_1)$*, let* $G' = (1+\eta_2)n\rho$*, the Nakamoto consensus protocol [46,47] (henceforth referred to as* snailchain*) parametrized with mining difficulty parameter* ρ *satisfies* $\eta\lambda$*-consistency,* $(\eta\lambda, Q)$*-chain quality, and* $(\eta\lambda, G, G')$*-chain growth w.r.t. to any p.p.t.* $(\mathcal{A}, \mathcal{Z})$ *that is* $\Gamma_\rho^{\mathsf{snail}}$*-admissible w.r.t.* snailchain*.*

Typical parametrizations. Typically in practice, we would set the puzzle's difficulty parameter $\rho := \Theta(\frac{1}{\Delta n})$ where Δ be an a-priori known upper bound of the network's delay δ. Under such typical parametrizations, we would need roughly 3/4 overall honest to ensure roughly 2/3-chain quality.

Corollary 1 (Nakamoto as the underlying snailchain [47])**.** *For any (arbitrarily small) constant* $\epsilon > 0$*, let* $\alpha = \frac{1}{4} - \epsilon$*, then for every* n, δ*, there exists sufficiently small* $\rho_0 := \Theta(\frac{1}{\delta n})$ *such that for any constant* $\eta > 0$*,* $\eta' > 0$*, Nakamoto's protocol with mining difficulty parameter* $\rho < \rho_0$ *satisfies* $\eta'\lambda$*-consistency,* $(\eta'\lambda, Q)$*-chain quality and* $(\eta'\lambda, G, G')$*-chain growth w.r.t. any* $\Gamma_\rho^{\mathsf{snail}}$*-admissible* $(\mathcal{A}, \mathcal{Z})$ *where*

$$Q > \frac{2}{3}, \quad G = \frac{3}{4}n\rho, \quad G' = (1+\eta)n\rho$$

Or more simply (and informally) put, for every $\alpha = \frac{1}{4} - \epsilon$ for an arbitrarily small constant $\epsilon > 0$, there exists an appropriately parametrized Nakamoto consensus protocol that achieves $Q > \frac{2}{3}$ chain quality.

4.1.2 Fruitchain as the underlying snailchain

The problem with using Nakamoto as the underlying snailchain is chain quality loss. Due to a selfish mining attack, Nakamoto requires $3/4 + \epsilon$ overall honest to attain 2/3-chain quality which is needed for the elected committees to be 2/3 honest. Since hybrid consensus takes a modular approach, we can instead use a drop-in replacement, the Fruitchain [48] protocol, which realizes (almost) the same formal abstraction as Nakamoto.

At a high level, the Fruitchain protocol runs a Nakamoto consensus underneath; however, miners mine fruits simultaneously as they search for blocks. Fruits contain the transactions, and blocks in the underlying Nakamoto blockchain contain the fruits (but not the transactions). In the Fruitchain protocol, a fruit is regarded as the new block and viewed as part of the blockchain abstraction, but the underlying Nakamoto can be regarded as simply an internal detail of the protocol and need not be exposed to the outside. We will assume that the Fruitchain protocol takes in the following parameters as inputs (see the Fruitchain paper for details [48]):

- Mining difficulty parameters ρ and ρ_f, for mining the block and fruit respectively. Henceforth we shall assume that $\rho_f := \rho$ is hardcoded (although $\rho_f := c\rho$ for any constant $c \geq 1$ should also work), and therefore we do not mention ρ_f explicitly any more.
- The look-back parameter κ, i.e., how far back in the blockchain to hang a fruit from;
- The recency parameter R, that is, a fruit is considered fresh if it is hanging from a $(\mathsf{R} \cdot \kappa)$-recent block in the underlying blockchain. Henceforth we will simply assume that $\mathsf{R} := 17$ is hardcoded (although any other constant great than 1 should also work), and therefore we do not explicitly mention the recency parameter any more henceforth[6].

Theorem 6 (Fruitchain as the underlying snailchain [48]). *For any $0 < \eta < 1$, $\rho > 0$, let $G = (1-5\eta)(1-\alpha)n\rho$, $G' = (1+5\eta)n\rho$, and $Q = (1-5\eta)(1-\alpha)$, the Fruitchain protocol [48] parametrized with (ρ, $\kappa = \frac{\lambda}{34}$) satisfies λ-consistency, $(\lambda/\eta, Q)$-fruit quality, $(\lambda/\eta, G, G')$-fruit growth w.r.t. any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is $\Gamma^{\mathsf{snail}}_{\rho}$-admissible w.r.t.* snailchain.

Corollary 2 (Fruitchain as the underlying snailchain [48]). *For any (arbitrarily small) constant $\epsilon > 0$, let $\alpha = \frac{1}{3} - \epsilon$, there exists a suitable $\kappa = \Theta(\lambda)$ and a constant $0 < \eta < 1$ (related to ϵ); moreover for every $n, \delta > 0$, there exists a sufficiently small $\rho := \Theta(\frac{1}{\delta n})$, such that Fruitchain with parameters (ρ, κ) satisfies λ-consistency, $(\frac{\lambda}{\eta}, Q)$-chain quality and $(\frac{\lambda}{\eta}, G, G')$-chain growth w.r.t. any $\Gamma^{\mathsf{snail}}_{\rho}$-admissible $(\mathcal{A}, \mathcal{Z})$ where*

$$Q > \frac{2}{3}, \quad G = \frac{2}{3}n\rho, \quad G' = (1 + 5\eta)n\rho$$

Or more simply (and informally) put, for every $\alpha = \frac{1}{3} - \epsilon$ where $\epsilon > 0$ is an arbitrarily small constant, there is an appropriately parametrized Fruitchain protocol that achieves $Q > \frac{2}{3}$ chain quality.

[6] We pick $\mathsf{R} = 17$ based on Theorem 3.1 in the Fruitchain paper [48]. However, note that a tighter bound can be proven for any constant $\mathsf{R} > 1$. This could be done by using a tighter version of the fruit freshness lemma in Fruitchain [48].

4.2 Strongly Secure Permissioned Byzantine Fault Tolerance

We will rely on a permissioned consensus protocol. It is well-known how to construct Byzantine Fault Tolerance (BFT) protocols in partially synchronous networks [18, 23, 42]; and furthermore, these protocols achieve responsiveness.

Due to technical subtleties related to a selective opening attack, we need to define a stronger security notion for our BFT building block than the most natural property-based notion. We consider BFT protocols that make blackbox usage of a signing algorithm. Let $\Sigma := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ denote a signature scheme. We use the notation BFT^Σ to mean that the protocol BFT is parametrized by the signature scheme Σ. Moreover, we require that BFT only makes blackbox usage of $\Sigma.\mathsf{Gen}$ and $\Sigma.\mathsf{Sign}$ functionalities — and in our formulation below, BFT nodes query the environment $\mathcal{Z}$ that will provide $\Sigma.\mathsf{Gen}$ and $\Sigma.\mathsf{Sign}$ oracles. Formally, we assume that a BFT^Σ protocol, parametrized by a signature scheme $\Sigma := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, realizes the following abstractions.

Inputs. The environment is allowed to send the following inputs to honest nodes. All other inputs are ignored.

- The environment $\mathcal{Z}$ can send $\texttt{start}(\mathsf{pk}_i, \mathsf{comm})$ once to an honest node i.
- If a $\texttt{start}$ command has been input, the environment $\mathcal{Z}$ can in each time step input a set of transactions TXs to an honest node.
- Answers to $\texttt{sign}(\mathsf{msg})$ queries.

Outputs. Honest nodes output the following terms to $\mathcal{Z}$ over time.

- If a $\texttt{start}$ command has been input, an honest node will in each time step, output to $\mathcal{Z}$ a totally ordered log of transactions log.
- If a $\texttt{start}$ command has been input, an honest node can output to $\mathcal{Z}$ queries of the form $\texttt{sign}(\mathsf{msg})$ where $\mathsf{msg} \in \{0,1\}^*$ denotes a message.

Compliant executions. We consider execution of a BFT protocol in a partially synchronous network with somewhat static corruptions as elaborated below. The environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ must also satisfy certain constraints. Let T_{stamp} be a polynomially-bounded function in λ, n, Q, and δ. A pair of probabilistic polynomial-time algorithms $(\mathcal{A}, \mathcal{Z})$ is said to be $(n, Q, \delta, \tau, T_{\text{stamp}})$-valid w.r.t. BFT iff the following hold:

- $(\mathcal{A}, \mathcal{Z})$ is (n, δ, τ)-valid w.r.t. BFT as per Definition 1.
- *Somewhat static corruption.* All "spawn" and "target corrupt" instructions must be declared before T_{start}, where T_{start} denotes the time when a $\texttt{start}$ command is first input to an honest node by $\mathcal{Z}$.
- *Committee agreement.* If honest node i receives input $\texttt{start}(\mathsf{pk}_i, \mathsf{comm})$ from $\mathcal{Z}$ at time t, and honest node j receives input $\texttt{start}(\mathsf{pk}_j, \mathsf{comm}')$ at time t', it holds that $\mathsf{comm} = \mathsf{comm}'$. Further, if $i \neq j$, then $\mathsf{pk}_i \neq \mathsf{pk}_j$.

- *Close start.* Let T_{start} be the earliest time an honest node receives input $\texttt{start}(\_,\_)$. Then, for any node i honest at time $T_{\text{start}} + \delta$, i must receive input $\texttt{start}(\_,\_)$ by time $T_{\text{start}} + \delta$. Each node receives $\texttt{start}$ at most once when it is honest.

- *Resilience.* At least $\lceil Q \cdot |\mathsf{comm}| \rceil$ number of $\mathsf{pk}_i \in \mathsf{comm}$ must be specified in $\texttt{start}$ commands that are input to nodes that remain honest till T_{stamp}.

- *Signature oracle correctness.* For any $\texttt{start}(\mathsf{pk}_i, \_)$ command input to an honest node, pk_i must be in the range of the valid public keys for the signature scheme Σ.

  Upon any $\texttt{sign}(\mathsf{msg})$ query from an honest node i, $\mathcal{Z}$ returns an answer σ immediately such that $\Sigma.\mathsf{Verify}(\mathsf{pk}_i, \mathsf{msg}, \sigma) = 1$.

Security definitions. Let $T_{\text{stamp}}, T_{\text{bft}}$ be polynomially-bounded functions in λ, n, Q, and δ. Suppose that $(\mathcal{A}, \mathcal{Z})$ is $(n, Q, \delta, \tau, T_{\text{stamp}})$-valid w.r.t. BFT. Then, for any view in the support of $\textsc{exec}[\mathsf{BFT}](\mathcal{A}, \mathcal{Z}, \lambda)$, we say that $\mathsf{secure}^{T_{\text{bft}}}(\mathsf{view}) = 1$ iff the following properties hold.

- *Consistency.* Consistency incorporates the following properties:
  - *Common prefix.* If an honest node i outputs log at any time $t < T_{\text{stamp}}$, and honest node j (same or different) outputs log' at any time $t' < T_{\text{stamp}}$, it holds that either $\mathsf{log} \prec \mathsf{log}'$ or $\mathsf{log}' \prec \mathsf{log}$.
  - *Self-consistency.* Suppose an honest node i outputs log and log' at times t and t' respectively such that $t < t' < T_{\text{stamp}}$, it must hold that $\mathsf{log} \prec \mathsf{log}'$.

- *Liveness.* If $\mathcal{Z}$ inputs TXs to an honest node at time $T_{\text{start}} \le t < T_{\text{stamp}} - T_{\text{bft}}$, then any node that is honest at time $t' = t + T_{\text{bft}}$ will output a log at time t' such that $\mathsf{TXs} \subseteq \mathsf{log}$. T_{bft} is referred to as the liveness parameter.

Definition 5 (Strongly secure BFT protocols)**.** *Let T_{bft} be a positive polynomial in λ, n, Q, and δ. We say that a protocol BFT is strongly secure against $(1-Q)$-corruption with liveness parameter T_{bft} iff for any $n, \delta > 0$, $\tau \ge 0$, any positive polynomial T_{stamp}, for any p.p.t. $\mathcal{A}$ and any polynomial g, there exists a p.p.t. adversary $\mathcal{B}$ and polynomial g', such that for any p.p.t. $\mathcal{Z}$ such that for any $(\mathcal{A}, \mathcal{Z})$ is $(n, Q, \delta, \tau, T_{stamp})$-valid w.r.t. BFT, for any $\lambda \in \mathbb{N}$,*

$$\begin{aligned} &\Pr\left[\mathsf{view} \leftarrow \textsc{exec}[\mathsf{BFT}](\mathcal{A}, \mathcal{Z}, \lambda) : \mathsf{secure}^{T_{\text{bft}}}(\mathsf{view}) \neq 1\right] \ge g(\lambda) \\ \Longrightarrow\ &\Pr\left[\mathsf{view} \leftarrow \textsc{exec}[\mathsf{BFT}](\mathcal{B}, \mathcal{Z}, \lambda) : \mathsf{forgery}(\mathsf{view}) = 1\right] \ge g'(\lambda) \end{aligned}$$

where $\mathsf{forgery}(\mathsf{view}) = 1$, iff in view,

- at some time t the adversary outputs to the environment $\mathcal{Z}$ a forgery pair $(i, \mathsf{msg}, \sigma)$ such that node i is honest at time t;
- by time t the environment $\mathcal{Z}$ has input $\texttt{start}(\mathsf{pk}_i, \_)$ to node i;
- $\Sigma.\mathsf{Verify}(\mathsf{pk}_i, \mathsf{msg}, \sigma) = 1$; and
- by time t, node i has not submitted a query $\texttt{sign}(\mathsf{msg})$ to $\mathcal{Z}$ where the answer was σ.

Definitional subtleties: corruption model for BFT. Our BFT building block must be secure under the "somewhat static" corruption model. We now elaborate the related definitional subtleties. First, any "spawn" or "target corrupt" instructions must be issued before T_{start}, i.e., when the first `start` is input to an honest node — in this sense, the security notion seems "somewhat static". On the other hand, our security notion is stronger than a standard "static" notion of security due to the following: for nodes that are in precorrupt state before T_{start}, there is an opportunity that they will become corrupt during the course of the BFT protocol and before T_{stamp}. Importantly, all of our security properties, including consistency and liveness properties, must hold before T_{stamp} for *any node that has not become corrupt yet.* In comparison, the static notion does not need to extend security guarantees to precorrupt nodes. In fact, it is not hard to show that our "somewhat static" security notion is strictly stronger than a standard "static" notion of security, and it is not difficult to construct a (possibly contrived) BFT protocol that demonstrates this separation.

Definitional subtleties: strong security of BFT. We remark that due to technical subtleties related to an adversarial selective opening attack, we need to define the above stronger notion of security for the BFT subprotocol. Below we compare this notion with the most natural property-based security notion.

First, while most other security definitions for protocols follow a most natural property-based definitional style, the above security notion for the BFT building block is stronger. In particular, if BFT satisfies the above strong security notion, then a natural instantiation, where honest nodes now generate their own signing key pairs and implement their own signing oracles, would satisfy the most natural property-based notion. Of course, to make the description complete, a valid environment $\mathcal{Z}$ in this case would wait to hear each honest BFT node i output a public key pk_i, and then input `start`(comm) to all honest nodes where comm contain sufficiently many honest nodes' public keys.

When honest nodes implement their own key generation and signing oracles, we can group the honest nodes' key generation and signing oracle implementations into the environment $\mathcal{Z}^*$. Then, this specific $\mathcal{Z}^*$ would never disclose honest nodes' respective secret signing keys. Therefore, if there exists some adversary p.p.t. $\mathcal{A}$, such that $\textsc{exec}[\mathsf{BFT}](\mathcal{A}, \mathcal{Z}^*, \lambda)$ fails any of these properties with non-negligible probability, then we can construct an adversary $\mathcal{B}$ such that during the interaction $\textsc{exec}[\mathsf{BFT}](\mathcal{A}, \mathcal{Z}^*, \lambda)$, $\mathcal{B}$ effectively breaks the security of the signature scheme.

