# Enig: Player Replaceable Finality Layers with Optimal Validity

Simon Holmgaard Kamp[1], Jesper Buus Nielsen*[1], Søren Eller Thomsen[1], and Daniel Tschudi[2]

[1]Concordium Blockchain Research Center, Aarhus University, Denmark
{kamp, jbn, sethomsen}@cs.au.dk
[2]Concordium, Zurich, Switzerland
dt@concordium.com

February 23, 2022

**Abstract**

We present two new provably secure finality layers for Nakamoto style blockchains. One is for partially synchronous networks and the other is for networks with periods of synchrony. Both protocols are player replaceable and therefore enjoy protection against denial of service attacks when run with a proof-of-stake lottery to elect the parties. The finality layers are proven secure to run on top of any Nakamoto style blockchain which has a property called *finality friendliness*. Both finality layers improve on all existing provably secure finality layers in terms of communication complexity or security.

A proof-of-stake finality layer has $v$-validity if whenever it declares a block $B$ final then honest parties holding a fraction $v$ of the stake had $B$ on the longest chain. Validity is important to prevent that the finality layer finalises blocks that were not "good" according to the Nakamoto style blockchain. We prove upper bounds on the achievable validity in partially synchronous networks and networks with periods of synchrony. Both our finality layers match these upper bounds.

## 1 Introduction

Research on blockchain consensus has surged in recent years. Blockchain consensus protocol design is highly varied, but many blockchains with provable security fall into one of three design paradigms, Nakamoto style consensus (NSC), committee-based Byzantine fault tolerance (CBFT), or hybrid consensus.

**Nakamoto Style Consensus.** In the NSC paradigm a chain of blocks is being built. There is a lottery which picks a party to extend the existing chain with a new block. No one knows who won the lottery until the block is sent. This gives strong protection against denial of service attacks. Due to malicious behaviour or honest parties winning the lottery at the same time, the chain might fork into a tree. In that case there is a longest chain rule (or in general, best chain

*Partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

rule) saying at which leaf in the tree an honest party should add new blocks. It can typically be proven that if a majority of the winners are honest and the network sufficiently synchronous then there will be a constant *look back length* $L$ such that if two honest parties look back $L$ blocks from the leafs of each their best chains, then they will look at two blocks on the same path. In other words, if we let the *honest stem* be the longest chain from the genesis block which is a prefix of the best chain of all honest parties, then the honest stem is at most $L$ blocks shorter then the longest chain held by an honest party. And this honest stem is consistent over time. This means that if an honest party looks back $L$ blocks from the leaf of its current best chain, then it looks at a block which will forever be on the best chain of any honest party. We call such a block *final*. In general we call the process of determining when a block is final a *finality rule*. The *look back finality rule* was the finality rule of the original Bitcoin protocol though no explicit $L$ was given.

The two most widespread ways to implement the lottery are proof-of-work and proof-of-stake. In both cases the probability to win lottery is tied to a limited resource, computing power or stake in the blockchain, to avoid Sybil attacks. The end goal is that most of the time an honest party wins the lottery. The seminal protocol and proof techniques in [10] gave the first PoS protocol with provable security. In [14] it was shown that the original PoW based Bitcoin protocol is in fact secure under well justified lem.

NSC can in general tolerate short periods of asynchrony. They only need that a majority of the winning events are blocks won by honest parties during periods of synchrony. This will over time build a growing and consistent honest stem. However, the look back finality length $L$ can only be computed correctly under some assumptions on how long the “bad” periods are. Analysing how long $L$ need to be is ongoing work, cf. [3, 15].

**Committee-based BFT.** In the CBFT paradigm a committee is elected. It is assumed that at most a constant fraction is corrupted. The committee uses a Byzantine agreement protocol to agree on the chain of blocks one block at a time. An advantage of CBFT protocols is that blocks are immediately final, i.e., the look back length is $L = 0$.

Using Byzantine agreement per block in principle allows to run classic BFT protocols like the PBFT protocol [7]. However, one reason for the NSC design pattern of having random winners produce blocks is to protect against denial of service attacks. In protocols like PBFT the servers have to act several times as the protocol proceeds in rounds where each server speaks in each round, opening up for denial-of-service attacks. This led to the invention of PoS CBFT protocols where each party only has to send a single message and at random points in time. The first provable secure example was ALGORAND [9], which also proposed a novel design paradigm going via so-called player replaceable (PR) protocols.

A PR protocol is for $n$ parties, $\mathsf{P}_1, \ldots, \mathsf{P}_n$, out of which for instance $t < n/3$ might be Byzantine corrupted. The protocol proceeds in rounds. The role of $\mathsf{P}_i$ in round $R$ is executed by a sub-party $\mathsf{P}_i^R$. The sub-parties of $\mathsf{P}_i$ cannot share secret state. State may be passed from $\mathsf{P}_i^R$ to future $\mathsf{P}_i^{R'}$ only by public flooding messages. One can then use a given PoS lottery to assign a separate machine to execute each role $\mathsf{P}_i^R$. This gives the same level of protection against denial-of-service attacks as PoS NSC protocols.

One disadvantage of CBFT protocols is the lower corruption threshold. Typically CBFT protocols only tolerate $t < n/3$ Byzantine corruptions, and this is optimal if the protocol tolerates periods of asynchrony, see, e.g., [11]. Another disadvantage is that they typically break for good once the $t < n/3$ assumption breaks, as the Byzantine agreement might break and create a fork in the chain. NSC can tolerate corrupted majority if only there is honest majority over long enough periods. There is a huge difference between having honest majority over long enough periods and always having honest super majority. A last disadvantage is higher communication.

In CBFT protocol each committee member floods at least one message per block. In NSC only the single block itself is flooded.

**Finality Layers.** A third approach is a hybrid approach with an underlying NSC consensus on top of which a CBFT finality layer is run. Real life examples of this approach is Grandpa used by the Polkadot blockchain [18], Casper [4] which is planned to be used with Ethereum 2.0, and Afgjort used by the Concordium blockchain [11].

The idea of the hybrid approach is to use a CBFT Byzantine agreement to agree on what the honest stem is right now and then *declare* it final, i.e., the best chain rule of the NSC disregards chains not including the last block having been declared final. The goal is to reap the best of both worlds. In NSC the look back parameter $L$ can be huge and sometimes hard to compute correctly for real world networks. However, in good weather conditions of the network—high synchrony and high honesty—the honest stem of a NSC will be identical to the longest chain most of the time. In this scenario the use of a finality layer would lead to blocks becoming provably final much faster.

There are also advantages of the hybrid approach over pure CBFT. As an example, if the blocks of the NSC has size $B$ the flooding complexity will be in the order of $B$. The BFT finality layer need not agree on the block, it is enough to agree on a hash of the block. The flooding complexity can therefore be in the order of $n\kappa$, where $\kappa$ is the security parameter. If the finality layer is run every $c$ blocks this gives a flooding complexity per block in the order of $n\kappa/c + B$. For large $B$ or large $c$ this means that the communication can be in the order of that of the NSC itself.

It is also conceivable that the system can be designed such that the NSC is live and secure when there is honest majority and the finality layer kicks in when there is honest super majority. As as a motivating example of this consider a model with $n = 3t + 1$ parties with at most $t$ malicious corruptions, but where some honest parties might be offline in some periods. If we assume that less than $t$ honest parties are offline, then there is still honest majority among the online parties and the NSC could still be sure. If the finality layer is designed such that if honest parties are offline it loses liveness, but not safety, and regains liveness once honest participation is high again, one would have an overall design where the NSC chain is always live and where fast finality "kicks in" when honest participation is high. This is meant as a motivating example, and we do not present a formal model of this or prove this result in the paper. It is an interesting venue for future work to design and prove "best possible" finality layers which gives finality when possible and does not interfere with the NSC when finality is not possible.

Security analysis of NSC protocols and CBFT protocols is fairly advanced. Security analysis of finality layers less so. One challenge in analysing hybrid protocols is that the finality layer steers the underlying NSC by forcing the hand of the best chain rule. On the other hand, the input of the finality layer is the tree grown by the NSC. One reason that the security analysis of the interaction between NSC consensus and finality layers is poorly understood could be the lack of good methodologies for handling this cycling dependency in the past. Analysing a NSC blockchain in isolation is hard enough it itself. The same for a finality layer. Analysing the combined hybrid design monolithically is daunting. To remedy state-of-the-art the paper [11] introduced a formal framework for analysing finality layers. They define a notion of *finality friendly NSC* which is a property of a NSC protocol itself. Then they design a finality layer and prove that it can be securely run on top of any finality friendly NSC. In [11] the authors argue common NSC protocol like Bitcoin and Ouroboros Praos [10] are finality friendly, showing that the methodology is meaningful. We continue this line of work of provably secure finality layers.