Second, we point out that the natural property-based definitions are weaker and not sufficient for our purposes. In particular, later in the HybridConsensus protocol, the environment $\mathcal{Z}$ for BFT can selectively open a set of public keys to include in the `start` command for the BFT protocol. For example, one can easily imagine a somewhat contrived BFT protocol that would be secure under the most natural property-based definition (like all other definitions in this paper), but would be vulnerable to selective opening attacks: imagine that honest nodes disclose their secret signing keys if some predicate over the chosen public keys is satisfied — this predicate can easily be chosen such that it is satisfied with only $\mathsf{negl}(\lambda)$ probability for an honestly generated set of public keys not subject to adversarial selective opening, but satisfied with overwhelming property under adversarial selective opening (e.g., if all public keys end with 1).

Fortunately, it is not hard to see that many known instantiations of permissioned BFT protocols satisfy this strong notion of security, e.g., PBFT [18] with digital signatures.

Remarks about the signing oracle. We note that alternatively, it is possible to partition away the signing oracle into a global signing functionality adopting the GUC approach [14]. In particular, GUC is necessary since the same signature scheme is shared by multiple protocols, the inner BFT protocol, and our outer DailyBFT protocol. If we adopted the GUC approach, our blackbox reduction notion of security might also be simpler since we need not deal with environment having the signing key. On the other hand, using GUC will likely introduce other complexities in terms of notation. The two approaches are essentially equivalent by repartitioning of algorithm boundaries.

Theorem 7 (Castro and Liskov [18], briefly described in Appendix A)**.** *There exists a* BFT *protocol that is strongly secure against* $(1 - Q) < \frac{1}{3}$ *corruption with liveness parameter* $T_{bft} := O(n\delta)$.

To achieve the above, we can modify PBFT's exponential timeout strategy such that nodes double the time-out every n view changes. For completeness, in Appendix A, we briefly describe the PBFT protocol, and we refer the reader to Castro and Liskov [18] for further details and optimizations. Note that later when we use BFT as a subprotocol in hybrid consensus, the number of BFT nodes n will be substituted with $\mathsf{csize} := \lambda$.

5 Formal Scheme: Hybrid Consensus over Nakamoto

5.1 Notational Conventions

Choice of formal framework. We use the well-accepted Universal Composition [13, 14, 17] framework for formalizing and modularly composing protocols. For the presentation of our construction, we will take a modular approach. For each (sub)protocol, we formally describe its abstraction — not by defining an ideal functionality, but using a property-based approach. We then show how to compose these subprotocols to eventually construct our HybridConsensus protocol.

Session identifier conventions. For any protocol prot, if we write prot[sid], then sid (or whatever variable is in square brackets) denotes the session identifier of the protocol instance. If we write prot only without the square brackets, then it means we only care about one specific session of the protocol (although a higher-level protocol can invoke multiple sessions), and therefore we do not denote the session identifier explicitly.

5.2 Daily Offchain Consensus Protocol

For modular protocol composition, we define an intermediate abstraction called a daily offchain consensus protocol, denoted DailyBFT. In DailyBFT, committee members run an offchain BFT instance to decide a daily log, whereas non-members count signatures from committee members.

Overview of DailyBFT. The definition of the DailyBFT intermediate abstraction extends BFT in the following ways:

- *Extends security to committee non-members and late-spawning nodes.* At a definitional level, the DailyBFT definition extends that of BFT to incorporate committee non-members as well. In particular, in the formal definition of DailyBFT below, all security properties must be satisfied not only by committee members, but also by committee non-members as well. Further, while the BFT definition assumes that all nodes are spawned prior to T_{start}, the definition of $\mathsf{DailyBFT}$ allows nodes to be spawned later. Therefore, here our security definitions including consistency and liveness apply to any node (committee member or non-member alike) that spawns early enough, i.e., before the deadline T_{stamp}. These security guarantees do not extend to nodes that spawn too late, since committee members can become corrupt far out in the future, at which point they can sign arbitrary tuples. For exactly this reason, our hybrid consensus protocol, which consumes $\mathsf{DailyBFT}$ as a building block, will need to explicitly handle late spawning to extend the security guarantees to nodes that spawn late.

- *Termination.* $\mathsf{DailyBFT}$ makes explicit a termination procedure which must satisfy two requirements, namely, *termination agreement* and *timely termination.* Specifically, the environment $\mathcal{Z}$ is allowed send a `stop` instruction to nodes. Timely termination requires that the BFT instance terminate quickly after honest nodes receive input `stop`. Termination agreement requires that all honest nodes agree on the same final log upon termination.

- *Signed daily log hashes.* In $\mathsf{DailyBFT}$, committee members output signed daily log hashes which will later be consumed by the hybrid consensus protocol. These signed daily log hashes satisfy *completeness* and *unforgeability.* Completeness says that honest committee members output the correctly signed hash of their daily logs. Unforgeability says that the environment/adversary cannot forge signatures on any other values besides the correct hash.

Formally, suppose that an $\mathsf{DailyBFT}[R]^{\mathcal{D}}$ protocol, with R being the session identifier (also referred to as the day), and parametrized by the distribution $\mathcal{D}$, provides the following abstraction.

Inputs. In each time step, the environment $\mathcal{Z}$ can provide the following types of inputs multiple times: 1) `keygen`; 2) $\texttt{start}(\mathsf{comm})$ where $\mathsf{comm} = \{\mathsf{pk}_i\}_{i\in[m]}$; 3) TXs; and 4) `stop`.

Outputs. Honest nodes output the following type of messages to $\mathcal{Z}$:

- On input `keygen`, honest nodes output $\mathsf{pk} \leftarrow \mathcal{D}$.
- In each time step t, honest nodes output to the environment $\mathcal{Z}$ $\texttt{notdone}(\mathsf{log}^t)$, until in one final step t^*, it outputs $\texttt{done}(\mathsf{log}^{t^*}, \mathsf{recs})$, where recs is either $\emptyset$ or a set of signed tuples vouching for the hash of the final daily log. After outputting $\texttt{done}(\mathsf{log}^{t^*}, \mathsf{recs})$, honest nodes stop outputting in future time steps.

Terminology. Suppose that in a specific view in the support of $\textsc{exec}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda)$, the environment $\mathcal{Z}$ inputs a unique $\texttt{start}(\mathsf{comm})$ command to all honest nodes — later our compliance rule will require that this be the case, then $\mathsf{comm} := \{\mathsf{pk}_i\}_i$ is referred to as the elected committee.

We say that a node i is an honest committee member at time t, if the following holds:

- Before the first $\texttt{start}(\mathsf{comm})$ command was input to any honest node, node i output to $\mathcal{Z}$ a pk that was included in comm.
- Node i remains honest till time t (but could become corrupt later).

Henceforth, if we say "an honest committee member i performs some action or is the receiver of some action at time t in some view", we implicitly mean that node i is an honest committee member at time t, i.e., it remains honest till time t but could be corrupt later.

The earliest time at which an honest committee member receives input `start` is denoted T_{start}. The earliest time at which an honest committee member receives input `stop` is denoted T_{stop}.

We say that an honest node outputs log as a shorthand to mean that it outputs either $\mathtt{done}(\mathsf{log}, \_)$ or $\mathtt{notdone}(\mathsf{log})$.

When an honest node i outputs $\mathtt{done}(\mathsf{log}, \_)$ at some time, we say that log is node i's final daily log.

Compliant executions. We say that a pair $(\mathcal{A}, \mathcal{Z})$ is $(n, Q, \delta, \tau, T_{\text{stamp}}, T_{\text{bft}})$-valid w.r.t. DailyBFT, if $(\mathcal{A}, \mathcal{Z})$ is not only (n, δ, τ)-valid w.r.t. DailyBFT by Definition 1, but the following also holds:

- *Committee agreement.* If honest node i receives input $\mathtt{start}(\mathsf{comm})$ from $\mathcal{Z}$ at time t, and honest node j receives input $\mathtt{start}(\mathsf{comm}')$ from $\mathcal{Z}$ at time t', it holds that $\mathsf{comm} = \mathsf{comm}'$.
- *Close start and stop.* Let T_{start} be the earliest time an honest node receives input $\mathtt{start}(\_)$. Then, for any node i honest at time $T_{\text{start}} + \delta$, i must receive input $\mathtt{start}(\_)$ by time $T_{\text{start}} + \delta$.

  Similarly, let T_{stop} be the earliest time an honest node receives input `stop`. Then, for any node i honest at time $T_{\text{stop}} + \delta$, i must receive input `stop` by time $T_{\text{stop}} + \delta$. For any honest node i that receives `stop` at time t, it must have received `start` at some time $t' < t$.
- *Resilience.* At least $\lceil Q \cdot |\mathsf{comm}| \rceil$ number of $\mathsf{pk}_i \in \mathsf{comm}$ must be output, earlier than the first `start` command input to any honest node, by nodes that remain honest till T_{stamp}.
- *Early enough stop.* $T_{\text{stop}} + T_{\text{bft}} + \delta \leq T_{\text{stamp}}$, where T_{stop} is the time at which the earliest honest committee member receives input `stop`.
- *Temporary static corruption.* For any $\mathsf{pk} \in \mathsf{comm}$, if pk was output by a node that became corrupt before T_{stamp}, then the "target corrupt" instruction must have been issued before T_{start}.

Security definitions. A DailyBFT protocol is said to be secure against $(1 - Q)$-corruption with liveness parameter T_{bft}, if for any $n > 0$, $\delta > 0$, any $\tau \geq 0$, any $T_{\text{stamp}} > 0$, for any $(\mathcal{A}, \mathcal{Z})$ that is $(n, Q, \delta, \tau, T_{\text{stamp}}, T_{\text{bft}})$-valid w.r.t. DailyBFT, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$, except with $\mathsf{negl}(\lambda)$ probability, the following properties hold for $\textsc{exec}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda)$:

- *Timely termination.* Time termination encompasses the following:
  - Any committee member i that is honest at time $T_{\text{stop}} + T_{\text{bft}}$ must have output $\mathtt{done}(\mathsf{log}, \_)$ by time $T_{\text{stop}} + T_{\text{bft}}$.
  - For any node i that is honest at time $t \geq T_{\text{stop}} + T_{\text{bft}} + \delta$, it must have output $\mathtt{done}(\mathsf{log}, \_)$ by time t.

  In both cases, when an honest node outputs $\mathtt{done}(\mathsf{log}, \_)$, we refer to log as the node's *final* daily log.

  Note that since $(\mathcal{A}, \mathcal{Z})$ is $(n, Q, \delta, \tau, T_{\text{stamp}}, T_{\text{bft}})$-valid w.r.t. DailyBFT, timely termination implies the following: any node that spawned before T_{stamp} and remains honest till T_{stamp} must have output $\mathtt{done}(\_, \_)$ by T_{stamp}. In other words, if any node spawned before T_{stamp} and outputs $\mathtt{done}(\_, \_)$ when it is honest, $\mathtt{done}(\_, \_)$ must be output no later than T_{stamp}.
- *Consistency.* Consistency encompasses the following:

– *Self-consistency.* For any node i that spawned before T_{stamp}, and is honest at time t', suppose node i outputs log at time $t \leq t'$ and outputs log' at time t', it holds that $\mathsf{log} \prec \mathsf{log}'$.
– *Termination agreement.* For any node i that spawned before T_{stamp}, and any node j that also spawned before T_{stamp}, suppose node i outputs $\mathtt{done}(\mathsf{log}', \_)$ and node j outputs $\mathtt{done}(\mathsf{log}', \_)$ before they become corrupt, it holds that $\mathsf{log} = \mathsf{log}'$.
– *Common prefix.* For any nodes i, j that spawned before T_{stamp}, suppose that i is honest at time t and outputs log at time t, and j is honest at time t' and outputs log at time t', it holds that either $\mathsf{log} \prec \mathsf{log}'$ or $\mathsf{log}' \prec \mathsf{log}$.

Note that it may seem like common prefix is implied by termination agreement and self-consistency, but keep in mind that common prefix must additionally hold for nodes that never have an opportunity to output $\mathtt{done}(\_, \_)$ before becoming corrupt.

- *Liveness.* Suppose that $\mathcal{Z}$ inputs TXs to an honest committee member at time $T_{\text{start}} \leq t < T_{\text{stop}} - T_{\text{bft}}$. Then, for any honest node i that spawns at time $t_{\text{spawn}} \leq T_{\text{stamp}}$, if i is honest at time $t' \geq t + T_{\text{bft}} + \delta$, then node i must have output log at some time $t^* \leq t'$ such that $\mathsf{TXs} \subseteq \mathsf{log}$.
- *Completeness.* Let comm be the unique set included in $\mathtt{start}$ commands input to honest nodes. For every $\mathsf{pk} \in \mathsf{comm}$ that is output by a node i honest at sometime t and if node i outputs $\mathtt{done}(\mathsf{log}, \mathsf{recs})$ at time t, then it holds that a valid record $\{R, \mathsf{hash}(\mathsf{log})\}_{\mathsf{pk}^{-1}} \in \mathsf{recs}$ where validity is defined by correct signature verification with pk.
- *Unforgeability.* Let $t \leq T_{\text{stamp}}$, and let $\mathsf{pk} \in \mathsf{comm}$ be output by a node i that is honest at time t. Then, if by time t the adversary $\mathcal{A}$ outputs to the environment $\mathcal{Z}$ a valid tuple $\{R, h\}_{\mathsf{pk}^{-1}}$ where R is the current $\mathsf{DailyBFT}$ instance's session identifier, then it must hold that node i has output $\mathtt{done}(\mathsf{log}, \_)$ by t and $h = \mathsf{hash}(\mathsf{log})$.

Construction. We present a construction of the $\mathsf{DailyBFT}$ protocol from BFT in Figure 1. Below is an informal description of the operations of $\mathsf{DailyBFT}$:

- *BFT virtual nodes and selective opening of committee.* A $\mathsf{DailyBFT}$ node outputs fresh public keys to its environment upon a keygen query. Then when it receives a $\mathtt{start}(\mathsf{comm})$ command, if comm contains one or more of its own public keys, then the node is elected as a committee member. In this case, the node will fork a BFT virtual node for each public key in comm that belongs to itself. Here the committee is selectively opened by the environment through the $\mathtt{start}(\mathsf{comm})$ command, later our proof will need to leverage the strong security of BFT.
- *Member and non-member basic operations.* Committee members populate their daily logs relying on the BFT protocol, whereas committee non-members count signatures from committee members to populate their logs.
- *Termination.* Nodes implement a termination procedure as follows: whenever an honest committee member receives a $\mathtt{stop}$ instruction, it inputs a special, signed $\mathtt{stop}$ transaction to each of its BFT virtual node. As soon as the inner BFT instance outputs a log containing $\mathtt{stop}$ transactions signed by at least $\lceil |\mathsf{comm}|/3 \rceil$ distinct committee public keys, the log is finalized and output. All transactions after the first $\lceil |\mathsf{comm}|/3 \rceil$ $\mathtt{stop}$ transactions (with distinct committee public keys) are ignored.

Subprotocol $\mathsf{DailyBFT}[R]$

On input `init`: $\ell := 0$, $\mathsf{log} := \emptyset$, $\mathsf{mykeys} := \emptyset$

On input `keygen`: $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \Sigma.\mathsf{Gen}(1^\lambda)$, add pk to mykeys, output pk

On input `stop` : for each $\mathsf{BFT}^{\mathsf{pk}}$ virtual node forked: input $\mathsf{TXs} := \{\{\mathtt{stop}\}_{\mathsf{pk}^{-1}}\}$ to $\mathsf{BFT}^{\mathsf{pk}}$

On input $\mathtt{start}(\mathsf{comm})$: if $\mathsf{mykeys} \cap \mathsf{comm} \neq \emptyset$: $\mathsf{isMember} = \mathsf{true}$, else $\mathsf{isMember} = \mathsf{false}$

Case $\mathsf{isMember} = \mathsf{true}$

For each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: fork a BFT virtual node, and $\mathsf{BFT}.\mathtt{start}(\mathsf{pk}, \mathsf{comm})$.
// henceforth this BFT virtual node is denoted $\mathsf{BFT}^{\mathsf{pk}}$

Every time some $\mathsf{BFT}^{\mathsf{pk}}$ virtual node outputs $\mathtt{sign}(\mathsf{msg})$: return $\{\mathsf{msg}\}_{\mathsf{pk}^{-1}}$
// implements a signing oracle for BFT virtual nodes

On input TXs: input TXs to each BFT virtual node forked

Let BFT_0 denote the first such BFT virtual node forked

Let $\mathsf{complete}(\mathsf{log}) = \mathsf{true}$ iff log contains `stop` correctly signed by $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm

Every time step t if `start` has been received and `done` has not been output:
Receive output log^* from BFT_0
If $\mathsf{complete}(\mathsf{log}^*)$ then $\mathsf{log}^* :=$ shortest prefix of log^* such that $\mathsf{complete}(\mathsf{log}^*)$
For each $\mathsf{tx} \in \mathsf{log}^* - \mathsf{log}$ that is not a `stop` transaction:
Let $\ell := \ell + 1$, for each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: gossip $\{R, \ell, \mathsf{tx}\}_{\mathsf{pk}^{-1}}$
$\mathsf{log} := \mathsf{log}^*$
If $\mathsf{complete}(\mathsf{log})$: call `Finalize`; else output $\mathtt{notdone}(\mathsf{log})$

`Finalize`:
$\mathsf{recs} := \emptyset$, remove all `stop` transactions from log
For each $\mathsf{pk} \in \mathsf{mykeys} \cap \mathsf{comm}$: let $x := \{R, |\mathsf{log}|\}_{\mathsf{pk}^{-1}}$, $\mathsf{recs} := \mathsf{recs} \cup \{x\}$, gossip x
Output $\mathtt{done}(\mathsf{log}, \mathsf{recs})$, and stop outputting in future time steps.

Case $\mathsf{isMember} = \mathsf{false}$

On receive $\{R, \ell\}_{\mathsf{pk}^{-1}}$ or $\{R, \ell, \mathsf{tx}\}_{\mathsf{pk}^{-1}}$: add message to $\mathsf{history}$ and check the following:

On collect (r, ℓ, tx) and signatures s.t. $r = R$ and $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm signed the tuple correctly:
If $\mathsf{log}[\ell]$ has not been set, let $\mathsf{log}[\ell] := \mathsf{tx}$

On collect (r, ℓ) and signatures s.t. $r = R$ and $\mathsf{th} := \lceil |\mathsf{comm}|/3 \rceil$ distinct pks in comm signed the tuple correctly:
Wait till $\mathsf{log}[: \ell]$ all populated
Output $\mathtt{done}(\mathsf{log}, \emptyset)$, and stop outputting in future time steps.

Each time step until `done` is output:
let $\mathsf{log}' :=$ longest contiguous prefix of log, output $\mathtt{notdone}(\mathsf{log}')$

Figure 1: **Daily offchain consensus protocol.** Since each signing key is reused for both the inner BFT protocol and the outer $\mathsf{DailyBFT}$ protocol, we assume that the signing algorithm tags each message for the inner BFT instance with the prefix "0", and each message for the outer $\mathsf{DailyBFT}$ with the prefix "1" to avoid namespace collision.

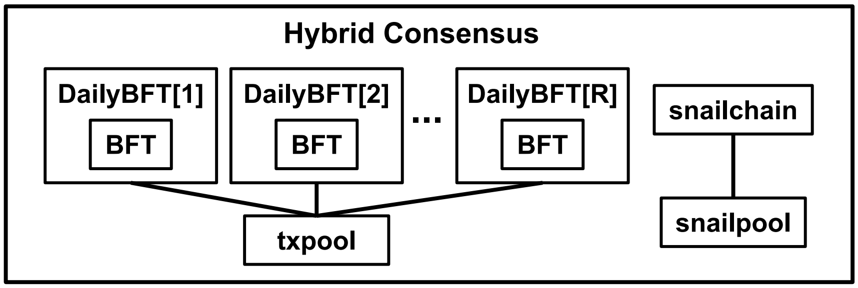

Figure 2: Modular composition of the hybrid consensus protocol.

Subprotocol mempool
On initialize: $\mathsf{TXs} := \emptyset$
On receive TXs_1: $\mathsf{TXs} := \mathsf{TXs} \cup \mathsf{TXs}_1$
On input $\mathtt{propose}(\mathsf{TXs}_1)$: $\mathsf{TXs} := \mathsf{TXs} \cup \mathsf{TXs}_1$, gossip TXs_1
On input $\mathtt{query}(\mathsf{confirmed})$: return $\mathsf{TXs} \backslash \mathsf{confirmed}$

Figure 3: **The mempool subprotocol** keeps track of transactions, and upon query, proposes a set of outstanding transactions. An obvious practical optimization not documented here for simplicity is that the mempool can purge transactions that are already confirmed in LOG.

- *Signed daily log hashes.* When committee members output done, they also output a signed digest of the final daily log — later, our HybridConsensus protocol will stamp this digest onto the snailchain.

Theorem 8 (DailyBFT from BFT)**.** *Suppose that the signature scheme Σ employed by* DailyBFT *is secure, and that* hash *is a random oracle. Suppose that* BFT *is secure against $(1-Q)$-corruption with liveness parameter ${T_{bft}}'$ for $Q > \frac{2}{3}$. Then,* DailyBFT *is secure against $(1-Q)$-corruption with liveness parameter $T_{bft} := {T_{bft}}' + \delta$.*

The proof of this theorem is deferred to Section 8.

5.3 Hybrid Consensus Protocol

We now describe our final product, the hybrid consensus protocol. Hybrid consensus consumes multiple instances of DailyBFT where rotating committees agree on daily logs. Hybrid consensus primarily does the following:

- It manages the spawning and termination of DailyBFT instances effectively using snailchain as a global clock that offers weak synchronization among honest nodes;
- Recall that each DailyBFT instance does not ensure security for nodes that spawn too late, since committee members can become corrupt far out in the future at which point they can sign arbitrary tuples. Therefore, hybrid consensus introduces an on-chain stamping mechanism to extend security guarantees to even nodes that spawn late.

Figure 4 is an algorithmic description of the HybridConsensus protocol. Figure 2 illustrates the modular composition of the hybrid consensus protocol. Specifically, the hybrid consensus protocol

Protocol $\mathsf{HybridConsensus}^{\lambda}$, parametrized by λ

On `init`: $R := 0$, $\mathsf{LOG}_s := \emptyset$, $\mathsf{LOG} := \emptyset$, $\mathsf{csize} := \lambda$.

Mempools:

- Fork an instance of mempool denoted snailpool that stores pending records for snailchain.
- Fork another instance of mempool that stores pending transactions, denoted txpool.
- On input TXs: txpool.`propose`(TXs).

Snailchain: Fork an instance of snailchain, in each time step:

- let chain denote the current local chain
- let $\mathsf{pk} := \mathsf{DailyBFT}[R+1].\mathtt{keygen}$ where R denotes the current day
- let $\mathsf{recs} := \mathsf{snailpool}.\mathtt{query}(\mathsf{ExtractRecs}(\mathsf{chain}[: -\lambda]))$, input $(\mathsf{recs}, \mathsf{pk})$ to snailchain

Preprocess:

Wait till $|\mathsf{chain}| \geq \mathsf{csize} + \lambda$

$L :=$ Find in history maximal, ordered list of (R, log_R) tuples such that R is incrementing with no gaps, and $(R, \mathsf{hash}(\mathsf{log}_R))$ is on-chain valid w.r.t. chain

$\mathsf{LOG} := \mathsf{log}_1 || \mathsf{log}_2 || \ldots || \mathsf{log}_{|L|}$

Daily Offchain Consensus:

Let $R := |L|$, fork an instance of $\mathsf{DailyBFT}[R+1]$:

Loop:

Wait till $|\mathsf{chain}| \geq \mathsf{upper}(R+1) + \lambda$, let $R := R+1$

Let $\mathsf{comm}_R :=$ MinersOf$(\mathsf{chain}[\mathsf{lower}(R) : \mathsf{upper}(R)])$ where MinersOf parses each block as $(\mathsf{recs}_i, \mathsf{pk}_i)$ and returns a list containing all pk_is

If an instance $\mathsf{DailyBFT}[R-1]$ exists, $\mathsf{DailyBFT}[R-1].\mathtt{stop}$

Fork an instance of $\mathsf{DailyBFT}[R+1]$

$\mathsf{DailyBFT}[R].\mathtt{start}(\mathsf{comm}_R)$

Each time step: let $\mathsf{TXs} := \mathsf{txpool}.\mathtt{query}(\mathsf{LOG})$, input TXs to $\mathsf{DailyBFT}[R]$

Output: In each time step: let R denote the current day. Let $\mathsf{isdone}(r) = \mathsf{true}$ if $\mathsf{DailyBFT}[r]$ has output `done` in this or earlier time steps.

- If $\mathsf{DailyBFT}[R-1]$ outputs $\mathtt{done}(\mathsf{log}_{-1}, \mathsf{recs})$ in this time step: $\mathsf{LOG}_s := \mathsf{LOG}_s || \mathsf{log}_{-1}$, $\mathsf{snailpool}.\mathtt{propose}(\mathsf{recs})$
- Let log_{-1} and log be the output logs of $\mathsf{DailyBFT}[R-1]$ and $\mathsf{DailyBFT}[R]$ in this time step respectively (or $\emptyset$ if nothing is output)
- If $\mathsf{isdone}(R-1)$: $\mathsf{LOG} := \mathsf{LOG}_s || \mathsf{log}$; else $\mathsf{LOG} := \mathsf{LOG}_s || \mathsf{log}_{-1}$. Output LOG

On-chain valid: A tuple (R, h) is on-chain valid w.r.t chain iff the following holds

- For at least $\mathsf{th} := \lceil \mathsf{csize}/3 \rceil$ distinct $\mathsf{pk} \in$ MinersOf$(\mathsf{chain}[\mathsf{lower}(R), \mathsf{upper}(R)])$: $\{R, h\}_{\mathsf{pk}^{-1}}$ is the first occurrence in $\mathsf{chain}[: -\lambda]$ where pk signed some tuple of the form $(R, \_)$.

Figure 4: **Main** HybridConsensus **protocol.** A newly spawned, honest node starts running this protocol. We assume history is the set of all historical transcripts sent and received. We assume that message routing to subprotocol instances is implicit: whenever any subprot[sid] instance is forked, history[subprot[sid]] and protocol messages pertaining to subprot[sid] are automatically routed to the subprot[sid] instance.

internally runs the following subprotocol instances: two mempool instances denoted snailpool and txpool respectively, a snailchain instance, and multiple DailyBFT instances. We now explain these subprotocol instances more concretely.

Transaction mempools. The HybridConsensus protocol maintains two instances of the mempools protocol (see Figure 3), denoted txpool and snailpool respectively. The txpool is a mempool used for maintaining outstanding transactions to be confirmed with DailyBFT instances, and the snailpool is used for maintaining daily log digests to be stamped on the snailchain. The mempool protocol is very simple: it gossips transactions over the network whenever the environment inputs new transactions. Whenever it hears transactions from the network, it saves them in the mempool.

snailchain. The HybridConsensus protocol internally forks a snailchain instance. First, the snailchain is used for reaching agreement on committees who will then run the offchain BFT consensus. The committee is selected as the miners of $\mathsf{csize} := \lambda$ consecutive blocks. The chain quality property of the underlying snailchain ensures that sufficiently many of these miners are honest for sufficiently long. Second, this snailchain instance is used not for committing transactions, but for stamping daily log digests such that the protocol can resist retroactive corruptions where the adversary corrupts committee members in the future.

DailyBFT instances. The HybridConsensus protocol forks multiple instances of the DailyBFT protocol, and we use the index R to denote the session identifier of each instance. R is also referred to as the day number, and hence each $\mathsf{DailyBFT}[R]$ instance outputs a "daily log". In each $\mathsf{DailyBFT}[R]$ instance, the elected committee members rely on the underlying BFT protocol to commit transactions and output a daily log over time, whereas committee non-members count signatures from committee members to populate their daily logs.

Operations. Each node maintains a history of all past transcripts denoted history — we assume this for simplicity of formalism, and it can be optimized away in practice. Nodes that newly spawn obtain the historical transcripts instantly (in practice this can be instantiated by having honest nodes offer a history retrieval service).

When a new node spawns, it populates its LOG as follows:

- *Matching on-chain valid tuples.* A newly spawned node first identifies all on-chain valid tuples of the form (R, h), where R is the day number and h is the hash of the daily log. Then, the node will search history and identify an appropriate daily log log_R that is consistent with h. The node populates LOG with these daily logs. This on-chain matching process effectively provides a safe mechanism for a newly spawned node to catch up and populate old entries of its output LOG.
- *Through daily offchain consensus.* Once this catch-up process is complete, the node will henceforth rely on DailyBFT instances to further populate remaining entries of its output LOG. In each DailyBFT instance, a node can act as a committee member of a committee non-member.

  To do this, a node monitors its output chain from the snailchain instance. As soon as the chain length exceeds $\mathsf{csize} \cdot R + \lambda$, the R-th day starts, at which point the node inputs `stop` to the previous $\mathsf{DailyBFT}[R-1]$ instance (if one exists), and inputs $\texttt{start}(\text{MinersOf}(\mathsf{chain}[\mathsf{lower}(R) : \mathsf{upper}(R)]))$ to the $\mathsf{DailyBFT}[R]$ instance. There is typically a period of overlap during which both $\mathsf{DailyBFT}[R-1]$ and $\mathsf{DailyBFT}[R]$ instances are running simultaneously and outputting their respective daily logs. When nodes assimilate their daily logs into the final output LOG, they make sure that LOG is always contiguous leaving no gaps in between. Due to the timely termination property of $\mathsf{DailyBFT}$, the old $\mathsf{DailyBFT}[R-1]$ will terminate fairly soon at which point the new $\mathsf{DailyBFT}[R]$ instance fully takes over.

5.4 Theorem Statements

Definition 6 (Admissible parameters for hybrid consensus $\Gamma_\rho^{\mathsf{hc}}$)**.** *Let $T_{bft} := O(m\delta)$ be the liveness parameter for* BFT *with m nodes. We define $\Gamma_\rho^{\mathsf{hc}}(n, \alpha, \delta, \tau) = 1$ iff the following holds:*

- $n > 0$, $\delta > 0$, $\tau \geq 0$ are all polynomial functions in λ; $\alpha > 0$ is a constant;
- There exists a constant $\eta > 0$ such that $p(1 - (2\delta + 2)p) \geq (1 + \eta)q$. (This is needed for the underlying snailchain to be secure.)
- There exists a constant $\eta_0 > 0$ such that $Q := 1 - (1 + \eta_0)\frac{q}{\gamma} > \frac{2}{3}$. (This is needed such that we get $> 2/3$ chain quality for snailchain.)
- There exists a constant $\eta_1 > 0$ such that $G' := (1 + \eta_1)n\rho < \frac{\lambda}{T_{\mathrm{bft}}+\delta}$ for any $\lambda \in \mathbb{N}$. (This is needed such that the chain does not grow too fast to ensure liveness.)
- There exists a constant $\eta_2 > 0$ such that $\tau > 4\lambda(1 + \eta_2)/\gamma + c\lambda\delta$ for some appropriately large constant c. (This is needed such that the adversary is sufficiently constrained in agility.)

In the above, parameters p, q, γ are functions in n, α, δ as defined in Section 4.1.

Theorem 9 (Main theorem for HybridConsensus)**.** *Suppose that* $\mathsf{hash}, \mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *are independent random oracles, and that the signature scheme Σ is secure. Then, for any constant $\eta > 0$, hybrid consensus instantiated with Nakamoto as* snailchain *and with mining difficulty parameter ρ is secure with liveness parameters $(T_{warmup}, T_{confirm})$ w.r.t. any p.p.t. $\Gamma_\rho^{\mathsf{hc}}(n, \alpha, \delta, \tau)$-admissible $(\mathcal{A}, \mathcal{Z})$, where*

$$T_{warmup} := 2\lambda(1 + \eta)/\gamma, \quad T_{confirm} := O(\lambda\delta)$$

Note that the above T_{confirm} parameter is for the worst-case, in the optimistic case, hybrid consensus achieves a transaction confirmation time of $O(\delta)$.

The proofs of the above theorem will be presented in Section 8.

Typical parametrizations. Typically in practice, if we set the puzzle's difficulty parameter $\rho := \Theta(\frac{1}{n\Delta})$ to be sufficiently small, where Δ is a possibly loose upper bound on the network's delay known a priori. Under such parametrization, if the overall corruption α is roughly 3/4, then we ensure roughly 2/3 chain quality.