**Our Contributions.** We introduce two finality layers tolerating $t < n/3$ malicious corruptions. One works in the partially synchronous model with an unknown upper bound on the network delay and finalises blocks with 1-validity. A finality layer has $v$-validity if whenever a chain is declared final, then at least $v$ honest parties have, or had, this chain as part of their best chain. The other finality layer works in the partially synchronous model with a known upper bound $\Delta_{\texttt{net}}$ that sometimes holds and has $(t + 1)$-validity. Both protocols are safe during periods of asynchrony and are additionally live during synchronous periods. Our motivation for wanting this is that finality layers are run on top of NSC blockchains which already tolerate periods of asynchrony. If the finality layer did not have this property it would degrade the security of the combined protocol.

Both finality layers are guaranteed to finalise the entire honest stem. Our motivation for wanting this is that the ideal finality layer would discover exactly what the current common prefix of the honest parties is, and finalise exactly that (which would give $(2t + 1)$-validity) to stop unnecessary rollbacks. We show that this is impossible to obtain in a partially synchronous model or when periods of asynchrony must be tolerated. We then give protocols achieving the maximal possible validity in these two network models.

Our protocols are player-replaceable protocols and if run during periods of synchrony each committee member floods an expected constant number of messages.

Compared to [11] we define a weaker notion of finality friendly NSC and prove security under this definition. By weaker we mean that if a NSC is finalisation friendly in the sense of [11] it is also finalisation friendly in our weaker sense. We discuss the weakening further in Section 3. Furthermore, both of our protocols guarantee to finalise the entire *honest stem* in each iteration they are run. This is an improvement over Afgjort which, if there is a fork in the underlying NSC, needs to run for a logarithmic number of iterations in the fork depth to finalise the honest stem. We also improve communication complexity over Afgjort. In Afgjort each party at some point floods an index vector of length $n$ making the expected flooding communication complexity in bits $O(n(\kappa + n))$. In our protocols each party floods only messages of size $O(\kappa)$, giving expected flooding communication complexity in bits $O(n\kappa)$. Compared to Afgjort we also give the first provably secure PR finality layer. The protocol for weak core set selection in Afgjort is not player replaceable making the overall protocol not PR.

Compared to Casper [4] and Grandpa [18] we show well-defined security properties of the finality layer based on a well-defined property of the NSC itself. This is not the case for [4, 18]. As an example, in [18] the authors prove that a NSC blockchain run with the Grandpa finality layer has liveness when modelling the underlying NSC as an “eventually consistent oracle”, which produces trees with a growing honest stem. They show that on top of such a NSC the finality layer is live, i.e., it will advance the last final block. But this does not show that the NSC protocol combined with the finality layer will have liveness. If somehow the steering induced by the finality layer kills the liveness of the NSC, this would not show up in the model of [18] as the NSC is modelled using the eventual consistency oracle which *assumes liveness*. So the proof of the liveness of the overall construction is of the form that it shows liveness under the assumption of liveness. This is not satisfactory from a point of view of provable security. This is not a mute point. The Grandpa protocol in [18] is intricate and interacts with the NSC protocol in several ways where parties have different notions of what blocks are final and where other blocks than final blocks are used to steer the best chain rule. There is nothing in [18] which indicates that it is ruled out that one can construct NSC protocols which are live but which loses liveness when combined with Grandpa. Similarly, Casper [4] does not come with well defined properties of the NSC which would make it safe to run Casper on top of the NSC. Maybe as a consequence of this, the unpublished work [5] which applies Casper on top of a concrete NSC blockchain gives a monolithic and very substantial proof of the overall construction. Should either the NSC change or the finality layer, the proof would *a priori* have to be redone. And other NSC blockchains wanting to apply Casper have no specification of properties to implement for this to be secure. The paper also seems underspecified in term of desirable security properties. It for instance proves "plausible liveness" which essentially says the protocol can not reach a possible deadlock state where there does not exist some sequence of future events which could recover from the deadlock. It does not prove anything about the probability of such a recovery will take place. Casper also does not address the "updated" property that the finality layer is keeping up with the NSC.

Finally our protocols seem to be simpler than Afgjort, Casper and Grandpa and have more intuitive security proofs.

**Paper Structure.** In Section 2, we introduce relevant network models, background knowledge on PR protocols, proof-of-stake lotteries, and define how justifications are run light. In Section 3, we present our notion of a finality friendly NSC, our functionality for a finality layer and discuss relevant properties of finality layers. In Section 4, we present our bounds for finality layers with validity when finalising the entire honest stem. In Section 5, we give an overview of our two finality protocols for the respective network models. Defining our finality layers we make use of the protocol wBA that again relies on YABBA. We define and prove these secure in Sections 6 and 7 before we finally prove the two finality layers secure in Section 8. Lastly, in Section 9 we sketch how our protocols can be proven secure in a composable setting.

## 2 Preliminaries

### 2.1 Notation

We consider protocols for $n$ parties, $\mathcal{P} = \{\mathsf{P}_1, \dots, \mathsf{P}_n\}$, out of which $t < n/3$ might be Byzantine corrupted. We let $\mathcal{H}$ denote the set of parties that have not been corrupted. We let the infix notation ":=" denote variable assignment. We let $\mathcal{B}$ denote the type of all blocks. For two blocks, $b$ and $b'$ we write $b \preceq b'$ when $b$ is in the chain of $b'$, writing $b \prec b'$ when $b$ is in the prefix of $b'$ but $b = b'$.

### 2.2 Network Model

In the blockchain setting, parties typically have access to a *flooding network* allowing them to exchange messages. A party $\mathsf{P}$ can send a message $m$ and then $m$ will eventually arrive at all other parties. We will interchangeably use the terms that $\mathsf{P}$ sends $m$ and $\mathsf{P}$ floods $m$. We assume that the network is authenticated, i.e., from a flooded message we can see who sent it. Flooding networks typically do not guarantee that messages arrive within a known delay, i.e., they are not synchronous. We consider two variants of a *partially synchronous* flooding network [12]. The first variant is dubbed *strong periods of synchrony* [8] (SPS).

**Definition 1** (partial synchrony with strong periods of synchrony [8])**.** A network $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ is *partially synchronous with strong periods of synchrony* if there exists a bound $\Delta_{\mathtt{net}}$ such that

- For any period of synchrony $[\tau_0, \tau_1]$: Any message observed, i.e. sent or received, by an honest party at time $\tau \leq \tau_1$ is observed by all honest parties before $\max(\tau_0, \tau + \Delta_{\mathtt{net}})$.
- The network delay bound $\Delta_{\mathtt{net}}$ is known to honest parties and protocols may depend on it. However, honest parties do not know when periods of synchrony are happening.

The SPS model models that a flooding network often is in a good shape and ensures delivery within some known delay. However, it may happen due to network congestion, physical damage on the network, or other unforeseen events, that periods with no known delivery guarantee occur. Protocols designed for this model should maintain their safety properties even in such periods, whereas it is sufficient to prove liveness in a period of synchrony.[1]

The other variant is partial synchrony with *hidden bounded delay* (HBD).

**Definition 2** (partial synchrony with hidden delay bound [12]). A network $\mathcal{N}_{\mathtt{HBD}}^{\Delta_{\mathtt{net}}}$ is *partially synchronous with hidden bounded delay* $\Delta_{\mathtt{net}}$ if the following holds

- A message sent by an honest party at time $\tau$ is received by all honest parties before time $\tau + \Delta_{\mathtt{net}}$
- A message received by an honest party at time $\tau$ (possibly sent by the adversary) is received by all honest parties before time $\tau + \Delta_{\mathtt{net}}$.
- The delay $\Delta_{\mathtt{net}}$ is not known by honest parties. In particular, a protocol cannot depend on $\Delta_{\mathtt{net}}$.

If the flooding network eventually delivers all messages, then protocols proven secure in this model will terminate. This is an advantage if the known $\Delta_{\mathtt{net}}$ of SPS if often too small. Furthermore, their execution time is a function of the actual delay on the network, which is an advantage if the known $\Delta_{\mathtt{net}}$ of SPS if often too large.

In Section 4, we show that the achievable amount of validity for finality layers clearly separates the two network models, and in Section 5, we present two finality layers (one designed for each model), with optimal validity in their respective model.