Corollary 3 (Typical parameters for hybrid consensus over Nakamoto: restatement of Theorem 3)**.** *Assume that $\alpha = \frac{1}{4} - \epsilon$ for an arbitrary constant $\epsilon > 0$. Then for every n, δ, there exists sufficiently small $\rho_0 := \Theta(\frac{1}{\delta n})$ such that hybrid consensus with Nakamoto as the underlying* snailchain *and with mining difficulty parameter $\rho < \rho_0$ is secure w.r.t. any p.p.t. $\Gamma_\rho^{\mathsf{hc}}(n, \alpha, \delta, \tau)$-admissible $(\mathcal{A}, \mathcal{Z})$, where*

$$T_{warmup} := 8\lambda/3n\rho, \quad T_{confirm} := O(\lambda\delta)$$

5.5 Practical Considerations and Possible Optimizations

The scheme described above is optimized for simplicity and to aid formal analysis, but not for practical performance. There are many possible optimizations. For example, instead of having committee members sign each transaction one by one, they could instead sign a batch of transactions at a time. During stamping, instead of having each node send a separate signature onto snailchain, we can rely on a threshold signature scheme and stamp a single signature of the daily log's hash onto snailchain. It is also easy to prune storage of old transcripts. We leave practical optimizations and implementation to future work.

6 Extension: Fruitchain as the Underlying snailchain

Pass and Shi recently propose a new blockchain protocol called Fruitchain [48]. For a sufficiently small puzzle difficulty parameter $\rho := \Theta(\frac{1}{n\delta})$, and $\alpha := \frac{1}{3} - \epsilon$ for an arbitrarily small constant $\epsilon > 0$, Fruitchain achieves $Q > \frac{2}{3}$ chain quality over any sufficiently large window of consecutive blocks. It is not too hard to compose hybrid consensus and Fruitchains to obtain a responsive permissionless consensus protocol that is resilient against $1/3-\epsilon$ overall corruption for an arbitrarily small constant $\epsilon > 0$.

Fruitchain provides the same formal abstraction as Nakamoto consensus, but with different parameters. Henceforth we will use the term "fruit quality" to mean Fruitchain's chain quality, and "fruit growth" to mean Fruitchain's chain growth. Under typical parameters stated in Section 4.1.2 and Corollary 2, to obtain η-optimal chain quality, Fruitchain requires the fruit quality window to be reasonably large, i.e., $\Theta(\lambda/\eta)$. Similarly, fruit growth also requires the time window to be reasonably long. In light of this, when we adopt Fruitchain as the underlying snailchain, we need to make the following changes to the protocol described in Section 5:

- Let $\mathsf{HybridConsensus}^{\lambda,\eta}$ be parametrized with parameters λ, and η.
- Redefine $\mathsf{csize} := \frac{\lambda}{\eta}$, $\mathsf{lower}(R) := \frac{(R-1)\lambda}{\eta}+1$, and $\mathsf{upper}(R) := \frac{R\cdot\lambda}{\eta}$. In other words, the committee size is set to $\frac{\lambda}{\eta}$, and the protocol waits for the chain length $|\mathsf{chain}| \geq \mathsf{upper}(R) + \lambda$ to start the R-th day.

Definition 7 (Admissible parameters for hybrid consensus over Fruitchain $\Gamma^{\mathsf{hcfruit}}_{\rho,\eta}$)**.** *Let $T_{bft} := O(m\delta)$ be the liveness parameter for* BFT *with m nodes. We define $\Gamma^{\mathsf{hcfruit}}_{\rho,\eta}(n, \alpha, \delta, \tau) = 1$ iff the following holds:*

- $n > 0$, $\delta > 0$, $\tau \geq 0$ are all polynomial functions in λ; $\alpha > 0$ is a constant;
- There exists a constant $\eta' > 0$ such that $p(1 - (2\delta + 2)p) \geq (1 + \eta')q$. (This is needed for the underlying snailchain to be secure.)
- $Q := (1 - 5\eta)(1 - \alpha) > \frac{2}{3}$. (This is needed such that we get $> 2/3$ chain quality for snailchain.)
- $G' := (1 + 5\eta)n\rho < \frac{\lambda}{\eta(T_{\mathrm{bft}}+\delta)}$ for any $\lambda \in \mathbb{N}$. (This is needed such that the chain does not grow too fast to ensure liveness.)
- $\tau > 3\lambda(1+\frac{1}{\eta})/((1-5\eta)(1-\alpha)n\rho) + c\lambda\delta$ for some appropriately large constant c. (This is needed such that the adversary is sufficiently constrained in agility.)

In the above, parameters p, q, γ are functions in n, α, δ as defined in Section 4.1.

Theorem 10 (Hybrid consensus over Fruitchain: restatement of Theorem 4)**.** *For any (arbitrarily small) constant $\epsilon > 0$, let $\alpha = \frac{1}{3} - \epsilon$, and for every n, δ, there exists a sufficiently small $\rho := \Theta(\frac{1}{\delta n})$, a suitable $\kappa = \Theta(\lambda)$, and constant $\eta > 0$, such that* $\mathsf{HybridConsensus}^{\lambda,\eta}$ *over Fruitchain with parameters (ρ, κ), is secure w.r.t. any p.p.t. $\Gamma^{\mathsf{hcfruit}}_{\rho,\eta}(n, \alpha, \delta, \tau)$-admissible $(\mathcal{A}, \mathcal{Z})$, where*

$$T_{warmup} := 1.5\lambda(1 + \frac{1}{\eta})/(1 - 5\eta)n\rho, \quad T_{confirm} := O(\lambda\delta)$$

Again, the above $T_{\text{confirm}} = O(\lambda\delta)$ is the worst-case transaction confirmation time (i.e., even when under attack). The optimistic transaction confirmation time is $O(\delta)$, i.e., independent of the security parameter λ.

7 Proof Roadmap

Before presenting the detailed proofs, we first describe a high-level roadmap to aid understanding. For simplicity, we use hybrid consensus over Nakamoto as an example in our description, since the proof for hybrid consensus over Fruitchain is the same except with different parameters.

HybridConsensus from DailyBFT. Our proof will proceed in the following steps.

1. First, primarily in Lemma 2, Lemma 3, and Fact 4, we prove that when executing as a subprotocol of HybridConsensus, all instances of DailyBFT have a valid environment with overwhelming probability. Once we show this, we can henceforth rely on the the security properties of DailyBFT in the remainder of the proof.

2. Next, we prove a pair of lemmas that establishes $T_{\text{stamp}}(R)$ as a "deadline" for each day R. All honest committee members' actions will have completed and taken effect by time $T_{\text{stamp}}(R)$.

   Lemma 4 says, roughly, that all honest committee members for day R will have stamped a correct signed daily hash to snailchain by time $T_{\text{stamp}}(R)$. Fact 5 says, roughly, that nodes which spawn later than $T_{\text{stamp}}(R)$ will not create a DailyBFT(R) instance, but will rely on on-chain stamped daily log hashes to decide the R-th day's daily log.

3. Having established $T_{\text{stamp}}(R)$ as a deadline for day R, we then prove consistency using the following strategy (Lemma 5 and Theorem 11):

   - For nodes that actually created a DailyBFT(R) instance, we know that they must have spawned before $T_{\text{stamp}}(R)$. We therefore rely on properties of DailyBFT(R) to prove consistency for such nodes (committee member or non-member alike).
   - For nodes that did not spawn a DailyBFT(R) instance, we show that they would satisfy consistency too if they recovered their daily log by examining what is stamped on the snailchain. Intuitively, committee members always stamp the correctly signed daily log hash on snailchain before they ever become corrupt. Therefore, even if they become corrupt later and can henceforth stamp arbitrary things onto snailchain, it will be too late since honest node only recognize the first stamped daily log hash for committee member's public key.

4. Finally, we prove the liveness of hybrid consensus (Theorem 12) roughly as follows. Informally, suppose the environment inputs tx to an honest node during day R (roughly speaking). There are two cases: 1) tx is proposed early enough in day R, such that the liveness property of the DailyBFT[R] instance applies; and 2) tx is proposed too late to be incorporated in day R's log since tx is closed very close to the end of day. In this case, tx will be rolled over to day $R+1$. Intuitively since the DailyBFT[R] instance will terminate quickly, and by the liveness property of DailyBFT[$R+1$], it also will not take too long before tx is incorporated into the log of day $R+1$.

   The above, however, only applies to nodes who actually spawned a DailyBFT[R] (or DailyBFT[$R+1$]) instance. For any node that joins too late and did not spawn a DailyBFT[R] (or DailyBFT[$R+1$]), it will output some daily log for day R the moment they spawn by processing historical transcripts. Now, by the consistency property, we know that whatever daily log the late node outputs, it will contain tx as well.

DailyBFT from BFT. The most technical part of this proof involves proving the following. Observe that when BFT is run as a subprotocol inside DailyBFT, the environment perceived by BFT is partially specified by the DailyBFT protocol. Recall that the environment for BFT needs to implement a signing oracle for BFT. When run inside DailyBFT as a subprotocol, the signing oracle is implemented by the DailyBFT protocol. By definition of the honest DailyBFT protocol, honest nodes never disclose their signature secret keys. For such an environment (of BFT), if the security properties specified in Section 4.2 can be broken by a p.p.t. adversary $\mathcal{A}$, we can construct a reduction Re that breaks signature security.

The above essentially allows us to prove that when BFT is run inside DailyBFT as a subprotocol instance, the environment for BFT is nice such that all of the stated security properties for BFT will hold except with negligible probability. The remainder of this proof henceforth relies on these properties of BFT to make arguments.

8 Detailed Proofs

Below we present our proofs for hybrid consensus over Nakamoto, and the proof for hybrid consensus over Fruitchain is the same except with altered parameters.

8.1 Terminology and Simple Facts

Environment for a subprotocol. In an execution of the protocol HybridConsensus with $\mathcal{A}, \mathcal{Z}$, let $(\mathcal{A}, \mathcal{Z})$[subprot[sid]] be the adversary/environment pair that subprotocol instance subprot[sid] interfaces with. $(\mathcal{A}, \mathcal{Z})$[subprot[sid]] is defined by $(\mathcal{A}, \mathcal{Z})$ and the part of HybridConsensus outer to subprot[sid]. We also use the notation $\mathcal{Z}$[subprot[sid]] to denote the environment that subprotocol instance subprot[sid] interfaces with. Recall that Figure 2 in Section 5.3 illustrates the modular composition of our HybridConsensus protocol.

The following fact says that if DailyBFT's environment inputs some start(comm) where comm contains a pk output by a node i that is honest at time $t \geq T_{\text{start}}$, then node i must have output pk before T_{start}. In other words, DailyBFT's environment cannot predict future pk pairs output by honest nodes. This simple fact is handy throughout, since whenever we say that some pk $\in$ comm is output by an honest node i, this honest node is implicitly a committee member. Recall that by definition, for a node i to be considered a committee member, it must have output some $\mathsf{pk} \in \mathsf{comm}$ prior to T_{start}.

Fact 1 (Unpredictability of public keys.). *Assume that the signature scheme is secure, it must hold that for any p.p.t.* $(\mathcal{A}, \mathcal{Z})$ *and any* $\lambda \in \mathbb{N}$*, the following property holds for* $\text{EXEC}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda)$*, with* $1 - \mathsf{negl}(\lambda)$ *probability:*

If the $\mathcal{Z}$ *inputs to any honest node* $\texttt{start}(\mathsf{comm})$*, and let* i *be a node that is honest at time* $t \geq T_{start}$ *and moreover has output* $\mathsf{pk} \in \mathsf{comm}$*, then it holds that node* i *is an honest committee member w.r.t.* $\mathsf{DailyBFT}$ *at time* t*, i.e., the aforementioned* $\mathsf{pk} \in \mathsf{comm}$ *must be output by* i *before* T_{start}*.*

Proof. Straightforward by the fact that a secure signature scheme must have high-entropy public keys. □

Fact 2. *For a secure signature scheme, polynomially many honestly generated public keys will not collide with* $1 - \mathsf{negl}(\lambda)$ *probability.*

Therefore henceforth we simply assume that signature public keys generated by honest nodes do not collide.

8.2 Hybrid Consensus Proofs

Times for notable events. Given Lemma 2, we know that for every polynomially bounded $R \in N$, $(\mathcal{A}, \mathcal{Z})[\mathsf{DailyBFT}[R]]$ respects committee agreement. Therefore the notion of an honest committee member is well-defined for protocol instance $\mathsf{DailyBFT}[R]$. For convenience, we explicitly define the following times for important events.

- $T_{\text{start}}(R)$: earliest time that an honest committee member inputs `start` to its $\mathsf{DailyBFT}[R]$ instance;
- $T_{\text{stop}}(R)$: earliest time that an honest committee member inputs `stop` to its $\mathsf{DailyBFT}[R]$ instance.
- $T_{\text{stamp}}(R) := T_{\text{start}}(R) + \lambda/G + T_{\text{bft}} + \delta + T_{\text{snail}}$.

Lemma 2 ($\mathsf{DailyBFT}$'s environment satisfies committee agreement, close start and stop, and temporary static corruption). *Let* $(\mathcal{A}, \mathcal{Z})$ *be* $(n, \alpha, \delta, \tau)$*-valid w.r.t.* $\mathsf{HybridConsensus}$ *for any* $n \in \mathbb{N}, \alpha > 0, \delta > 0$*, and* $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ *such that* $\mathsf{snailchain}$ *satisfies consistency,* Q*-chain quality, and* (G, G')*-chain growth w.r.t.* $(\mathcal{A}, \mathcal{Z})[\mathsf{snailchain}]$*. Then, for any* $\lambda \in \mathbb{N}$*, the following properties hold for* $\text{EXEC}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ *with* $1 - \mathsf{negl}(\lambda)$ *probability:*

For any $R = \mathsf{poly}(\lambda) \in \mathbb{N}$*,* $(\mathcal{A}, \mathcal{Z})[\mathsf{DailyBFT}[R]]$ *satisfies committee agreement, close start and stop, and temporary static corruption.*

Proof. Committee agreement follows in a straightforward manner from the definition of $\mathsf{HybridConsensus}$ and from the consistency property of $\mathsf{snailchain}$.

Close start and stop follows in a straightforward manner from the definition of $\mathsf{HybridConsensus}$ and from the consistent length property of $\mathsf{snailchain}$ (which is part of chain growth).

Temporary static corruption follows in a straightforward manner from the definition of $T_{\text{stamp}}(R)$ and the underlying τ-agile corruption model. □

Fact 3 (Bounded day length)**.** *Let* $(\mathcal{A}, \mathcal{Z})$ *be* $(n, \alpha, \delta, \tau)$*-valid w.r.t.* HybridConsensus *for any* $n \in \mathbb{N}, \alpha > 0, \delta > 0$*, and* $\tau \geq 0$ *such that* snailchain *satisfies consistency,* Q*-chain quality, and* (G, G')*-chain growth w.r.t.* $(\mathcal{A}, \mathcal{Z})$[snailchain]*. Then, for any* $\lambda \in \mathbb{N}$*, the following holds for* EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ *with* $1 - \mathsf{negl}(\lambda)$ *probability:*

$$2\lambda/G' \leq T_{start}(1) \leq 2\lambda/G,$$
$$\forall R = \mathsf{poly}(\lambda) \in \mathbb{N} : T_{start}(R) + \lambda/G' \leq T_{stop}(R) = T_{start}(R+1) \leq T_{start}(R) + \lambda/G$$

Proof. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

The fact that $T_{\text{stop}}(R) = T_{\text{start}}(R + 1)$ follows in a straightforward manner from the definition of the honest HybridConsensus protocol. Further, by the definition of the honest HybridConsensus protocol, honest nodes send `stop` to DailyBFT$[R-1]$ when their local chain length reaches csize$R+\lambda$. By (G, G')-chain growth, it follows that $T_{\text{start}}(R) + \lambda/G' \leq T_{\text{stop}}(R) = T_{\text{start}}(R + 1) \leq T_{\text{start}}(R) + \mathsf{csize}/G = T_{\text{start}}(R) + \lambda/G$. Similarly, by (G, G')-chain growth, it holds that $2\lambda/G' \leq T_{\text{start}}(1) \leq 2\lambda/G$. ☐

Henceforth, we will assume that EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ will assert the bad events declared in Fact 1, Lemma 2, and Fact 3. If such bad events ever happen, the execution aborts. Since all these bad events occur with $\mathsf{negl}(\lambda)$ probability, the new execution with the bad events asserted is computationally indistinguishable to an $(\mathcal{A}, \mathcal{Z})$ pair that satisfies the conditions specified in Fact 1, Lemma 2, and Fact 3.

In particular, since we assume committee agreement, henceforth we will use the notation comm_R to denote the globally agreed upon committee for the R-th day in any specific view in the support of EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$.

Lemma 3 (Sufficiently many comm_R members remain honest till $T_{\text{stamp}}(R)$)**.** *Let* Q, G, G' *be polynomially-bounded functions in* $\lambda, n, \alpha, \delta$*. For any* $T_{bft} > 0$*, and any constant* $\eta > 0$*, let* $T_{snail} := (1 + \eta)\lambda/G$*, let* $(\mathcal{A}, \mathcal{Z})$ *be* $(n, \alpha, \delta, \tau)$*-valid w.r.t.* HybridConsensus *for any* $n \in \mathbb{N}, \alpha > 0, \delta > 0$*, and* $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ *such that* snailchain *satisfies consistency,* Q*-chain quality, and* (G, G')*-chain growth w.r.t.* $(\mathcal{A}, \mathcal{Z})$[snailchain]*. Then, for any* $\lambda \in \mathbb{N}$*, the following property holds for* EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ *with* $1 - \mathsf{negl}(\lambda)$ *probability:*

For any $R = \mathsf{poly}(\lambda) \in \mathbb{N}$*, at least* Q *fraction of* comm_R *are output by nodes that remain honest till*

$$T_{stamp}(R) := T_{start}(R) + \lambda/G + T_{bft} + \delta + T_{snail},$$

Proof. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

Let chain be an honest node's local chain at any time s.t. $|\mathsf{chain}| \geq \mathsf{upper}(R)$. By Q-chain quality and since $\mathsf{csize} = \lambda$, at least Q-fraction of chain[lower(R), upper(R)] are intact blocks w.r.t. chain[: lower$(R) - 1$]. This means that for at least Q fraction of indices $i \in$ [lower(R), upper(R)], there exists a node j that was intact at some earlier time t, such that it output to $\mathcal{Z}$[snailchain] chain′ that contains the prefix chain[: lower$(R) - 1$] at time $t - 1$, and $\mathcal{Z}$[snailchain] provided input chain$[i]$:= (recs, pk) to node j at time t. By definition of the HybridConsensus protocol, $\mathsf{pk} \in \mathsf{comm}_R$

and pk must be an output of some $\mathsf{DailyBFT}[R]$ instance of node j at time t. Now due to (G, G')-chain growth, at most $(\mathsf{csize} + \lambda)/G = 2\lambda/G$ time elapsed between t and $T_{\text{start}}(R)$. Finally, due to $\tau > (3\lambda/G + T_{\text{bft}} + \delta + T_{\text{snail}})$-agility, we have that node j remains honest till $T_{\text{stamp}}(R)$. Therefore, at least Q fraction of comm_R are output by nodes that remain honest till $T_{\text{stamp}}(R)$. □

Next, we show that $T_{\text{stamp}}(R)$ is chosen sufficiently far out to give enough time for all comm_R members' DailyBFT instances to output a final daily log (possibly empty) that is consistent with each other.

Fact 4 (DailyBFT's environment is valid.). *Let Q, G, G' be polynomially-bounded functions in $\lambda, n, \alpha, \delta$. For any $T_{bft} > 0$, and any constant $\eta > 0$, let $T_{snail} := (1+\eta)\lambda/G$, let $(\mathcal{A}, \mathcal{Z})$ be $(n, \alpha, \delta, \tau)$-valid w.r.t.* HybridConsensus *for any $n \in \mathbb{N}, \alpha > 0, \delta > 0$, and $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ such that* snailchain *satisfies consistency, Q-chain quality, and (G, G')-chain growth w.r.t. $(\mathcal{A}, \mathcal{Z})[\mathsf{snailchain}]$.*

Then, for any $\lambda \in \mathbb{N}$, with $1-\mathsf{negl}(\lambda)$ probability, the following holds for $\text{EXEC}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$: $(\mathcal{A}, \mathcal{Z})[\mathsf{DailyBFT}[R]]$ is $(n, Q, \delta, \tau, T_{stamp}(R), T_{bft})$-valid w.r.t. DailyBFT*.*

Proof. Straightforward by combining Lemmas 2, 3 and Fact 3. □

Lemma 4 (Timely stamping). *Let G, G', Q be polynomially-bounded functions in $\lambda, n, \alpha, \delta$, Suppose that* DailyBFT *is secure against $(1-Q)$-corruption with liveness parameter T_{bft}. For any constant $\eta > 0$, let $T_{snail} := (1+\eta)\lambda/G$. Let $(\mathcal{A}, \mathcal{Z})$ be $(n, \alpha, \delta, \tau)$-valid w.r.t.* HybridConsensus *for some $n \in \mathbb{N}, \alpha > 0, \delta > 0$, and $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ such that* snailchain *satisfies consistency, Q-chain quality, and (G, G')-chain growth except w.r.t. $(\mathcal{A}, \mathcal{Z})[\mathsf{snailchain}]$. Then, for any $\lambda \in \mathbb{N}$, with $1 - \mathsf{negl}(\lambda)$ probability, the following property holds for $\text{EXEC}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ and for any $R = \mathsf{poly}(\lambda) \in \mathbb{N}$.*

Suppose that any honest node outputs a chain *at time $t \geq T_{stamp}(R)$. For any $\mathsf{pk} \in \mathsf{comm}_R$ output by some node i that is honest at $T_{stamp}(R)$, let $h := \mathsf{hash}(\mathsf{log}_R)$ where log_R represents the final log output by node i's $\mathsf{DailyBFT}[R]$ instance, then, a valid record of the form $\{R, h\}_{\mathsf{pk}^{-1}}$ (where validity is defined by signature verification) is included in $\mathsf{chain}[: -\lambda]$, and it is not preceeded by any other valid record of the form $\{R, h'\}_{\mathsf{pk}^{-1}}$ for a different $h' \neq h$.*

Proof. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for $\text{EXEC}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

Due to Fact 4, $(\mathcal{A}, \mathcal{Z})[\mathsf{DailyBFT}[R]]$ is $(n, \alpha, \delta, \tau, Q, T_{\text{stamp}}(R), T_{\text{bft}})$-valid w.r.t. DailyBFT. By Fact 1, for such a $\mathsf{pk} \in \mathsf{comm}_R$ output by node i that is honest at time $T_{\text{stamp}}(R)$, node i is an honest committee member w.r.t. $\mathsf{DailyBFT}[R]$. Therefore, by timely termination of DailyBFT, node i's $\mathsf{DailyBFT}[R]$ instance will have output some tuple $\mathtt{done}(\mathsf{log}, \mathsf{recs})$ by time $T_{\text{stop}}(R) + T_{\text{bft}}$. By definition of honest HybridConsensus algorithm, node i will have called $\mathsf{snailpool}.\mathtt{propose}(\mathsf{recs})$ by $T_{\text{stop}}(R) + T_{\text{bft}}$; and by definition of the mempool protocol, every honest node's snailpool.TXs will contain recs by $T_{\text{stop}}(R) + T_{\text{bft}} + \delta$. Therefore, starting at time $T_{\text{stop}}(R) + T_{\text{bft}} + \delta$, in every time step, $\mathcal{Z}[\mathsf{snailchain}]$ will include recs in its input to snailchain for every honest node whose local $\mathsf{chain}[: -\lambda]$ does not yet contain rec. By the liveness property of snailchain, any time after $T_{\text{stop}}(R) + T_{\text{bft}} + \delta + T_{\text{snail}}$, recs will appear in every honest node's local $\mathsf{chain}[: -\lambda]$. By the completeness property of the DailyBFT protocol, $\{R, h\}_{\mathsf{pk}^{-1}} \in \mathsf{recs}$, where $h := \mathsf{hash}(\mathsf{log})$. Finally, due to Fact 3, $T_{\text{stop}}(R)$ - $T_{\text{start}}(R) < \lambda/G$. Therefore it holds that by time $T_{\text{stamp}}(R) = T_{\text{start}}(R) + \lambda/G + T_{\text{bft}} + \delta + T_{\text{snail}}$, every honest node's local $\mathsf{chain}[: -\lambda]$ will contain $\{R, \mathsf{hash}(\mathsf{log})\}_{\mathsf{pk}^{-1}}$, where log is the final daily log output by node i's $\mathsf{DailyBFT}[R]$ instance.

We now prove that such a tuple $\mathsf{rec} := \{R, h\}_{\mathsf{pk}^{-1}}$ is the first of its kind to appear in $\mathsf{chain}[: -\lambda]$, i.e., there is no other valid record of the form $\mathsf{rec}' := \{R, h'\}_{\mathsf{pk}^{-1}}$ where $h' \neq h$ preceeding rec in chain. We prove by contradition. Suppose there is such a tuple rec′ preceeding rec in $\mathsf{chain}[: -\lambda]$. Since rec must have appeared in $\mathsf{chain}[: -\lambda]$ at time T_{stamp} , due to the consistency property of snailchain, so must rec′. However, by the definition of the honest DailyBFT algorithm, an honest node should not have output two different $\mathtt{done}(\_, \_)$ messages with different final logs. Therefore, if rec′ appears in an honest node's $\mathsf{chain}[: -\lambda]$ by time T_{stamp}, this would obviously violate the unforgeability property of DailyBFT. □

Fact 5 (No late spawning of DailyBFT)**.** *Let G, G', Q be polynomially-bounded functions in $\lambda, n, \alpha, \delta$, Suppose that* DailyBFT *is secure against $(1-Q)$-corruption with liveness parameter T_{bft}. For any constant $\eta > 0$, let $T_{snail} := (1+\eta)\lambda/G$. Let $(\mathcal{A}, \mathcal{Z})$ be $(n, \alpha, \delta, \tau)$-valid w.r.t.* HybridConsensus *for some $n \in \mathbb{N}, \alpha > 0, \delta > 0$, and $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ such that* snailchain *satisfies consistency, Q-chain quality, and (G, G')-chain growth w.r.t. $(\mathcal{A}, \mathcal{Z})[\mathsf{snailchain}]$. Then, for any $\lambda \in \mathbb{N}$, with $1 - \mathsf{negl}(\lambda)$ probability, the following property holds for* $\textsc{exec}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ *and for any $R = \mathsf{poly}(\lambda) \in \mathbb{N}$.*

An honest node only forks an $\mathsf{DailyBFT}[R]$ *instance if it spawns by time $T_{stamp}(R)$.*

Proof. Straightforward by the definition of the HybridConsensus algorithm and Lemma 4. □

Lemma 5 (Retroactive consistency)**.** *Let $Q > \frac{2}{3}$, let G, G' be polynomially-bounded functions in $\lambda, n, \alpha, \delta$. Suppose that* DailyBFT *is secure against $(1-Q)$-corruption with liveness parameter T_{bft}. For any constant $\eta > 0$, let $T_{snail} := (1+\eta)\lambda/G$, let $(\mathcal{A}, \mathcal{Z})$ be $(n, \alpha, \delta, \tau)$-valid w.r.t.* HybridConsensus *for some $n \in \mathbb{N}, \alpha > 0, \delta > 0$, and $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ such that* snailchain *satisfies consistency, Q-chain quality, and (G, G')-chain growth w.r.t. $(\mathcal{A}, \mathcal{Z})[\mathsf{snailchain}]$. Then, for any $\lambda \in \mathbb{N}$, with $1-\mathsf{negl}(\lambda)$ probability, the following property holds for* $\textsc{exec}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ *and for any $R = \mathsf{poly}(\lambda) \in \mathbb{N}$.*

Let chain *denote the output of an honest node at any time. Suppose that (R, h) is an on-chain valid tuple w.r.t.* chain*, it holds that there exists a $\mathsf{pk} \in \mathsf{comm}_R$ output by a node i that is honest at $T_{stamp}(R)$, and $h = \mathsf{hash}(\mathsf{log}_R)$ where log_R is the (unique) final daily log output by node i's* $\mathsf{DailyBFT}[R]$ *instance.*

Proof. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for $\textsc{exec}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

For (R, h) to be a valid on-chain tuple w.r.t. chain, for at least $\lceil \mathsf{csize}/3 \rceil$ number of $\mathsf{pk} \in \mathsf{comm}_R$: 1) a correctly signed tuple $\{R, h\}_{\mathsf{pk}^{-1}}$ must appear in $\mathsf{chain}[: -\lambda]$; and 2) this tuple is the first occurrence of any valid tuple of the form $\{R, \_\}_{\mathsf{pk}^{-1}}$ on chain.

By Fact 4, at least Q fraction of comm_R were output by nodes that are honest at time $T_{\text{stamp}}(R)$. By assumption $Q > \frac{2}{3}$, and due to the pigeon-hole principle, for at least one such signature signed by some public key pk, it must hold that it is output by a node i (prior to $T_{\text{start}}(R)$) that remains honest till $T_{\text{stamp}}(R)$. By Lemma 4, this signature must vouch for $(R, \mathsf{hash}(\mathsf{log}))$ where log denotes the (unique) final log output by node i. □

Theorem 11 (Consistency for Hybrid Consensus)**.** *Suppose that* $\mathsf{hash}, \mathsf{H} : \{0,1\}^* \to \{0,1\}^\lambda$ *are independent random oracles, and that the signature scheme Σ is secure. Let $Q > \frac{2}{3}$, let G, G' be polynomially bounded functions in $\lambda, n, \alpha, \delta$. Suppose that* DailyBFT *is secure against $(1-Q)$-corruption with liveness parameter T_{bft}. For any constant $\eta > 0$, let $T_{snail} := (1+\eta)\lambda/G$. Let $(\mathcal{A}, \mathcal{Z})$*

be $(n, \alpha, \delta, \tau)$*-valid w.r.t.* HybridConsensus *for some* $n \in \mathbb{N}, \alpha > 0, \delta > 0$*, and* $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ *such that* snailchain *satisfies consistency, Q-chain quality, and* (G, G')*-chain growth w.r.t.* $(\mathcal{A}, \mathcal{Z})$[snailchain]*. Then, for any* $\lambda \in \mathbb{N}$*, with* $1-\mathsf{negl}(\lambda)$ *probability,* EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ *satisfies consistency as defined in Section 3.2.*

Proof. Self-consistency holds trivially from the definition of HybridConsensus. Below we focus on proving common prefix. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

By definition of the honest HybridConsensus algorithm, we can parse an honest node's LOG as the following for some $R \in \mathbb{N}$:

$$\mathsf{LOG} := \mathsf{log}_1 || \mathsf{log}_2 || \ldots || \mathsf{log}_{R-1} || \mathsf{log}_R$$

For each log_r where $r \in [R]$, it can be one of the following cases:

1. *Final log of daily offchain consensus.* log_r is the final log output by a DailyBFT$[r]$ instance. By Fact 5, it holds that this node spawned before $T_{\text{stamp}}(r)$.
2. *Non-final log of daily offchain consensus.* log_r is contained in an output of a DailyBFT$[r]$ instance, where the output is in the form of notdone(log_r) — in this case, by definition of the honest HybridConsensus algorithm, it must be the case that log_r is the last daily log included in LOG. Further, by Fact 5, it holds that this node spawned before $T_{\text{stamp}}(r)$.
3. *Matching on-chain valid tuple.* There is an on-chain valid tuple (r, h) w.r.t. the honest node's local chain$[: -\lambda]$ such that $\mathsf{hash}(\mathsf{log}_r) = h$.

First, note that if an honest node i's DailyBFT$[r]$ instance outputs log_r due to 1, and an honest node j's DailyBFT$[r]$ instance (same or different), outputs log'_r due to 1, it must hold that $\mathsf{log}_r = \mathsf{log}'_r$. This is straightforward by the consistency property of DailyBFT.