## 2.3 Proof-of-Stake Lotteries

We need a non-trivial leader election protocol, i.e., one which elects a leader which is honest with at least a constant, positive probability. We assume that parties have access to a proof-of-stake lottery which for each lottery identifier $\mathsf{lid}$ allows a party $\mathsf{P}$ to compute a lottery ticket $\mathsf{ticket}_{\mathsf{lid},\mathsf{P}}$ with some weight that can be sent to the other parties. The leader $\mathsf{P}_\ell$ is *defined* to be the party with the largest ticket. Note that discovering who is the leader is a separate problem. Following [11] we use a very simplistic model of this, where we assume that if we have a committee $\mathsf{P}_1^{\mathsf{lid}}, \ldots, \mathsf{P}_n^{\mathsf{lid}}$ with $n-t$ honest parties for $t$ and we let each $\mathsf{P}_i^{\mathsf{lid}}$ compute $\mathsf{ticket}_{\mathsf{lid},\mathsf{P}_i}$ and define $\ell = \operatorname{argmax}_i \mathsf{ticket}_{\mathsf{lid},\mathsf{P}_i}$, then there is a probability ${}^{n-t}/_{t}$ for each $\mathsf{lid}$ independently that $\mathsf{P}_\ell^{\mathsf{lid}}$ is honest. We can implement this in the UC model using for instance the lottery from [10], but most reasonable PoS lotteries should allow such non-trivial leader election. See Appendix A for a discussion.

## 2.4 Player-Replaceable Protocols

As discussed above, all our protocols are *player-replaceable protocol*. Because we assume a model with periods of asynchrony we need the following slightly more restrictive class of PR protocols than [9], which we call *symmetric PR* (SPR). In Appendix B we discuss why we need this more restrictive class. In each round there are ground population of $N$ parties $\mathsf{Q}_1^R, \ldots, \mathsf{Q}_N^R$, which we call roles. Roles can as usual be honest or corrupted (malicious). But now in addition some roles might be crashed. A crashed role $\mathsf{Q}_i^R$ will simply not execute its role. A role is *live* if not crashed. A role is *honest* if live and not corrupted. The model comes with two thresholds $t$ and $h$, where in our case $h = 2t+1$. Let $n = t + h$. In each round the adversary is allowed to crash roles and corrupt roles, as long as the number of live roles is at most $n$, the number of corrupted

[1]This model fits nicely with known NSC protocols, which require that the block-production frequency is set such that most of the time one block propagates to all parties before the next one is created. The expected block-production time seems as a natural choice for $\Delta_{\mathtt{net}}$ in this model, as it by assumption of the underlying NSC will hold "most of the time", or the NSC would be insecure itself.

roles is less than $t$, and the number of honest parties is at least $h$. We call a protocol secure in this model a *symmetric* PR protocol, as it is only guaranteed that in each round there exist $n = 3t + 1$ parties $\mathsf{P}_1^R, \ldots, \mathsf{P}_n^R$ for which it is guaranteed that $2t + 1$ are honest, but the honest parties might not agree who these parties are, it can be any subset of the ground population. Note, however, that in a SPR protocol an honest party can wait for messages from $h = n - t$ roles without deadlocking as the $h$ honest parties will each send their message. Furthermore, if two honest parties both wait for $h = n - t = 2t + 1$ messages, then because there are at most $n$ live roles, they will have heard from at least $(n - t) - t = t + 1$ common roles. Since there are at most $t$ corrupted roles, they will in turn have heard from at least one common honest role. These will be the central properties exploited when proving liveness and safety of our SPR protocols.

Note that if roles are assigned using PoS lotteries, then PR protocols immediately give strong protection against adaptive corruption in models with *atomic message delivery*. Atomic message delivery says that if a party floods messages then they are eventually delivered even if the party becomes corrupted before the messages were delivered at the first honest party. A stronger model would be to assume that messages not delivered to honest parties yet can be "taken back" if the sender becomes adaptively corrupted. It has been shown in [16, 1] that this model has strong impossibility results and indeed most blockchain protocols are not secure in this model. In a Nakamoto style consensus the adversary can for instance always corrupt the block winner and take back the block. It was argued in [13] that adaptive corruption with atomic message delivery is a realistic model of adaptive corruption in real world networks and that it avoids the impossibility results.

## 2.5 Justifications

We use the notion of *justifications* from Afgjort [11] with some simplifying modifications. Intuitively a message $m$ is *justified* if the party receiving it has also received a set of messages that would make an honest sender produce $m$. Consider a protocol for a flooding network. Messages are authenticated, so we can represent a received message as $(\mathsf{S}, m)$, where $\mathsf{S}$ is the sender. Let $\mathsf{Msgs}(\mathsf{P})^\tau$ be the set of messages $(\mathsf{S}, m)$ received by $\mathsf{P}$ at time $\tau$. Since we assume that all messages eventually propagate we know that it holds for all honest parties $\mathsf{P}_i$ and $\mathsf{P}_j$ and all times $\tau_1$ that there exists $\tau_2$ such that if the protocol reaches time $\tau_2$ then $\mathsf{Msgs}(\mathsf{P}_i)^{\tau_1} \subseteq \mathsf{Msgs}(\mathsf{P}_j)^{\tau_2}$.

**Definition 3** (Justification predicates)**.** A justification predicate for a protocol $\Pi$ is an efficiently computable predicate $J$ which takes as input a sender-message pair $(\mathsf{S}, m)$ and a set of messages $\mathsf{Msgs}$ and outputs $\perp$ or $\top$, where we think of $\top$ meaning that it is allowed for $\mathsf{S}$ to send $m$ if it received the messages in $\mathsf{Msgs}$. We require that $J$ is monotone, i.e., if $\mathsf{Msgs} \subset \mathsf{Msgs}'$ and $J((\mathsf{S}, m), \mathsf{Msgs}) = \top$ then $J((\mathsf{S}, m), \mathsf{Msgs}') = \top$. We say that $\mathsf{Msgs}$ is a justification for $(\mathsf{S}, m)$ if $J((\mathsf{S}, m), \mathsf{Msgs}) = \top$. If $J((\mathsf{S}, m), \emptyset) = \top$ we call $m$ *self-justifying.*

When specifying a justification predicate $J$ for a protocol it can be specified by a set of justification predicates $J_1, \ldots, J_n$ handling messages from different sub-protocol and rounds. Then $J((\mathsf{S}, m), \mathsf{Msgs}) = \bigvee_i J_i((\mathsf{S}, m), \mathsf{Msgs})$.

We say that a protocol is justifying if it has a justification predicate $J$ and each message that can be sent by an honest party is justified by $J$. We say that a justifying protocol for a flooding network is being run *light* if it has the following behaviour. When flooding a message $(\mathsf{S}, m)$ in a justifying protocol the justification is not sent along, the sender $\mathsf{S}$ only sends $m$. When receiving a message $(\mathsf{S}, m)$ the receiver $\mathsf{R}$ will buffer $(\mathsf{S}, m)$ until $J((\mathsf{S}, m), \mathsf{Msgs}(\mathsf{R})) = \top$. Only then does it add $(\mathsf{S}, m)$ to $\mathsf{Msgs}(\mathsf{R})$ and start processing $(\mathsf{S}, m)$. This will eventually happen as all flooded message eventually propagate. Note that the size of the justifications does not affect the communication complexity, as only the messages themselves are being flooded.

All our protocols will be justifying protocols and will tacitly be run light. All of our justifications are monotone and efficiently computable. In cases where this is not clear from the definition of a justification we provide a proof of efficient computability.

## 3 Abstract Model of Blockchains and Finality Layers

We wish to build our finality layers such that they can be run in combination with as many existing blockchains as possible. In Afgjort [11] the notion of a *finalization friendly blockchain* was introduced as a functionality, $\mathcal{F}_{\mathsf{Tree}}$. The functionality maintains a tree $\texttt{Tree}_i$ for each party $\mathsf{P}_i \in \mathcal{P}$. It also provides an interface to the adversary to grow these trees subject to certain constraints, an interface for parties to select final blocks, and an interface for parties to see their current tree.

The tree in [11] respects canonical blockchain properties such as *chain growth* and *common prefix* which ensures that all trees grow at a certain rate and that the current best path in all honest trees have a common prefix that are only a constant number of steps from the longest path known by any honest party. However, also more exotic non-standard properties such as *bounded path growth* and *dishonest chain growth* which limits how fast both honest and dishonest path can grow within the functionality. These constraints are necessary in Afgjort, in order to ensure that their finality process will eventually terminate and stay up to date.

In this work, we take a similar approach, defining a finalization friendly blockchain as a functionality $\mathcal{F}_{\mathsf{Tree}}$. Our tree-functionality is however, different than the one presented in Afgjort on two key-points:

1. The finality protocols that we present here does not depend on the underlying properties of the NSC to function. Such properties might however be important for the combined protocol. Therefore, we parametrise our tree functionality functionality by a set of trace-properties $\mathbb{P}$ that the underlying functionality enforces. Similarly, we also parametrise the finality-functionality $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$ by such a set of trace properties. This allows us to prove for a restricted class of properties that any such property the underlying finality friendly blockchain $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ has, is preserved when $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ is used as a building block of our finality protocol. Additionally, our functionality layer brings additional properties such that what the functionality we really implement is $\mathcal{F}^{\mathbb{P}'}_{\mathsf{FinTree}}$ for a $\mathbb{P}' \supseteq \mathbb{P}$. The specifics of $\mathbb{P}'$ will be discussed later in this section and in Section 9.