Second, note that if an honest node i's DailyBFT$[r]$ instance outputs log_r due to 1, and an honest node j's DailyBFT$[r]$ instance (same or different), outputs log'_r due to 2, then it must hold that $\mathsf{log}'_r \prec \mathsf{log}_r$. This follows immediately from the consistency property of DailyBFT.

Next, if an honest node j (same or different), outputs log'_r due to 3, there must exist an honest node i that outputs log_r due to 1, such that $\mathsf{log}_r = \mathsf{log}'_r$. This follows directly from Lemma 5, as well as the fact that the hash oracle has negligible probability of collision.

The rest of the proof follows in a straightforward manner, by observing that day lengths are polynomially bounded in λ due to Fact 3; and that for all $r = \mathsf{poly}(\lambda) \in \mathbb{N}$, at least at least one $\mathsf{pk} \in \mathsf{comm}_r$ is honest at $T_{\text{stamp}}(r)$ and will have output done$(_, _)$ by time $T_{\text{stamp}}(r)$.

□

Theorem 12 (Liveness for Hybrid Consensus)**.** *Suppose that* hash, H : $\{0,1\}^* \to \{0,1\}^\lambda$ *are independent random oracles, and that the signature scheme* Σ *is secure. Let* $Q > \frac{2}{3}$*, let* G *be a function in* $\lambda, n, \alpha, \delta$*. Suppose that* DailyBFT *is secure against* $(1 - Q)$*-corruption with liveness parameter* T_{bft}*. For any constant* $\eta > 0$*, let* $T_{snail} := (1 + \eta)\lambda/G$*. Let* $(\mathcal{A}, \mathcal{Z})$ *be* $(n, \alpha, \delta, \tau)$*-valid w.r.t.* HybridConsensus *for some* $n \in \mathbb{N}, \alpha > 0, \delta > 0$*, and* $\tau > 3\lambda/G + T_{bft} + \delta + T_{snail}$ *such that* snailchain *satisfies consistency, Q-chain quality, and* (G, G')*-chain growth w.r.t.* $(\mathcal{A}, \mathcal{Z})$[snailchain] *for some* $G' < \frac{\lambda}{T_{bft}+\delta}$*. Then, for any* $\lambda \in \mathbb{N}$*, with* $1-\mathsf{negl}(\lambda)$ *probability,* EXEC[HybridConsensus]$(\mathcal{A}, \mathcal{Z}, \lambda)$ *satisfies liveness as defined in Section 3.2 with parameters* $T_{warmup} := 2\lambda/G$*, and* $T_{confirm} := 2T_{bft} + 2\delta$*.*

Proof. For a $(\mathcal{A}, \mathcal{Z})$ pair satisfying the above requirements, the following statements hold with $1 - \mathsf{negl}(\lambda)$ probability for $\textsc{exec}[\mathsf{HybridConsensus}](\mathcal{A}, \mathcal{Z}, \lambda)$ for any $\lambda \in \mathbb{N}$.

Suppose that $\mathcal{Z}$ inputs TXs to some honest node at time $t \geq T_{\text{warmup}} = 2\lambda/G$. By definition of the txpool protocol, at time $t+\delta$, all honest nodes have $\mathsf{TXs} \subseteq \mathsf{txpool.TXs}$. Suppose that an honest node i is honest at some time $t' \geq t + T_{\text{confirm}} = t + 2T_{\text{bft}} + 2\delta$, and we show that node i's output LOG at time t' must include all of TXs.

By Fact 3, and the fact that $G' > \frac{\lambda}{T_{\text{bft}}+\delta}$, $T_{\text{start}}(1) \leq 2\lambda/G = T_{\text{warmup}}$. Further, for any $R \in \mathbb{N}$, $T_{\text{start}}(R) < T_{\text{stop}}(R) - T_{\text{bft}} - \delta < T_{\text{start}}(R+1)$. Let R be the *smallest* integer such that $t \leq T_{\text{stop}}(R) - T_{\text{bft}} - \delta$. Now, one of the following two cases has to be true:

- Case 1: $t \geq T_{\text{start}}(R) = T_{\text{stop}}(R-1)$.

  In this case, by definition of $\mathsf{HybridConsensus}$ and Fact 4, there exists an honest comm_R member whose $\mathcal{Z}[\mathsf{DailyBFT}[R]]$ will include $\mathsf{TXs}\backslash\mathsf{LOG}$ to in its input to $\mathsf{DailyBFT}[R]$ by $t+\delta < T_{\text{stop}}(R) - T_{\text{bft}}$.

- Case 2: $T_{\text{stop}}(R-1) - T_{\text{bft}} - \delta < t < T_{\text{stop}}(R-1) = T_{\text{start}}(R) < T_{\text{stop}}(R) - T_{\text{bft}}$

  In this case, by definition of $\mathsf{HybridConsensus}$ and Fact 4, there exists an honest comm_R member whose $\mathcal{Z}[\mathsf{DailyBFT}[R]]$ will include $\mathsf{TXs}\backslash\mathsf{LOG}$ to in its input to $\mathsf{DailyBFT}[R]$ by $T_{\text{start}}(R) < t + T_{\text{bft}} + \delta$.

Therefore, in either case, there exists an honest comm_R member whose $\mathcal{Z}[\mathsf{DailyBFT}[R]]$ will include $\mathsf{TXs}\backslash\mathsf{LOG}$ to in its input to $\mathsf{DailyBFT}[R]$ by at time $t^* < t + T_{\text{bft}} + \delta$ such that $T_{\text{start}}(R) \leq t^* < T_{\text{stop}}(R) - T_{\text{bft}}$. Henceforth let the LOG^* denote the output LOG of this committee member at time t^*.

- We first show that if node i ever forked a $\mathsf{DailyBFT}[R]$ instance, and is honest at time $t' \geq t + 2T_{\text{bft}} + 2\delta$, then node i's output LOG at time t' will include all of TXs.

  By Fact 5, if a node i forked a $\mathsf{DailyBFT}[R]$ instance, its $t_{\text{spawn}} \leq T_{\text{stamp}}(R)$. By the liveness property of the $\mathsf{DailyBFT}$ protocol and Fact 4, if node i is honest at time $t' \geq t + 2T_{\text{bft}} + 2\delta$, then by t' node i's $\mathsf{DailyBFT}[R]$ instance must have output $\mathtt{done}(\mathsf{log}_R, \_)$ such that $\mathsf{TXs}\backslash\mathsf{LOG}^* \subseteq \mathsf{log}_R$.

  Now, for any $1 \leq r \leq R-1$, by definition of the honest $\mathsf{HybridConsensus}$ protocol, either node i did not fork a $\mathsf{DailyBFT}[r]$ instance and log_r was computed during **Preprocess** (see Figure 4); or node i did fork a $\mathsf{DailyBFT}[r]$ instance. In the latter case, we know that node i forked by $T_{\text{stamp}}(r)$ due to Fact 5. By Fact 4 and the timely termination property of the the $\mathsf{DailyBFT}$ protocol, if node i is honest at time $t' \geq t + 2T_{\text{bft}} + 2\delta$, node i's $\mathsf{DailyBFT}[r]$ instance will have output $\mathtt{done}$ by time t'.

  By the definition of the honest $\mathsf{HybridConsensus}$ protocol, if node i' is honest at time $t' \geq t + 2T_{\text{bft}} + 2\delta$, its output LOG will contain all of $\mathsf{TXs}\backslash\mathsf{LOG}^*$ at time t'.

- Now, consider the case when node i did not fork a $\mathsf{DailyBFT}[R]$ instance. In this case, node i must have computed log_R by matching on-chain valid tuples, and log_R is output in time t_{spawn}.

  Due to the Fact 4, there must exist at least one $\mathsf{pk} \in \mathsf{comm}_R$ output by some node ν that is honest at time $T_{\text{stamp}}(R)$, and by the timely termination property of $\mathsf{DailyBFT}$, node ν's $\mathsf{DailyBFT}[R]$ instance will output log_R during the time it is honest. By the liveness property of $\mathsf{DailyBFT}$, this final log log_R output by node ν must include all of $\mathsf{TXs}\backslash\mathsf{LOG}^*$.

  Due to Theorem 11, the log_R output by node i must be the same as that output by node ν's $\mathsf{DailyBFT}[R]$ instance, and thus must include all of $\mathsf{TXs}\backslash\mathsf{LOG}^*$ as well.

Finally, due to the consistency property of HybridConsensus, for any node that is honest at time $t' \geq t + 2T_{\text{bft}} + 2\delta > t^*$ let LOG be its output at time t', then it holds that either $\mathsf{LOG}^* \prec \mathsf{LOG}$ or $\mathsf{LOG} \prec \mathsf{LOG}^*$. However, since $\mathsf{TXs} \backslash \mathsf{LOG}^* \subseteq \mathsf{LOG}$, it must be the case that $\mathsf{LOG}^* \prec \mathsf{LOG}$. Therefore we conclude that $\mathsf{TXs} \subseteq \mathsf{LOG}$.

□

8.3 Daily Offchain Consensus Proofs

In some view, suppose that $(\mathcal{A}, \mathcal{Z})$ is an adversary/environment pair for the protocol DailyBFT, we use the notation $(\mathcal{A}, \mathcal{Z})[\mathsf{BFT}]$ to denote the adversary/environment pair that interfaces with a BFT subprotocol instance. $(\mathcal{A}, \mathcal{Z})[\mathsf{BFT}]$ is jointly defined by $(\mathcal{A}, \mathcal{Z})$ as well as the honest DailyBFT protocol. Notice that upon corruption of a node i, $\mathcal{Z}[\mathsf{BFT}]$ reveals to $\mathcal{A}[\mathsf{BFT}]$ all public/secret keys of all BFT instances running on node i.

Lemma 6 (BFT security within the context of DailyBFT). *If* BFT *is strongly secure against* $(1-Q)$ *corruption and with liveness parameter* T_{bft} *by Definition 5, and further* BFT *is instantiated with a secure signature scheme* Σ*, then for any* $n, \delta, T_{stamp} > 0$*,* $\tau \geq 0$*, for any p.p.t.* $(\mathcal{A}, \mathcal{Z})$ *that is* $(n, Q, \delta, \tau, T_{stamp}, T_{bft})$*-valid w.r.t.* DailyBFT*, there exists a negligible function* negl *such that for any* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{view} \leftarrow \text{EXEC}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda) : \ \mathsf{secure}^{T_{bft}}(\mathsf{view}) = 1\right] \geq 1 - \mathsf{negl}(\lambda)$$

where $\mathsf{secure}^{T_{bft}}(\mathsf{view})$ *is as defined as in Section 4.2 — but replace any occurrence of "an honest node" with "an honest* BFT *virtual node".*

Proof. We construct a p.p.t. reduction Re as below.

- The reduction Re interacts with a signature challenger Σ^*, and obtains a signature public key denoted pk^* upfront. Henceforth we refer to pk^* as the challenge public key.
- The reduction Re will simulate in its head all honest nodes running DailyBFT. When some honest node's BFT virtual node asks for `keygen`, Re flips a random coin, and with probability $1/n$, it will return the challenge public key pk^*; otherwise, the reduction Re generates the signing key pair using the honest algorithm. Therefore, the reduction Re knows all secret signing keys except for the coordinate where the challenge public key pk^* is embedded. Whenever an honest node's BFT virtual node issues a `sign` query for the challenge public key pk^*, Re simply forwards the query to the signature challenger.
- The reduction Re interacts with the environment $\mathcal{Z}$. Whenever $\mathcal{Z}$ inputs anything to honest nodes, Re simply forwards the message to the corresponding honest node being simulated in its head. Whenever a simulated honest node needs to output anything to $\mathcal{Z}$, the reduction Re simply forwards the message. Similarly, the reduction Re also forwards any messages between $\mathcal{Z}$ and $\mathcal{A}$.

  Now, if an honest node becomes corrupt at some point, Re needs to return the node's private state to $\mathcal{Z}$. As long as the corrupt node is not the one where pk^* is embedded, the reduction Re can successfully simulate. If the corrupt node happens to be where pk^* is embedded, the reduction simply aborts. It is obvious that the probability that the reduction does not abort is non-negligible.

It is obvious that conditioned on the event that the reduction Re does not abort, the reduction's interfaces to $(\mathcal{A}, \mathcal{Z})$ are identically distributed as a real execution of the $\mathsf{DailyBFT}$ protocol.

It is not hard to see that if $(\mathcal{A}, \mathcal{Z})$ is $(n, Q, \delta, \tau, T_{\rm stamp}, T_{\rm bft})$-valid w.r.t. $\mathsf{DailyBFT}$, then with $1 - \mathsf{negl}(\lambda)$ probability, $(\mathcal{A}, \mathcal{Z})[\mathsf{BFT}]$ is $(n, Q, \delta, \tau, T_{\rm stamp})$-valid w.r.t. BFT. In particular, honestly generated public keys using the $\Sigma.\mathsf{Gen}(1^\lambda)$ algorithm have $\mathsf{negl}(\lambda)$ probability of collision, since otherwise the signature scheme can easily be broken.

For the sake of a contradition, suppose that the lemma is not true, i.e., there exists some polynomial g and p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is $(n, Q, \delta, \tau, T_{\rm stamp}, T_{\rm bft})$-valid w.r.t. $\mathsf{DailyBFT}$ such that

$$\Pr\left[\mathsf{view} \leftarrow \textsc{exec}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda) : \ \mathsf{secure}^{T_{\rm bft}}(\mathsf{view}) \neq 1\right] \geq g(\lambda)$$

where $\mathsf{secure}^{T_{\rm bft}}(\mathsf{view})$ is defined just like in Section 4.2, but with respect to the virtual BFT nodes inside $\mathsf{DailyBFT}$. By definition of strong security (see Definition 5), we know that there exists a p.p.t. adversary $\mathcal{B}$ and polynomial g' such that

$$\Pr\left[\mathsf{view} \leftarrow \textsc{exec}[\mathsf{DailyBFT}](\mathcal{B}, \mathcal{Z}, \lambda) : \ \mathsf{forgery}(\mathsf{view}) = 1\right] \geq g'(\lambda)$$

Now, consider an execution where the reduction Re is interacting with $(\mathcal{B}, \mathcal{Z})$. As mentioned earlier, as long as the execution does not abort, the execution is identically distributed as a real execution from the perspectives of $(\mathcal{B}, \mathcal{Z})$. Therefore, with non-negligible probability, $\mathcal{B}$ will output to Re some forgery, and if the forgery happens to be for pk^* which happens with non-negligible probability, the reduction Re will have found a forgery to the signature scheme. □

Theorem 13 (DailyBFT from BFT, restatement of Theorem 8)**.** *Suppose that the signature scheme Σ employed by* DailyBFT *is secure, and that* hash *is a random oracle. Suppose that* BFT *is secure against $(1 - Q)$-corruption with liveness parameter $T_{bft}{}'$ for $Q > \frac{2}{3}$. Then,* DailyBFT *is secure against $(1 - Q)$-corruption with liveness parameter $T_{bft} := T_{bft}{}' + \delta$.*

Proof. For any $n, \delta, T_{\rm stamp} > 0$, any $\tau \geq 0$, for any $(\mathcal{A}, \mathcal{Z})$ that is $(n, Q, \delta, \tau, T_{\rm stamp}, T_{\rm bft})$-valid w.r.t. $\mathsf{DailyBFT}$, the following properties hold for $\textsc{exec}[\mathsf{DailyBFT}](\mathcal{A}, \mathcal{Z}, \lambda)$ except with negligible probability:

- *Timely termination.* Since the environment of BFT respects close stop, all BFT virtual nodes receive `stop` by $T_{\rm stop} + \delta$. By the liveness property of BFT, and definition of $\mathsf{DailyBFT}$, all honest committee members that remain honest till $T_{\rm stop} + T_{\rm bft}{}' + \delta = T_{\rm stop} + T_{\rm bft}$ will have in their log by time $T_{\rm stop} + T_{\rm bft}$, `stop` transactions signed by $\lceil |\mathsf{comm}|/3 \rceil$ distinct public keys in comm. When this happens, by definition of $\mathsf{DailyBFT}$ honest committee members gossip the signed hash of their daily log, and output `done`$(\_, \_)$. Since at least $Q > \frac{2}{3}$ fraction of comm were output by committee members that remain honest till $T_{\rm stamp} > T_{\rm stop} + T_{\rm bft}$, it is not hard to see that if a committee non-member is honest at time $T_{\rm stop} + T_{\rm bft} + \delta$, it will have collected enough signatures and will have output `done`$(\_, \_)$.
- *Consistency.*
  - *Self-consistency.* Self-consistency is obvious by definition of $\mathsf{DailyBFT}$.
  - *Termination agreement.* By definition, $\mathsf{DailyBFT}$ an honest committee member simply outputs the same final log as the final log output by the inner BFT_0 virtual node (with `stop` transactions removed). By the consistency property of the underlying BFT, if an honest committee member i's BFT outputs log at some time $t < T_{\rm stamp}$, and an honest committee

member j's BFT outputs $\log'$ at some time $t' < T_{\text{stamp}}$, it must hold that $\log \prec \log'$ or $\log \prec \log'$.