2. We require that the command for setting final blocks are only used by parties in a way such that all blocks that declared final, by any honest party must be on a chain. At first this seems like that we are strengthening the functionality, but in fact this makes the functionality significantly easier to implement as we allow the properties in $\mathbb{P}$ to be broken in case this restriction is violated.

### 3.1 The Finalization Friendly Tree

Below we present our tree functionality, $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$. Before presenting the minimalistic tree functionality we define the *honest tree*, as the union of all of the honest parties' trees:

$$\texttt{HonestTree} \coloneqq \bigcup_{P_i \in \mathcal{P}} \texttt{Tree}_i.$$

When we define our functionality below we let $\texttt{Chain}_T(B)$ denote the function returns a path of blocks in tree $T$ ending in block $B$. The functionality ensures that the users only input final blocks on a chain. If this is not the case the functionality sets the Boolean $\texttt{Exploded}$ to true, after which it will not maintain any other properties. Note that the functionality only checks that the properties in $\mathbb{P}$ are preserved when receiving *adversarial inputs*. This implies that user's interactions are *not* restricted by these, and any protocol building on top can freely use these.

**Functionality $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$**

For any $\mathsf{P}_i \in \mathcal{P}$ the functionality maintains the variables $\texttt{Tree}_i$, $\texttt{Pos}_i$, $\texttt{LastFinal}_i$, and $\texttt{Finals}_i$. It additionally maintains a global flag $\texttt{Exploded}$ which initially is set to $\perp$, and a global list $\texttt{Trace}$, initially set to empty.

**Initialisation:** For any $\mathsf{P}_i \in \mathcal{P}$ let $\texttt{Tree}_i := ((V_i := \{G\}, E_i := \emptyset), r_i := G)$, set $\texttt{Pos}_i := G$, and $\texttt{LastFinal}_i := G$.

**Get Tree:** On input GetTree from party $\mathsf{P}_i \in \mathcal{P}$ the functionality returns $(\texttt{Tree}_i, \texttt{Pos}_i, \texttt{LastFinal}_i)$.

**Set Final:** On input $(\textsc{SetFinal}, R)$ from party $\mathsf{P}_i \in \mathcal{P}$ the functionality sets $\texttt{LastFinal}_i := R$ and $\texttt{Finals}_i := \texttt{Finals} \cup \{R\}$. Furthermore, it updates $\texttt{Exploded} := \top$ *iff* $\bigwedge_{\mathsf{P}_j \in \mathcal{P}} \bigwedge_{F \in \texttt{Finals}_j} R \in \texttt{Chain}_{\texttt{HonestTree}}(F) \vee F \in \texttt{Chain}_{\texttt{HonestTree}}(R)$ or $R \notin \texttt{Tree}_i$. Finally, if $R \notin \texttt{Chain}_{\texttt{Tree}_i}(\texttt{Pos}_i)$ the position is updated, $\texttt{Pos}_i := R$.

**Add Node:** On input $(\textsc{AddNode}, \mathsf{P}_i, B, p)$ from the adversary, if $B \notin V_i$ and $\texttt{HonestTree}$ remains a tree after adding $B$ as a child of $p$ in $\texttt{Tree}_i$ then set $V_i := V_i \cup \{B\}$, $E_i := E_i \cup \{(p, B)\}$.

**Set Position:** On input $(\textsc{SetPosition}, \mathsf{P}_i, B)$ from the adversary then the position of party $\mathsf{P}_i$ is updated, $\texttt{Pos}_i := B$.

On input $i$ from the adversary if $\texttt{Exploded} \vee \bigwedge_{P \in \mathbb{P}} P(\texttt{Trace} :: i)$, then the input is processed and $\texttt{Trace} := \texttt{Trace} :: i$. Otherwise $i$ is ignored.

**Desirable Properties of $\mathcal{F}_{\mathsf{Tree}}$.** The functionality might at first seem unable to capture interesting properties as only properties that can be evaluated on concrete traces can be enforced. Standard blockchain properties such as *chain-growth*, *common-prefix* and *chain-quality* are probabilistic and cannot be evaluated on a single trace. These guarantees can however be translated into properties that hold for a concrete trace with overwhelming probability.

One property that we need in order to implement our finality layers is that updates of the position *respects* the last final block. That is for any input $(\textsc{SetPosition}, \mathsf{P}_i, B)$ it must be that $B \in \texttt{Tree}_i$ and $\texttt{LastFinal}_i \in \texttt{Chain}(B)$. We call this property $P_{\text{FinalityRespecting}}$ and only consider trees $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ where $P_{\text{FinalityRespecting}} \in \mathbb{P}$.

## 3.2 The Finalised Tree

Similarly, we describe a functionality for the finalised tree, $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$, which also respects a set of properties. The main difference between $\mathcal{F}_{\mathsf{Tree}}$ and $\mathcal{F}_{\mathsf{FinTree}}$ is that in $\mathcal{F}_{\mathsf{FinTree}}$ it is the adversary that decides which nodes will be final – subject to restrictions.

**Functionality $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$**

For any $\mathsf{P}_i \in \mathcal{P}$ the functionality maintains the variables $\mathtt{Tree}_i$, $\mathtt{Pos}_i$, and $\mathtt{LastFinal}_i$. It additionally maintains a global initially empty list $\mathtt{Trace}$.

**Initialisation:** For any $\mathsf{P}_i \in \mathcal{P}$ let $\mathtt{Tree}_i \coloneqq ((V_i \coloneqq \{G\}, E_i \coloneqq \emptyset), r_i \coloneqq G)$, set $\mathtt{Pos}_i \coloneqq G$, and $\mathtt{LastFinal}_i \coloneqq G$.

**Get Tree:** On input GETTREE from party $\mathsf{P}_i \in \mathcal{P}$ the functionality leaks $(\mathtt{Tree}_i, \mathtt{Pos}_i, \mathtt{LastFinal}_i)$ to the adversary before returning it to $P_i$.

**Set Final:** On input $(\textsc{SetFinal}, \mathsf{P}_i, R)$ from the adversary the functionality sets $\mathtt{LastFinal}_i \coloneqq R$. If $R \notin \mathtt{Chain}_{\mathtt{Tree}_i}(\mathtt{Pos}_i)$ the position is updated, $\mathtt{Pos}_i \coloneqq R$.

**Add Node:** On input $(\textsc{AddNode}, \mathsf{P}_i, B, p)$ from the adversary, if $B \notin V_i$ and $\mathtt{HonestTree}$ remains a tree after adding $B$ as a child of $p$ in $\mathtt{Tree}_i$ then set $V_i \coloneqq V_i \cup \{B\}$, $E_i \coloneqq E_i \cup \{(p, B)\}$ and if $p = \mathtt{Pos}_i$ then set $\mathtt{Pos}_i \coloneqq B$.

**Set Position:** On input $(\textsc{SetPosition}, \mathsf{P}_i, B)$ from the adversary then the position of party $\mathsf{P}_i$ is updated, $\mathtt{Pos}_i \coloneqq B$.

On input $i$ from the adversary if $\bigwedge_{P \in \mathbb{P}} P(\mathtt{Trace} :: i)$, then the input is processed and $\mathtt{Trace} \coloneqq \mathtt{Trace} :: i$. Otherwise $i$ is ignored.

**Desirable Properties of $\mathcal{F}_{\mathsf{FinTree}}$.** The job of a finality layer is to *detect* what honest parties already agree upon and ensure that this becomes visibly *final.* That is: we want that final inputs can only extend and not contradict previous final inputs, and that what an honest party considers the best chain at all times should respect all final inputs. Furthermore, we want that two party cannot consider contradicting blocks final. This can be summarized as that all final blocks (across different parties) should form a chain.

To describe exactly what a finality layer should *detect* we define the notion of "everything that honest parties agree upon" as the *honest stem* of a tree:

$$\mathtt{HonestStem} \coloneqq \bigcap_{\mathsf{P}_i \in \mathcal{H}} \mathtt{Chain}(\mathtt{Pos}_i).$$

Ideally, a finality layer would then preserve any *good* property the underlying tree provides, and each time the finality layer is run it should finalise the entire honest stem for all parties.

As the messages of parties are delayed by a parameter of the network model, we will not try to agree on the honest stem at a specific point in time, instead we define a more relaxed notion of an honest stem which is the intersection of any combination of the positions of all honest parties within a $\Delta_{\mathtt{net}}$ long interval.

**Definition 4.** [$(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$] For a party $\mathsf{P}_i$, time $\tau$, and network delay $\Delta_{\mathtt{net}}$ we define $\mathtt{RecentPositions}^{\mathsf{P}_i}_{(\tau, \Delta_{\mathtt{net}})}$ as the set of positions $\mathtt{Pos}_i$ that were kept by $\mathcal{F}_{\mathsf{Tree}}$ in the interval $[\tau, \tau + \Delta_{\mathtt{net}}]$. We say that a chain is a $(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$ if it is the intersection of the positions of a tuple in $\Pi_{\mathsf{P}_i \in \mathcal{H}} \mathtt{RecentPositions}^{\mathsf{P}_i}_{(\tau, \Delta_{\mathtt{net}})}$.

Ensuring that the entire honest stem is finalised is however not enough. We would also like that what is finalised has been in the best chain of an honest party. [18] defines a notion of *recent validity*: any time a block, $b$, becomes final at an honest party at time $\tau$ then at least one honest party had this block in their best chain no more than a bounded time $T$ ago. [11] defines the similar restriction $(k)$-*support*: any time a block, $b$, becomes final at an honest party then at least $k$ honest parties previously had $b$ in their best chain at the height being finalised.

Disregarding time and number of parties these restrictions are morally the same. Furthermore, having a concrete bound on the time it takes to terminate and $(k)$-support implies recent validity. In that vein we choose to concentrate on the former which we define below.

**Definition 5** ($k$-validity)**.** Any time a block becomes final at an honest party at time $\tau$ then there is at least $k$ honest parties that had this block in their chain at a time earlier than $\tau$.

## 4 Validity Bounds When Finalising An Honest Stem

We show that the validity lower bounds achieved by our protocols are optimal when composed with $\mathcal{F}_{\mathsf{Tree}}$ by providing matching upper bounds. The basic $\mathcal{F}_{\mathsf{Tree}}$ imposes almost no restrictions on how an adversary interacts with the tree, so to justify that the results apply not only in this setting but also to a NSC we limit the adversary to adhere to a set of rules. The rules informally model that parties follow a simplified Bitcoin chain rule when choosing their position and that blocks propagate as specified by the network model.

**Chain rule:** Positions of honest parties can only be updated to longer chains.

**Block propagation:** Parties can send the blocks in their tree to other parties. When an honest party $\mathsf{P}$ receives a block $b$, $\mathsf{P}$ will update its position to $b$ if $\mathsf{P}$'s previous position is a shorter chain than $b$.

These restrictions on the adversary's abilities can be specified as trace properties. If the validity upper bounds holds for a tree with these properties, then it of course also holds for a tree without these properties as the adversary can just choose to follow the special rules.

In the following proofs we do not need to consider protocols finalising a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem` and terminating before $\tau$. We note that this is no serious restriction. If a party can finalise a block $B$ at some time $\tau' < \tau$, then the adversary could change the position of all honest parties to a successor of $B$ at some time $\tau''$ with $\tau' < \tau'' < \tau$, in which case $B$ is not a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem`.

We first show that $(n-2t)$-validity is maximal for finality layers that are secure in the $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ model and finalise an honest stem in periods of synchrony with probability 1.

**Theorem 1.** *If $\pi$ is a protocol implementing a finality layer tolerating $t$ corruptions in the $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ model, and $\pi$ finalises a $(\tau, \Delta_{\mathtt{net}})$-*`HonestStem`* in polynomial time when a round is contained in a strong period of synchrony, then the finalised blocks can have at most* $(\max(1 + (n \bmod 2), n - 2t))$*-validity.*

**Proof.** If $n = t$ there is nothing to show. So W.L.O.G. assume $n < t$ and that no honest party terminates before $\tau$. Partition the parties into sets $Q_1$, $Q_2$, and $Q_{\mathsf{P}}$ where $|Q_{\mathsf{P}}| = \max(1 + (n \bmod 2), n - 2t)$. Let $\mathsf{P} \in Q_{\mathsf{P}}$ be an honest party and let the remaining sets have equal size, i.e. $|Q_1| = |Q_2| \leq t$. Assume that at time $\tau_0 < \tau$ all parties agree on the same block $A$.

We now describe two worlds which are indistinguishable from the perspective of $\mathsf{P}$. In world 1 the network is in a period of synchrony and thus $\pi$ must finalise an honest stem, while in world 2 the network is not in a period of synchrony and messages can be delayed indefinitely. The adversary will simultaneously at time $\tau_0$ change the position of $Q_{\mathsf{P}}$ and $Q_2$ to a block $B \succ A$. and then:

**In world 1** corrupt $Q_1$ and let them stay silent.

**In world 2** corrupt $Q_2$ but let them act honestly, delay all messages to and from $Q_1$, and deliver all messages between parties in $Q_2 \cup Q_{\mathsf{P}}$ as if in a strong period of synchrony.

| Position | $A$ | $B$ |
|---|---|---|
| World 1 | $Q_1$ (corrupt, silent) | $Q_{\mathsf{P}}$, $Q_2$ |
| World 2 | $Q_1$ (isolated) | $Q_{\mathsf{P}}$, $Q_2$ (corrupt) |

Figure 1: The position and state of each partition in world 1 and 2.

As the parties in $Q_1$ are corrupted in world 1, the remaining parties must be able to terminate without hearing from them due to the network being in period of synchrony. Specifically $\mathsf{P}$ must terminate with a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem` at some time $\tau'$. If in world 2 the adversary delays all messages to and from parties in $Q_1$ until $\tau'$ while scheduling the messages between the remaining parties as in the execution in world 1, then $\mathsf{P}$ is unable to distinguish the two worlds (see Figure 1)at time $\tau'$ and must also terminate at time $\tau'$ in world 2 with the same output distribution as in world 1. Observe that:

**In world 1** all honest parties have $B$ as their best block at time $\tau_0$ and will not change back to $A$ by rule 1. So in the interval $[\tau, \tau + \Delta_{\mathtt{net}}]$ finalising $A$ would violate the guarantee of finalising a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem`. I.e. the output of $\mathsf{P}$ is $B$.

**In world 2** $\mathsf{P}$ terminates at $\tau'$ with some output $f$, and the validity of the output is defined as the number of honest parties who at $\tau'$ or earlier had $f$ in the prefix of their chain. Finalising $B$ results in exactly $(|Q_{\mathsf{P}} \cap \mathcal{H}|)$-validity.

We conclude that $\pi$ cannot get more than $(\max(1+(n \mod 2), n-2t))$-validity in general. □

It is fairly easy to prove analogously that protocols in the $\mathcal{N}_{\mathtt{HBD}}^{\Delta_{\mathtt{net}}}$ model cannot guarantee more than $(n-2t)$-validity if it finalises a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem`. But using the fact that $\Delta_{\mathtt{net}}$ is unknown we can prove a stronger bound.

**Theorem 2.** *If a protocol $\pi$ tolerates $t$ corruptions, terminates in polynomial time in the $\mathcal{N}_{\mathtt{HBD}}^{\Delta_{\mathtt{net}}}$ model, and guarantees finalisation of a $(\tau, \Delta_{\mathtt{net}})$-`HonestStem` in polynomial time, then the finalised blocks can at most have guaranteed $(\max(1 + (n \mod 3), n - 3t))$-validity.*

**Proof.** Partition the parties into sets $Q_1$, $Q_2$, $Q_3$, and $Q_{\mathsf{P}}$ where $|Q_{\mathsf{P}}| = \max(1+(n \mod 3), n-3t)$. Let the remaining sets have equal size, i.e. $|Q_1| = |Q_2| = |Q_3| \leq t$. We now describe three worlds which are indistinguishable from the perspective of an honest party $\mathsf{P} \in Q_{\mathsf{P}}$ until time $\tau_{output}$, which we define based on the execution in world 1. In world 1 the network parameter $\Delta_{\mathtt{net}}$ is set to $\Delta_1$ and in the two other worlds it will be chosen as a function of $\Delta_1$. Assume that all parties agree on some block $A$ at time $\tau_{partition} = \tau - \Delta_1$, and let $B$ and $C$ be different children of $A$.

The adversarial strategy will be to exploit a combination of a small fork and a partition to force low validity when polynomial time termination and finalisation of an honest stem must be observed. In all worlds the adversary will simultaneously at time $\tau_{partition}$ change the position of the parties in $Q_{\mathsf{P}}$ and $Q_3$ to $B$, and the parties in $Q_2$ to $C$, and then:

**In world 1** corrupt $Q_1$ and let them remain silent.

**In world 2** corrupt $Q_2$ and change the position of $Q_1$ to $B$.

**In world 3** corrupt $Q_3$ and change the position of $Q_1$ to $C$.

Observe that in world 1, the honest parties must terminate $\pi$ in polynomial time. Let $\tau_{output}$ be an upper bound on when $\mathsf{P}$ must terminate. We can then define the message delay bound of world 2 and 3, $\Delta_{2,3} = \tau_{output} - \tau_{partition}$. With this delay bound the adversary is allowed to delay all messages to and from $Q_1$ in world 2 and 3 in the interval $[\tau_{partition}, \tau_{output}]$. Additionally, as $\Delta_1 \leq \Delta_{2,3}$, the adversary can schedule messages in world 2 and 3, exactly as in world 1 for all parties except those in $Q_1$. It now follows that the view of the parties $Q_2 \cup Q_3 \cup Q_{\mathsf{P}}$ is identical in worlds 1, 2, and 3 at every time-step up to $\tau_{output}$. As $\mathsf{P}$ has to output by this time in world 1, it must also output in the other worlds with the same output distribution.