If both node i and j spawned before T_{stamp} and output $\mathtt{done}(\_, \_)$ while they are still honest, then $\mathtt{done}(\_, \_)$ must be output no later than T_{stamp}. By the definition of the honest $\mathsf{DailyBFT}$ algorithm, it must be the case they output identical daily logs upon termination.

We now extend the termination agreement proof to committee non-members who spawned before T_{stamp}. This will rely on the security of the signature scheme. Suppose that some committee non-member k spawned before T_{stamp}, and outputs $\mathtt{done}(\mathsf{log}^*, \_)$ at some time at which it is honest. It must be the case that

1. $\mathtt{done}(\mathsf{log}^*, \_)$ is output by time T_{stamp} due to the timely termination property;
2. log^* can be parsed as
$$\mathsf{log}^* = \mathsf{tx}_1^*||\mathsf{tx}_2^*|| \ldots \mathsf{tx}_{\ell^*}^*$$
3. For each (R, i, tx_i) where $i \in [\ell^*]$, and for the tuple (R, ℓ) where R is the session identifier, node k has received valid signatures signed by at least $\lceil \frac{1}{3}|\mathsf{comm}| \rceil$ number of distinct public keys in comm.

Due to the fact that that at least Q fraction of comm are output by committee members which are honest at T_{stamp}, and by the pigeonhole principle, at least one valid signature on each tuple (R, i, tx_i) or the tuple (R, ℓ^*) is by $\mathsf{pk} \in \mathsf{comm}$ output by a node ν that remains honest till T_{stamp}. As mentioned earlier, honest committee members output identical final daily logs, henceforth denoted log. Parse $\mathsf{log} := \mathsf{tx}_1|| \ldots ||\mathsf{tx}_\ell$.

Suppose for a contradiction that $\mathsf{log}^* \neq \mathsf{log}$. Then there must exist a tuple (R, i, tx_i^*) or (R, ℓ^*) such that $\mathsf{tx}_i^* \neq \mathsf{tx}_i$ or $\ell^* \neq \ell$. Without loss of generality, assume that $\mathsf{tx}_i^* \neq \mathsf{tx}_i$ (the other case is similar). Then, node k will have received at least one valid signature on (R, i, tx_i^*) signed by a $\mathsf{pk} \in \mathsf{comm}$ output by a node ν that remains honest till T_{stamp}. If there exists p.p.t. $(\mathcal{A}, \mathcal{Z})$ such that $\mathsf{log}^* \neq \mathsf{log}$ with non-negligible probability, then we can build a reduction that break the signature scheme. Basically this reduction simulates all honest users and interact with the $(\mathcal{A}, \mathcal{Z})$ pair. Whenever $\mathcal{Z}$ inputs keygen, it has a choice of embedding a pk from the signature's challenger. To do this, every time a keygen is queried, the reduction makes a random guess and decides whether to embed the signature challenger's pk. Such guesses are correct with $1/\mathsf{poly}(\lambda)$ probability – and if the guess turns out to be wrong later the reduction simply aborts. Whenever the embedded pk needs to sign something, the reduction queries the signature challenger. For all other key pairs the reduction knows the corresponding secret key and can disclose it to $(\mathcal{A}, \mathcal{Z})$ when a node becomes corrupt. Finally, when $(\mathcal{A}, \mathcal{Z})$ outputs a valid signature on such a tuple $\{R, i, \mathsf{tx}_i^*\}_{\mathsf{pk}^{-1}}$, the reduction outputs this as a forgery. Note that the reduction should never have to make such a query to the signature challenger since $\mathsf{tx}_i^* \neq \mathsf{tx}_i$.

– *Common prefix.* Suppose that committee members i and j are honest at times t and t' respectively and i outputs log at time t and j outputs log at time t'. Due to the timely termination property, it must be the case that $t < T_{\text{stamp}}$ and $t' < T_{\text{stamp}}$. Now, due to the consistency property of the underlying BFT and definition of $\mathsf{DailyBFT}$, it is not hard to see that either $\mathsf{log} \prec \mathsf{log}'$ or $\mathsf{log}' \prec \mathsf{log}$.

We now extend the argument to committee non-members who spawned before T_{stamp}. Here we rely on the security of the signature scheme. Let k be any committee non-member that spawned before T_{stamp}. We have argued that if node k outputs $\mathtt{done}(\mathsf{log}', \_)$ then log' must agree with honest committee members' final log denoted log — as we argued earlier, all honest committee members must output an identical final log; and by definition there exists at least one committee member who remains honest till T_{stamp}.

If node k outputs $\texttt{notdone}(\mathsf{log}^*)$ at a time when it is honest, then due to the timely termination property and the fact that honest nodes never output again after outputting $\texttt{done}(\_, \_)$, it must hold that $\texttt{notdone}(\mathsf{log}^*)$ is output before T_{stamp}. Now parse $\mathsf{log}^* := \mathsf{log}^*_1 || \ldots || \mathsf{log}^*_m$, and parse $\mathsf{log} := \mathsf{log}_1 || \ldots || \mathsf{log}_\ell$. Then it must hold that $\ell \geq m$ and $\mathsf{log}^*_i = \mathsf{log}_i$ — the latter can be shown using exactly the same type of argument as termination agreement.

- *Liveness.* By definition of the DailyBFT protocol, any TXs input to an honest committee member at time t is input to the inner BFT virtual nodes in the same time step t. Further, the inner BFT virtual nodes have the same T_{start} and T_{stop} as the outer DailyBFT, since honest committee members simply pass $\texttt{start}$ and $\texttt{stop}$ commands to the inner BFT virtual nodes.

  By liveness of the BFT protocol, if $T_{\text{start}} \leq t < T_{\text{stop}} - T_{\text{bft}}$, for any committee member honest at time $t + T_{\text{bft}}$, its inner BFT virtual nodes will have output a log that includes all of TXs by time $t + T_{\text{bft}}$. Since $t < T_{\text{stop}} - T_{\text{bft}}$, $\mathsf{complete}(\mathsf{log})$ must return false, since otherwise we can easily construct an adversary that breaks signature security.

  Now consider the set of all committee members honest at time $t + T_{\text{bft}}$, for each committee member, consider the longest log it has output by time $t + T_{\text{bft}}$. Now take the intersection of all such logs and denote it as log'. We have argued that $\mathsf{TXs} \subseteq \mathsf{log}'$, and clearly $\mathsf{complete}(\mathsf{log}') = \mathsf{false}$. Now, by definition of the DailyBFT protocol, for every $i \in [|\mathsf{log}'|]$, for every $\mathsf{pk} \in \mathsf{comm}$ output by some node that is honest at T_{stamp}, it must hold that every node honest at time $t + T_{\text{bft}} + \delta$ must have received a validly signed tuple $\{R, i, \mathsf{log}'[i]\}_{\mathsf{pk}^{-1}}$ where R is the session identifier of DailyBFT. There must be at least $\lceil Q|\mathsf{comm}| \rceil$ number of pks in comm output by nodes who remain honest till $T_{\text{stamp}} > t + T_{\text{bft}} + \delta$. Since $Q > \frac{2}{3}$, any node honest at time $t + T_{\text{bft}} + \delta$ will have output a log^* s.t. $|\mathsf{log}^*| \geq i$ by time $t + T_{\text{bft}} + \delta$. By consistency of DailyBFT, $\mathsf{log}^*[i] = \mathsf{log}'[i]$, and therefore $\mathsf{tx} \in \mathsf{log}^*$.

- *Completeness.* Obvious by definition of DailyBFT and correctness of the signature scheme.

- *Unforgeability.* If there exists p.p.t. $(\mathcal{A}, \mathcal{Z})$ such that unforgeability can be broken with non-negligible probability, then we can build a reduction that break the signature scheme. Basically this reduction simulates all honest users and interact with the $(\mathcal{A}, \mathcal{Z})$ pair. Whenever $\mathcal{Z}$ inputs keygen, it has a choice of embedding a pk from the signature's challenger. To do this, every time a keygen is queried, the reduction makes a random guess and decides whether to embed the signature challenger's pk. Such guesses are correct with $1/\mathsf{poly}(\lambda)$ probability – and if the guess turns out to be wrong later the reduction simply aborts. Whenever the embedded pk needs to sign something, the reduction queries the signature challenger. For all other key pairs the reduction knows the corresponding secret key and can disclose it to $(\mathcal{A}, \mathcal{Z})$ when a node becomes corrupt. Henceforth we assume that the embeded pk belongs to node i. Finally, when $(\mathcal{A}, \mathcal{Z})$ outputs a valid signature on such a tuple $\{R, h\}_{\mathsf{pk}^{-1}}$, the reduction outputs this as a forgery. Note that the reduction should never have to make such a query (R, h) to the signature challenger since by definition, for the $(\mathcal{A}, \mathcal{Z})$ pair to break unforgeability, it must be the case that node i has not output $\texttt{done}(\mathsf{log}, \_)$ by the forgery such that $h = \mathsf{hash}(\mathsf{log})$.

□

8.4 Extending the Proof for Hybrid Consensus over Fruitchain

So far, we have completed the proof for hybrid consensus over Nakamoto. When we use Fruitchain as the underlying snailchain, the proof is almost the same, except that parameters must be readjusted.

Consider $\mathsf{HybridConsensus}^{\lambda,\eta}$ parametrized with η. In other words, the committee size csize is chosen to be $\mathsf{csize} := \lambda/\eta$. In the proof, we can plug in the following modified parameters:

- $T_{\text{snail}} := (2\lambda + \frac{\lambda}{\eta})/G$;
- $T_{\text{stamp}}(R) := T_{\text{start}}(R) + \frac{\lambda}{\eta G} + T_{\text{bft}} + \delta + T_{\text{snail}}$;
- $\tau > (\frac{2\lambda}{\eta} + \lambda)/G + T_{\text{bft}} + \delta + T_{\text{snail}} = 3\lambda(1 + \frac{1}{\eta})/G + T_{\text{bft}} + \delta$;
- $T_{\text{warmup}} := \lambda(1 + \frac{1}{\eta})/G$;
- $G' > \frac{\lambda}{\eta(T_{\text{bft}}+\delta)}$;
- $G = (1 - 5\eta)(1 - \alpha)n\rho$, and $G' = (1 + 5\eta)n\rho$.

With these new parameters, the rest of the proof follows in the same manner as hybrid consensus over Nakamoto.

9 Lower Bound

9.1 Proof-of-Work Cannot Stop

We now prove a lower bound suggesting that any secure permissionless consensus protocol must invoke proofs-of-work infinitely often, assuming no additional trust assumptions. We stress that this lower bound does not rule out approaches that rely on additional trust assumptions such as proofs-of-stake [8, 36, 49, 50].

Theorem 14 (Any secure permissionless consensus protocol must call proofs-of-work infinitely often.)**.** *Let Γ denote any binary function in n, α, δ, τ such that $\Gamma(n, \alpha, \delta, \tau) = 1$ for some n, δ that are positive polynomials (in λ), non-negative polynomial τ, and $\alpha > 1/\mathsf{poly}$ for some positive polynomial* poly*. Let Π be a protocol such that for any p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is Γ-admissible, there exists a polynomial function* poly *such that for every $\lambda \in \mathbb{N}$, $\textsc{exec}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$ satisfies the following properties with $1/\mathsf{poly}(\lambda)$ probability:*

- *Bounded proof-of-work. Honest nodes stop querying* H *after some time $T_{pow} := \mathsf{poly}_0(n, \alpha, \delta, \lambda)$ for some polynomial poly_0;*
- *Liveness. Liveness as defined in Section 3.2 is satisfied with parameters $T_{confirm} = \mathsf{poly}_1(n, \alpha, \delta, \lambda)$, $T_{warmup} = \mathsf{poly}_2(n, \alpha, \delta, \lambda)$, and $T_{bootstrap} = \mathsf{poly}_3(n, \alpha, \delta, \lambda)$, for some non-negative polynomial poly_1, poly_2, and poly_3.*

Then, there exists p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is Γ-admissible, such that for any $\lambda \in \mathbb{N}$, $\textsc{exec}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$ does not satisfy consistency with probability $1/\mathsf{poly}(\lambda)$ for some polynomial poly*.*

Intuitively, this theorem says that any permissionless consensus protocol secure against $1/\mathsf{poly}(\lambda)$ fraction of corruption must call proofs-of-work infinitely often — *even in the synchronous network model and against static corruptions.*

Proof. Let Π be any protocol such that for any p.p.t. $(\mathcal{A}', \mathcal{Z}')$ that is Γ-admissible, there exists a polynomial function poly such that for every $\lambda \in \mathbb{N}$, with $1/\mathsf{poly}(\lambda)$ probability, $\textsc{exec}[\Pi](\mathcal{A}', \mathcal{Z}', \lambda)$ satisfies bounded proof-of-work and liveness as defined above. We now show how to construct p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is Γ-admissible such that $\textsc{exec}[\Pi](\mathcal{A}, \mathcal{Z}, \lambda)$ breaks the common prefix property with $1/\mathsf{poly}(\lambda)$ probability for some polynomial poly.

We consider a pair $(\mathcal{A}, \mathcal{Z})$ that is Γ-admissible w.r.t. Π, and behaves as follows.

Transaction input. At time T_{warmup}, $\mathcal{Z}$ samples $\mathsf{tx} \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$, and inputs $\mathsf{TXs} := \{\mathsf{tx}\}$ to an honest node. Besides this, $\mathcal{Z}$ does not input any other transactions.

Real execution. $\mathcal{A}$ instructs all corrupt nodes to behave honestly in the real execution.

Simulated execution. Starting at time T_{pow}, $\mathcal{A}$ also simulates an imaginary execution in its head. To do this, $\mathcal{A}$ simulates the execution of all honest nodes and the environment $\mathcal{Z}$. Suppose that in this simulated execution, the simulated environment $\mathcal{Z}$ inputs $\mathsf{TXs} := \{\mathsf{tx}^*\}$ where $\mathsf{tx}^* \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$ at simulated time T_{warmup}.

Network and corruption. In both the real and the simulated execution, the adversary $\mathcal{A}$ delivers messages instantly, i.e., within the next time step. The environment statically corrupts α fraction of the nodes.

Late spawning node. Suppose that a new node i spawns at time $t_{\text{spawn}} := \max(T_{\text{pow}} + T_{\text{pow}}/\alpha, T_{\text{warmup}} + T_{\text{confirm}}) + 1$. At this moment, the adversary $\mathcal{A}$ will have all the simulated honest nodes interact with node i where all simulated honest nodes follow the honest protocol. Whenever node i gossips a message, the message is delivered to both the honest nodes in the real execution within $\delta = 1$ time, as well as delivered to honest nodes in the simulated execution.

No consistency. Now, with $1/\mathsf{poly}(\lambda)$ probability, both bounded proof-of-work and liveness are satisfied for the real and simulated execution. Conditioned on the fact these properties are satisfied for the real and simulated execution, we argue that consistency cannot be satisfied with at least $1/2$ probability.

Given that α fraction of the nodes are corrupt, it is not hard to see that at any time $t \geq t_{\text{spawn}} - 1$, the adversary $\mathcal{A}$ is able to output a simulated execution that is identically distributed as the real execution, since the adversary $\mathcal{A}$ will have enough time to make all the necessary H queries.