**In world 2** all honest parties have $B$ as their best block in the interval $[\tau, \tau + \Delta_{2,3}]$. Thus $B$ is the only $(\tau, \Delta_{2,3})$-`HonestStem` and the only allowed output.

**In world 3** $B$ has $(|Q_{\mathsf{P}} \cap \mathcal{H}|)$-validity.

We conclude that $\pi$ cannot guarantee more than $(\max(1 + (n \mod 3), n - 3t))$-validity. $\square$

**Applicability of the Bounds.** Any specific protocol implementing $\mathcal{F}_{\mathsf{Tree}}$ could have properties circumventing our results. E.g. by using a CBFT protocol to get pre-agreement before prevoting in each round our protocols would always have $n - t$-validity, however this would defeat the purpose of a finality layer. It is an interesting open problem if any trace property (short of ones requiring BFT consensus) would allow better validity results. Note that if forks of the same length are decided by a chain rule with a *global* tie-break, the proof of Theorem 2 needs to take block propagation into account. Tie-breaks are used in some NSC protocols, but in e.g. the Bitcoin protocol the tie is decided by the order blocks arrive in *locally*.

## 5 Enig

In this section we present two finality layers; $\mathsf{Enig}_{\mathsf{HBD}}$ and $\mathsf{Enig}_{\mathsf{SPS}}$. Both protocols guarantee finalisation of an honest stem assuming $n \geq 3t + 1$ parties with at most $t$ corruptions. $\mathsf{Enig}_{\mathsf{HBD}}$ achieves $n - 3t$-validity in the $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$ network model, while $\mathsf{Enig}_{\mathsf{SPS}}$ achieves $n - 2t$-validity in the $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ network model. These validity bounds are shown to be optimal for each network model in Theorem 2 and Theorem 1 respectively.

**Weak Byzantine Agreement.** $\mathsf{Enig}_{\mathsf{HBD}}$ and $\mathsf{Enig}_{\mathsf{SPS}}$ rely on a "Weak Byzantine Agreement" protocol (wBA), inspired by the FilteredWMVBA protocol of [11]. It accepts an arbitrary set of inputs, and outputs a value to all honest parties which is either $\perp$ or consistent with the input of at least one honest party. If all honest parties have the same input, then it is guaranteed to be the output. FilteredWMVBA has the same properties, but it is not player replaceable and has quadratic flood complexity. However most parts of FilteredWMVBA are – or can trivially be adapted to be – SPR and have linear flood complexity, except for the subprotocol ABBA [2]. To achieve these additional properties we provide the binary byzantine agreement protocol YABBA[3]. We will present the details of YABBA in Section 6 and the details of FilteredWMVBA in Section 7.

**Running Enig.** Both of the finality layers are presented as running in a loop, in which the process of finding the next final block begins as soon as the previous final block is determined. The proofs of security are however agnostic to how the rounds are initiated, as long as: 1) a round cannot be started before the previous one has an output and 2) if some honest party can start a finality round at some time, $\tau$, then in a $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$ network all other honest parties must be

[2] *Another Binary Byzantine Agreement.*
[3] *Yet Another Binary Byzantine Agreement.*

ready to start the same round at time $\tau + \Delta_{\mathtt{net}}$, while in the case of $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ this only needs to hold in strong periods of synchrony.

**Notation.** Formally both of our protocols has access to the tree functionality $\mathcal{F}_{\mathsf{Tree}}$ and uses it for querying their position in the tree. For simplicity, we will omit such explicit calls to $\mathcal{F}_{\mathsf{Tree}}$ but instead use $\mathtt{Pos}_i$ for the position of $\mathsf{P}_i$ obtained by making such query. Furthermore, each party will maintain a variable $\mathsf{Final}_r$ that denote the block that this party considers final for round $r$. When this is changed we assume the party should input $(\textsc{SetFinal}, \mathit{Final}_r)$ to $\mathcal{F}_{\mathsf{Tree}}$, but these are also omitted for clarity. Lastly, both protocols takes a parameter `sid` which is a unique session identifier that the protocol is to be instantiated with. This is used to generate a unique round specific finalization identifier `faid`, which again is used to generate a unique identifier for running WMVBA.

## 5.1 Enig in Partial Synchrony with HBD

$\mathsf{Enig}_{\mathtt{HBD}}$ utilises a slightly generalised version of the ghost function from [18], which finds the best block with support by some amount of distinct parties from a set of votes on blocks. We generalize this to a function $\mathsf{G}$ that takes a threshold $\mathtt{T}$ and a set of prevotes $S \subset (\mathcal{P} \times \mathcal{B})$. Intuitively, $\mathsf{G}(S, \mathtt{T})$ calculates the "best block" $b$ s.t. at least $\mathtt{T}$ different parties in $S$ have prevoted on a block in the subtree of $b$. However in some NSC's (e.g. Bitcoin) the chain rule is not well-defined across different parties. To account for cases where it is not clear across all parties which block is best we let $\mathsf{G}$ return a set of blocks. All blocks in that set are justified and a party can choose its vote among these by a local chain rule, or any other rule. Our only assumption on the chain rule is that $b \prec b'$ implies $b'$ is better than $b$ in the view of all honest parties. Additionally, to mitigate complications arising from a single party voting for more than one block: whenever a party submits conflicting prevotes $(\mathsf{P}, b)$ and $(\mathsf{P}, b')$ the $\mathtt{T}$ is lowered by 1 and all current and future prevotes from $\mathsf{P}$ ignored. We define this formally below.

**Definition 6** (Threshold GHOST function ($\mathsf{G}$))**.** Let $\mathtt{T}$ be a threshold and let $S \subset (\mathcal{P} \times \mathcal{B})$ be a set of prevotes. Let $E(S)$ denote be the set of equivocating parties in $S$, i.e. $E(S) = \{p \in \mathcal{P} \mid (p, b), (p, b') \in S \land b \neq b'\}$. Further, let $\mathtt{supported}(b, S)$ be a predicate that is true when there are at least $\mathtt{T} - |E(S)|$ non-equivocating prevotes in the subtree of $b$. We now define $\mathsf{G}(S, \mathtt{T})$ to be the set of all blocks where for any block in the set $b$, we have $\mathtt{supported}(b, S)$ and for any block $b'$ where $b \prec b'$ we have $\neg\mathtt{supported}(b', S)$.

$\mathsf{Enig}_{\mathtt{HBD}}$ uses two justifications. The first one ensures that the finalised blocks form a chain.

**Definition 7** (Prevote justification)**.** We say that a block is $\mathtt{J}_{\textsc{Prevote}}$-justified for round $r$ if it is in the subtree of $\mathsf{Final}_{r-1}$.

A prevote message $(\mathtt{faid}, \textsc{Prevote}, b)$ is justified if $b$ is $\mathtt{J}_{\textsc{Prevote}}$-justified.

**Definition 8** (Vote justification)**.** We say that a block, $b$, is $\mathtt{J}_{\textsc{Vote},\mathtt{T}}$ justified for round $r$ if there is a set of $\mathtt{J}_{\textsc{Prevote}}$-justified prevotes from $n - t$ different parties, $S$, such that $b \in \mathsf{G}(S, \mathtt{T})$.

A vote message $(\mathtt{faid}, \textsc{Vote}, b)$ is justified if $b$ is $\mathtt{J}_{\textsc{Vote},n-2t}$-justified.

*Remark* 1. It is efficiently computable to check whether or not a particular block $b$ is $\mathtt{J}_{\textsc{Vote},\mathtt{T}}$ justified from a set of $\mathtt{J}_{\textsc{Prevote}}$-justified prevotes. Initially, let $S$ be the set of received prevotes and let $L(S, b)$ be the blocks in the subtree of $b$ which are in $\mathsf{G}(S, \mathtt{T})$. I.e. $L(S, b) = \{b' \in \mathsf{G}(S, \mathtt{T}) | b \preceq b'\}$. While $|S| \geq \mathtt{T}$ and $L(S, b) \neq \emptyset$:

1. If $L(S, b) = \{b\}$ then $b \in \mathsf{G}(S, \mathtt{T})$, i.e. $b$ is $\mathtt{J}_{\textsc{Vote},\mathtt{T}}$ and we are done.

2. Otherwise for each $b' \in L(S, b)$. While $b' \in \mathsf{G}(S, \mathtt{T})$ remove a prevote in the subtree of $b'$ from $S$.

Note that for any block in $\mathsf{G}(S, \mathtt{T})$ none of its predecessors can be in $\mathsf{G}(S, \mathtt{T})$. This procedure leaves the maximal number of prevotes in $S$ that does not give $\mathtt{T}$-support to any block better than $b$.

**Lemma 1.** *Assuming $n$ participants and at most $t$ corruptions a $J_{\text{VOTE},\mathtt{T}}$ -justified block has at least $\mathtt{T} - t$ prevotes from honest parties in its subtree for $t \leq \mathtt{T} \leq n$.*

**Proof.** Let $b$ be a $\mathsf{J}_{\text{VOTE},\mathtt{T}}$-justified block. Then $b \in \mathsf{G}(S, \mathtt{T})$ for some set of prevotes $S$. Let $k$ be the number of equivocating parties in $S$. Then $b$ has at least $\mathtt{T} - k$ prevotes from non-equivocating parties in its subtree. At most $t - k$ of these are from corrupted parties. Thus there are at least $\mathtt{T} - k - (t - k) = \mathtt{T} - t$ prevotes from honest parties in the subtree of $b$. □

With these definitions in place we are now ready to define $\mathsf{Enig}_{\mathtt{HBD}}$.

**Protocol** $\mathsf{Enig}_{\mathtt{HBD}}(\mathtt{sid}, \delta)$

Let $\mathsf{Final}_0$ be the genesis block.
In each round $r = 1, 2, \ldots$ party $\mathsf{P}_i$ does the following:

```
1: Let $\mathtt{faid} := (\mathtt{sid}, r)$.
2: Flood $(\mathtt{faid}, \text{PREVOTE}, \mathsf{Pos}_i)$.
3: Collect prevotes from at least $n - t$ parties, and set $\mathtt{i} := 1$.
4: while $\mathsf{Final}_r = \perp$ do
5:     Let $S$ be the set of received prevotes, pick $b$ arbitrarily from $\mathsf{G}(S, n - 2t)$ and flood $(\mathtt{faid}, \text{VOTE}, b, \mathsf{ticket}_{r,\mathtt{i}})$ at time $\tau$.
6:     Wait until $\tau + \delta \cdot \mathtt{i}$, collect votes from at least $n - t$ parties, and pick $(\mathtt{faid}, \text{VOTE}, b, \mathsf{ticket}_{r,\mathtt{i}})$ with the best ticket received.
7:     Using $b$ as proposal, let $\mathtt{i} := \mathtt{i}+1$ and $\mathsf{Final}_r := \mathsf{wBA}((\mathtt{faid}, \mathtt{i}), \mathsf{J}_{\text{VOTE},n-2t}, \{k \cdot \delta\}_{k \in \mathbb{N}})$.
8: end while
```

*Remark* 2. The running time of the protocol is a polynomial of $\delta$ and $\Delta_{\mathtt{net}}$. The parameter $\delta$ can for the sake of the proof be chosen as an atomic unit of time. In practice we conjecture that choosing a good guess on the typical network delay works well, but that is outside the scope of the theoretical analysis.

**Protocol Intuition.** Parties send a prevote message for their best block, wait until they have received at least $n - t$ of such prevotes, and then send a vote message for $\mathsf{G}(S, n - 2t)$, where $S$ is the set of received prevotes. At this point all justified votes are valid candidates for finalisation. Intuitively justifying votes using $\mathsf{J}_{\text{VOTE},n-2t}$ means that the at most $t$ adversarial prevotes in the set cannot hinder finalisation of an honest stem by choosing blocks in the prefix of the intersection of honest prevotes: in that case the result would simply become the intersection of $n - 2t$ honest prevotes in $S$. At the same time if the adversarial votes are in the subtree of a the resulting vote $b$, then there is still at least $n - 3t$ honest votes in the subtree of $b$. It follows that agreeing on any single justified vote will match the validity bound of Theorem 2. To get agreement on a justified vote the parties attach a ticket to the votes and iteratively run $\mathsf{wBA}$ picking the best seen so far as input until the output is a block. The honest tickets propagate within a $\Delta_{\mathtt{net}}$, and after that the expected number of iterations is constant, as only an expected constant amount of adversarial tickets are better than all honest tickets.

## 5.2 Enig in Partial Synchrony with SPS

$\mathsf{Enig}_{\mathsf{SPS}}$ is designed for the $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ model and uses the fact that $\Delta_{\mathtt{net}}$ is known to conclude that in periods of synchrony all honest messages have propagated within a known timeout, and as such the threshold of the ghost function can be increased to $n-t$ without hurting liveness in periods of synchrony. So while we reuse the justification of prevotes, in $\mathsf{Enig}_{\mathsf{SPS}}$ a vote message $(\mathtt{faid}, \textsc{Vote}, b)$ is justified if $b$ is $\mathsf{J}_{\textsc{Vote},n-t}$-justified.

Notice that the vote justification in contrast to our other justifications does not prove that the party sending the vote acted as an honest party could have done under certain schedulings of the messages. While honest parties (in periods of synchrony) will base their vote on all honest prevotes, corrupted parties could justify a vote based on any $n-t$ prevotes. Determining whether that is the case is in general hard, but we can get around this by only accepting votes that are at least as good as $t$ blocks included in other votes, and thus at least as good as the vote of an honest party. This constraint on inputs to $\mathsf{wBA}$ is sufficient to achieve finalisation of the honest stem. It can be implemented by simply ignoring up to $t$ votes and proceed as in $\mathsf{Enig}_{\mathtt{HBD}}$, which does invalidate up to $t$ honest votes, and consequently might reduce the change of getting an honest leader. Alternatively one can add an extra round where parties pick a "qualified vote" and only then publish their tickets. Either approach works but we choose the latter as it results in a simpler proof and statement of liveness guarantees.

**Definition 9** (Qualified vote justification)**.** A block $b$ is $\mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}$-justified proposal if it is $\mathsf{J}_{\textsc{Vote},n-t}$-justified and the set of votes for blocks in the prefix of $b$ contains votes from at least $t+1$ parties.

**Protocol** $\mathsf{Enig}_{\mathsf{SPS}}(\mathtt{sid})$

```
Let $\mathsf{Final}_0$ be the genesis block.
In each round $r = 1, 2, \dots$ party $\mathsf{P}_i$ does the following:
 1: Let $\mathtt{faid} := (\mathtt{sid}, r)$.
 2: Flood $(\mathtt{faid}, \textsc{Prevote}, \mathsf{Pos}_i)$ at time $\tau$.
 3: Wait until $\tau + 2\Delta_{\mathtt{net}}$, and collect $\textsc{Prevote}$ messages from at least $n-t$ parties.
 4: Let $S$ be the set of received prevotes, pick $b$ arbitrarily from $\mathsf{G}(S, n-t)$ and flood
    $(\mathtt{faid}, \textsc{Vote}, b)$.
 5: Collect $\textsc{Vote}$ messages from at least $t+1$ parties and set $\mathtt{i} := 1$.
 6: while $\mathsf{Final}_r = \bot$ do
 7:     Pick $(\mathtt{faid}, \textsc{Vote}, b)$, where all other received votes are in the prefix of $b$.
 8:     Flood $(\mathtt{faid}, \textsc{QualifiedVote}, b, \mathsf{ticket}_{r,\mathtt{i}})$ at time $\tau'$.
 9:     Wait until $\tau' + 2\Delta_{\mathtt{net}}$, and collect $\textsc{QualifiedVote}$ messages from at least $n-t$
    parties.
10:     Pick $(\mathtt{faid}, \textsc{QualifiedVote}, b', \mathsf{ticket}_{r,\mathtt{i}})$ with the best ticket received.
11:     Using $b'$ as proposal, let $\mathtt{i} := \mathtt{i} + 1$ and $\mathsf{Final}_r :=$
    $\mathsf{wBA}((\mathtt{faid}, \mathtt{i}), \mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}, \{\Delta_{\mathtt{net}}\}_{k \in \mathbb{N}})$.
12: end while
```

# 6 Yet Another Binary Byzantine Agreement

In this section we describe $\mathsf{YABBA}$: a binary byzantine agreement inspired by, and usable as a drop in replacement for $\mathsf{ABBA}$. Additionally $\mathsf{YABBA}$ has linear flood complexity and is SPR.

YABBA is a randomised graded agreement protocol consisting of multiple phases. Instead of using core-set selection as ABBA, YABBA uses a double proposal stage after which parties make a graded choice to update their bit. Parties use randomisation in the form of an elected leader to help choosing their bit in case they did not find a super-majority for one of the proposals. The pre-agreed parameters of the protocol are a unique id $\texttt{baid}$, a justification $J_{\texttt{in}}$, and a sequence[4] of guesses on the network delay $\{\Delta_{\texttt{YABBA,k}}\}_{k \in \mathbb{N}}$. Each party $\mathsf{P}_i$ inputs a $J_{\texttt{in}}$-justified bit $\texttt{b}_i \in \{0,1\}$. The output of each honest party is a $J_{\texttt{out}}$-justified bit (cf. Definition 13).