Now, due to liveness, honest nodes in the real execution must have output a LOG by $t_{\text{spawn}} - 1$ where $\mathsf{tx} \in \mathsf{LOG}$ and $\mathsf{tx}^* \notin \mathsf{LOG}$ — except with $\mathsf{negl}(\lambda)$ probability since the the real execution cannot know tx^* before t_{spawn}. Similarly, except with $\mathsf{negl}(\lambda)$ probability, honest nodes in the simulated execution must have output a LOG by $t_{\text{spawn}} - 1$ and where $\mathsf{tx}^* \in \mathsf{LOG}$ and $\mathsf{tx} \notin \mathsf{LOG}$.

Due to liveness, the newly spawned node i must output a non-empty LOG by time $\max(t_{\text{spawn}} + T_{\text{bootstrap}}, T_{\text{warmup}} + T_{\text{confirm}})$ such that $\mathsf{tx} \in \mathsf{LOG}$. And since the real and simulated execution are identically distributed, $\mathsf{tx}^* \in \mathsf{LOG}$ too. Since the real and simulated execution are identically distributed, the probability that tx^* precedes tx in LOG is at least $1/2$ — in which case consistency (for the real execution) cannot be satisfied.

□

9.2 1/3 Corruption is Tight for Responsive Protocols

We now show that in the permissionless model, even when the protocol knows an a-priori upper bound Δ of the network's delay, there does not exist responsive protocols that can tolerate 1/3 or more corruption in terms of hashpower. Since our hybrid consensus protocol tolerates $1/3 - \epsilon$ corruption, it is (nearly) tight since no responsive protocol can tolerate more than 1/3 corruption.

Our lower bound is a straightforward modification of a related lower bound proven by Sompolinsky [1], who showed that in the partially synchronous setting, if the network's delay upper bound is unknown to the protocol, then no secure permissionless consensus protocol can tolerate more than 1/3 corruption. Our lower bound proof (and also Sompolinsky's) is also close in spirit to the partially synchronous lower bound shown by Dwork, Lynch, and Stockmeyer [23] — however, their bound needs to be adapted to the permissionless setting with proof-of-work. In particular, Dwork et al.'s lower bound constructs an explicit attack with 3 nodes, where one node controlled by the adversary acts as two separate nodes with different inputs, and interact with two honest nodes to split their views. In the proof-of-work setting, the difficulty is that the adversary cannot simultaneously simulate two nodes since to do that it would have to solve twice the proof-of-work. However, we use a trick similar to Sompolinsky [1], where the adversary still acts as two players, but space out the proof-of-work over time — and the victim honest node cannot distinguish whether the adversary started solving puzzles late, or simply the network delay is large.

Theorem 15 (Responsive protocols cannot tolerate 1/3 corruption)**.** *No secure permissionless consensus protocol that is also responsive can tolerate* 1/3 *or more corruption.*

Proof. Suppose that there exists a protocol Π that defends against 1/3 corruption and is responsive, i.e., its liveness parameter $T_{\text{confirm}} = T_{\text{confirm}}(\lambda, n, \alpha, \delta)$ is a function of the network's actual delay δ, but not of the a-priori known upper bound delay Δ. This means that after some $T_{\text{warmup}} = \mathsf{poly}(\lambda, n, \alpha, \delta, \Delta)$ time, a transaction input to an honest node will be included in any honest node's output LOG within T_{confirm} time, even when 1/3 of the nodes crash.

We now describe an explicit attack that can break consistency when $\alpha = 1/3$. Suppose that there are 3 nodes, A, P_0, and P_1, where A is controlled by the adversary $\mathcal{A}$, and P_0 and P_1 are honest. Let $\Delta := 2T_{\text{confirm}}(\lambda, n, \alpha, T_{\text{confirm}}(\lambda, n, \alpha, 1))$ which is polynomial bounded in terms of λ. The adversary $\mathcal{A}$ first behaves honestly and delivers all messages instantly until $T_{\text{warmup}}(\lambda, n, \alpha, 1, \Delta)$ time has passed.

At this time, the adversary $\mathcal{A}$ starts to delay messages between P_0 and P_1 for the maximum amount Δ, but delivers messages instantly between P_0 and A. At this time, the environment $\mathcal{Z}$ inputs a transaction $\mathsf{tx} \overset{\$}{\leftarrow} \{0,1\}^\lambda$ to P_0 and a different transaction $\mathsf{tx}' \overset{\$}{\leftarrow} \{0,1\}^\lambda$ to P_1. Further, the corrupt node A stops sending messages to P_1; however, it remembers and stores every message received from P_1. The corrupt node follows the honest protocol when interacting with P_0. Since the protocol is responsive even when 1/3 nodes crash, except with negligible probability, P_0's output LOG will include tx in some fixed polynomial time $T_{\text{confirm}}(\lambda, n, \alpha, 1) < \Delta$.

Let $t^* = T_{\text{warmup}}(\lambda, n, \alpha, 1, \Delta) + T_{\text{confirm}}(\lambda, n, \alpha, 1)$ denote an upper bound on the time when tx is included in P_0's output LOG. At time t^*, the corrupt node A stops sending messages to P_0, but begins interating with P_1 as follows. First, A resets its internal state to what it was at time T_{warmup}. Recall that A queues all messages received P_1 in a buffer. It will now pretend that any real time $t \geq t^*$ is fake time $t - T_{\text{confirm}}(\lambda, n, \alpha, 1)$, and that it replays (from the buffer) all messages received from P_1 during the real time step $t - T_{\text{confirm}}(\lambda, n, \alpha, 1)$. Now A follows the honest protocol, and for every message destined for P_1, the adversary delivers the message instantly. Note that P_1 cannot distinguish whether A started solving proofs-of-work $T_{\text{confirm}}(\lambda, n, \alpha, 1)$ time late, or whether the network link from A to P_1 has $T_{\text{confirm}}(\lambda, n, \alpha, 1)$ actual delay, but the P_1 to A link delivers messages instantly. Since the protocol is responsive even when 1/3 nodes crash, P_1 will include $\mathsf{tx}' \neq \mathsf{tx}$ in its output LOG in $T_{\text{confirm}}(\lambda, n, \alpha, T_{\text{confirm}}(\lambda, n, \alpha, 1)) < \Delta$ time. Note that since Δ is large, P_1 has not heard the transaction tx from P_0 yet. Therefore, the probability that tx is in P_1's output log at time $T_{\text{confirm}}(\lambda, n, \alpha, T_{\text{confirm}}(\lambda, n, \alpha, 1))$ is negligibly small in λ. Clearly, this breaks consistency. □

Remark. We note that it is not hard to show a similar lower bound for the classical permissioned setting. Specifically, in the classical permissioned setting even when assuming PKI, any responsive, secure consensus protocol cannot tolerate 1/3 corruption or more. Such a lower bound would be a straightforward generalization of Dwork et al.'s lower bound proof for partial synchrony with an unknown Δ (in the permissioned setting).

Acknowledgments

We thank Kyle Croman, Ittay Eyal, Ari Juels, Antonio Marcedone, Andrew Miller, Emin Gun Sirer, Yonatan Sompolinsky, Dawn Song, and Fan Zhang for helpful discussions, and especially Dominic Williams for several excellent discussions that inspired us to tackle this problem.

This work is supported in part by NSF grants CNS-1217821, CNS-1314857, CNS-1514261, CNS-1544613, CNS-1561209, CNS-1601879, CNS-1617676, AFOSR Award FA9550-15-1-0262, an Office of Naval Research Young Investigator Program Award, a Microsoft Faculty Fellowship, a Packard Fellowship, a Sloan Fellowship, Google Faculty Research Awards, and a VMWare Research Award. This work was done in part while a subset of the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467.

References

[1] Personal communication with Yonatan Sompolinsky.

[2] Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Theoretical Aspects of Computer Software, 4th International Symposium (TACS)*, pages 82–94, 2001.

[3] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[4] Pedro Adão, Gergei Bana, Jonathan Herzog, and Andre Scedrov. Soundness of formal encryption in the presence of key-cycles. In *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, pages 374–396, 2005.

[5] Gavin Andresen. Increase maximum block size (bip 101). `https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki`, retrieved October 2015.

[6] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timon, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. https://blockstream.com/sidechains.pdf.

[7] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, 2003:15, 2003.

[8] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies without proof of work. *CoRR*, abs/1406.5694, 2014.

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

[10] Florian Bohl and Dominique Unruh. Symbolic universal composability. In *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium*, CSF '13, pages 257–271, 2013.

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

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

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

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

[15] Ran Canetti and Jonathan Herzog. Universally composable symbolic security analysis. *J. Cryptology*, 24(1):83–147, 2011.

[16] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, 1993.

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

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

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

[20] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, 2016.

[21] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P*, pages 1–10. IEEE, 2013.

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

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

[24] Pieter Wuille Eric Lombrozo, Johnson Lau. Segregated witness (consensus layer). `https://github.com/CodeShark/bips/blob/segwit/bip-codeshark-jl2012-segwit.mediawiki`.

[25] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert van Renesse. Bitcoin-NG: A scalable blockchain protocol. In *NSDI*, 2016.

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

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

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

[29] Jeff Garzik. Block size increase to 2mb (bip 102). `https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki`, retrieved October 2015.

[30] Jeff Garzik. Making decentralized economic policy. `http://gtf.org/garzik/bitcoin/BIP100-blocksizechangeproposal.pdf`.

[31] Omer Horvitz and Virgil D. Gligor. Weak key authenticity and the computational completeness of formal encryption. In *CRYPTO*, pages 530–547, 2003.

[32] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS)*, pages 172–185, 2005.

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

[34] Aggelos Kiayias and Giorgos Panagiotakos. Speed-security tradeoffs in blockchain protocols. Cryptology ePrint Archive, Report 2015/1019, 2015.

[35] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. *CoRR*, abs/1602.06997, 2016.

[36] Jae Kwon. TenderMint: Consensus without Mining, August 2014.

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

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

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

[40] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, November 2006.

[41] Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. Scp: A computationally-scalable byzantine consensus protocol for blockchains. Cryptology ePrint 2015/1168.

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

[43] Daniele Micciancio and Bogdan Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *J. Comput. Secur.*, 12(1):99–129, January 2004.

[44] Daniele Micciancio and Bogdan Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference (TCC)*, 2004.

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

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

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

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

[49] Ryan Pfeiffer. The decrits consensus algorithm: Decentralized agreement without proof of work. *CoRR*, abs/1411.1101, 2014.

[50] QuantumMechanic. Proof of stake instead of proof of work. `bitcointalk.org`, July 2011.

[51] David Schwartz, Noah Youngs, and Arthur Britto. The Ripple Protocol Consensus Algorithm, September 2014.

[52] Isaac C. Sheff, Robbert van Renesse, and Andrew C. Myers. Distributed protocols and heterogeneous trust: Technical report. *CoRR*, abs/1412.3136, 2014.

[53] Marko Vukolic. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *Open Problems in Network Security - IFIP WG 11.4 International Workshop, iNetSec 2015, Zurich, Switzerland, October 29, 2015, Revised Selected Papers*, pages 112–125, 2015.

[54] Gavin Wood. Ethereum: A secure decentralized transaction ledger. `http://gavwood.com/paper.pdf`.

[55] Pieter Wuille. Block size following technological growth (bip 103). `https://github.com/bitcoin/bips/blob/master/bip-0103.mediawiki`.

Appendix

A Background on Permissioned BFT

We briefly describe one possible instantiation of the permissioned BFT protocol using PBFT [18] as an example. Roughly speaking, PBFT [18] is a partially synchronous protocol for Byzantine state machine replication.

Below we informally describe the protocol for the case when $n = 3f + 1$. It is not hard to modify the protocol for the more general case $n > 3f + 1$. In our description, we assume transactions are proposed in units called batches.

Normal-case operations. We first describe the normal-case operations of the PBFT protocol, where all messages are signed by the sender.

1. The leader of the current view proposes a tuple ("propose", v, ℓ, batch) to all nodes where v denotes the view number and ℓ denotes the sequence number.

2. When an honest node hears ("propose", v, ℓ, batch), if it has not sent a prepare message for (v, ℓ), it multicasts ("prepare", v, ℓ, batch).

3. When an honest node collects ("prepare", v, ℓ, batch) from $2f + 1$ distinct nodes for the same $(v, \ell, \mathsf{batch})$ tuple, it multicasts ("commit", v, ℓ, batch). Further, the honest node now considers $\mathsf{prepared}(v, \ell, \mathsf{batch}) := 1$.

4. When an honest node first collects ("commit", v, ℓ, batch) from $2f+1$ distinct nodes for the same $(v, \ell, \mathsf{batch})$ tuple; or when it first collects ("commited", v, ℓ, batch) from $f + 1$ distinct nodes for the same $(v, \ell, \mathsf{batch})$ tuple: the node considers $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) := 1$ and multicasts ("commited", v, ℓ, batch). Here $\mathsf{lcommitted}$ is short for "locally committed".

The normal-case protocol satisfies the following important properties:

- *Agreement.* If two honest nodes each believes that $\mathsf{prepared}(v, \ell, \mathsf{batch}) := 1$ and $\mathsf{prepared}(v, \ell, \mathsf{batch}') := 1$ respectively, then $\mathsf{batch} = \mathsf{batch}'$.

- *Liveness under an honest leader.* If the leader is honest and no honest node has timed out since start of the latest view, then any batch submitted by an honest node will be locally committed by all honest nodes in $O(1)$ atomic time steps.

- *Ample proofs of preparedness.* If at least one honest node considers $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) := 1$, then at least $f + 1$ honest node considers $\mathsf{prepared}(v, \ell, \mathsf{batch}) := 1$.

  If an honest node believes that $\mathsf{prepared}(v, \ell, \mathsf{batch}) := 1$, then it can produce $2f + 1$ signed prepare messages that led to this belief. We refer to the collection of these $2f + 1$ prepare messages a *proof-of-preparedness.*

Notice that an immediate corrolary of the agreement property is that if two honest nodes each believes that $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) := 1$ and $\mathsf{lcommitted}(v, \ell, \mathsf{batch}') := 1$ respectively, then $\mathsf{batch} = \mathsf{batch}'$. However, the normal-case operation does not guarantee, under a potentially corrupt leader, that if one honest node thinks $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) := 1$, other honest nodes will necessarily think $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) := 1$. This therefore motivates the view change protocol.

View change. The normal-case protocol alone does not guarantee liveness when the leader is corrupt. To guarantee liveness even when the leader is corrupt, a view change protocol is invoked upon timeouts. To obtain $O(n\delta)$ worst-case response time, we can make a small modification to PBFT's original exponential backoff strategy: instead, the timeouts could double every n view changes. In the partially synchronous model, when the timeout backs off to $\Theta(\delta)$ and the leader is honest, liveness ensues.

Roughly speaking, if an honest node hears $f+1$ valid view change requests for a new view v', it will echo the view change request by multicasting a view change message itself for view v'.

When the new view's leader collects $2f+1$ valid view change requests, the set of $2f+1$ valid view change requests together form a *new-view* message. The leader then proposes the new-view message to all nodes. When an honest node receives the new-view message, For every $(v, \ell, \mathsf{batch})$ with a valid proof-of-preparedness contained in the new-view message, the node acts as if it has just received a ("propose", v, ℓ, batch) message, therefore multicasts a prepare message for the tuple, and continues as in the normal-case operations.

Due to the "ample proofs of preparedness" property of the normal-case operation, the following property holds: If an honest node believes that $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) = 1$, then at least one valid proof-of-preparedness will be included in any valid new-view message. This ensures that if at least one honest node believes that $\mathsf{lcommitted}(v, \ell, \mathsf{batch}) = 1$, the tuple $(v, \ell, \mathsf{batch})$ is guaranteed to carry over to the new view, and therefore if other honest nodes locally commits $(v, \ell, \mathsf{batch}')$ in the new view, it holds that $\mathsf{batch} = \mathsf{batch}'$.

Finally, as long as the new leader is honest and no honest node has timed out yet in the new view, then liveness ensues for the new view.

We refer the reader to the PBFT paper [18] for a detailed description of the view change protocol as well as the checkpointing optimization. It is not difficult to formalize the proofs in the PBFT paper [18] and extend them to a cryptographically sound framework. Further, it is not difficult to show that the PBFT protocol realizes our strong notion of security as defined in Section 4.2.