We will show that YABBA (with the right sequences of delays) achieves the following properties both in the hidden bounded delay and periods of synchrony setting.

**Consistency:** If some honest $\mathsf{P}_i$ and $\mathsf{P}_j$ output bits $\texttt{b}_i$ and $\texttt{b}_j$, then $\texttt{b}_i = \texttt{b}_j$.

**Validity:** If all honest parties input the same $J_{\texttt{in}}$-justified bit $\texttt{b}$, then no honest party outputs a bit $\texttt{b}' \neq \texttt{b}$.

**Termination:** If all honest parties input some $J_{\texttt{in}}$-justified bit, then eventually all honest parties will output some bit.

We require the following justifications in YABBA.

**Definition 10** ($J_{\texttt{proposal,1}}$)**.** A bit $\texttt{b}$ is $J_{\texttt{proposal,1}}$-justified if it is $J_{\texttt{in}}$-justified.

**Definition 11** ($J_{\texttt{proposal,k}}$)**.** A bit $\texttt{b}$ is $J_{\texttt{proposal,k}}$-justified for party $\mathsf{P}$ if either

- $\mathsf{P}$ received at least one double proposal for $\texttt{b}$ from phase $k-1$ or;
- $\mathsf{P}$ received at least $t+1$ proposals for $\texttt{b}$ and $n-t$ double proposals for $\perp$ from phase $k-1$.

A proposal message $(\texttt{baid}, \textsc{proposal}, k, \texttt{b})$ is justified for phase $k$ if $\texttt{b}$ is $J_{\texttt{proposal,k}}$-justified.

**Definition 12** ($J_{\texttt{doubleProposal,k}}$)**.** A bit $\texttt{b}$ is $J_{\texttt{doubleProposal,k}}$-justified if there are at least $n-t$ $J_{\texttt{proposal,k}}$-justified proposals for $\texttt{b}$.

$\perp$ is $J_{\texttt{doubleProposal,k}}$-justified if there is a $J_{\texttt{proposal,k}}$-justified proposal for both 1 and 0.

A message $(\texttt{baid}, \textsc{double proposal}, k, c_i)$ is justified if $c_i$ is $J_{\texttt{doubleProposal,k}}$-justified.

**Definition 13** ($J_{\texttt{out}}$)**.** A bit $\texttt{b}$ is $J_{\texttt{out}}$-justified if $n-t$ parties in some phase $k$ sent $(\texttt{baid}, \textsc{double proposal}, k, \texttt{b})$.

**Protocol** YABBA($\texttt{baid}$, $J_{\texttt{in}}$, $\{\Delta_{\texttt{YABBA,k}}\}_{k \in \mathbb{N}}$)

The protocol is described from the view point of a party $\mathsf{P}_i$ which has $J_{\texttt{in}}$-justified input $\texttt{b}_i$. The party starts both the "Graded Agreement" and the "Closing Down" part of the protocol.

**Graded Agreement**

In each phase $k = 1, 2, \ldots$ do the following:

```
1: $\mathsf{P}_i$ records current time, $\tau_{start}$, then computes its lottery ticket $\texttt{ticket}_i$ and floods proposal message
   $(\texttt{baid}, \textsc{proposal}, k, \texttt{b}_i)$ along with $\texttt{ticket}_i$.
2: $\mathsf{P}_i$ collects at least $n-t$ justified proposal messages, and sets
```

$$c_i = \begin{cases} 0 & \textbf{if } \text{at least } n-t \text{ collected proposal bits are } 0 \\ 1 & \textbf{if } \text{at least } n-t \text{ collected proposal bits are } 1 \\ \perp & \text{otherwise} \end{cases}$$

```
   and floods double proposal $(\texttt{baid}, \textsc{double proposal}, k, c_i)$.
```

[4]The description of this sequence should be polynomial in the security parameter $\lambda$.

3: Finally $\mathsf{P}_i$ waits until $\tau_{start} + 2\Delta_{\texttt{YABBA,k}}$, collects at least $n-t$ double proposal messages, and then sets

$$\mathtt{b}_i = \begin{cases} 0 & \textbf{if } \text{at least one collected double proposal bit is } 0 \\ 1 & \textbf{if } \text{at least one collected double proposal bit is } 1 \\ b & \text{otherwise} \end{cases}$$

where bit $b$ was in the collected proposals $> t$ times. If this bit is not unique select the proposed bit $b$ from the party with highest lottery ticket.

**Closing Down**

Once having receiving at least $n-t$ messages $(\mathtt{baid}, \textsc{double proposal}, k, \mathtt{b})$ for any bit $\mathtt{b} \in \{0,1\}$ and phase $k$, terminate the protocol and output $J_{\texttt{out}}$-justified $\mathtt{b}$.

We first show that double proposals in any phase are for the same bit or $\perp$.

**Lemma 2.** *For any phase $k$, if at least half of the honest parties propose $\mathtt{b}$, then $1-\mathtt{b}$ cannot be $J_{\texttt{doubleProposal,k}}$-justified.*

**Proof.** Assume $m \geq \frac{n-t}{2}$ honest parties propose $\mathtt{b}$ in phase $k$. Then there can be at most $n-m$ proposals for $1-\mathtt{b}$. In order for $1-\mathtt{b}$ to become $J_{\texttt{doubleProposal,k}}$-justified there must be at least $n-t$ justified proposal messages for $1-\mathtt{b}$. Assume for contradiction that $n-m \geq n-t$. It follows that $\frac{n-t}{2} \leq m \leq t \Rightarrow n \leq 3t$; a contradiction to $t < \frac{n}{3}$. ☐

We now show that if any bit is uniquely justifiable for some phase, then all honest parties terminate in that phase.

**Lemma 3.** *If the only $J_{\texttt{proposal,k}}$-justified bit is $\mathtt{b}$, then all honest parties terminate with output $\mathtt{b}$ before ending phase $k$.*

**Proof.** As the only $J_{\texttt{proposal,k}}$-justifiable proposal bit in phase $k$ is $\mathtt{b}$, the only $J_{\texttt{doubleProposal,k}}$-justifiable double proposal is $\mathtt{b}$. And thus as soon as honest parties receive $n-t$ double proposals from phase $k$ (which they do before starting the next phase), they will terminate with output $\mathtt{b}$, unless they terminated previously using double proposals from another round. ☐

It follows that pre-agreement on a proposal in any phase, causes termination with that proposal as output in that or the following phase.

**Lemma 4.** *If all honest parties have the same $J_{\texttt{proposal,k}}$-justified bit $\mathtt{b}$ as their proposal in phase $k$, then all honest parties output $\mathtt{b}$ before ending phase $k+1$.*

**Proof.** Assume all honest parties have $J_{\texttt{in}}$-justified input $\mathtt{b}$. By Lemma 2 the only $J_{\texttt{doubleProposal,k}}$-justified $c_i$ are $\mathtt{b}$ or $\perp$. This leaves the following two options for $J_{\texttt{proposal,k+1}}$-justified bits. If there is at least one justified double proposal for $\mathtt{b}$, then $\mathtt{b}$ is $J_{\texttt{proposal,k+1}}$-justifiable. If there are at least $n-t$ double proposals with $\perp$, any bit proposed by at least $t+1$ parties can be justified. However, as all honest parties proposed $\mathtt{b}$, the only bit justifiable this way is $\mathtt{b}$. As $\mathtt{b}$ is the only $J_{\texttt{proposal,k+1}}$-justifiable proposal, Lemma 3 applies in phase $k+1$. ☐

**Corollary 1** (Validity)**.** *If all honest parties input the same $J_{\texttt{in}}$-justified bit $\mathtt{b}$, then no honest $\mathsf{P}_j$ outputs a decision $\mathtt{b}' \neq \mathtt{b}$.*

Even stronger, Lemma 4 implies that with pre-agreement on $\mathtt{b}$ at the start of YABBA honest parties will terminate within two phases and output $\mathtt{b}$. In particular, this termination guarantee does not depend on the delays $\Delta_{\texttt{YABBA,k}}$.

**Corollary 2** (Fast Termination)**.** *If all honest parties input the same $J_{\texttt{in}}$-justified bit $\mathtt{b}$, they terminate within two phases with output $\mathtt{b}$.*

Consistency follows from Lemma 2 and Lemma 4.

**Lemma 5** (Consistency)**.** *If some honest* $\mathsf{P}_i$ *and* $\mathsf{P}_j$ *output bits* $\mathtt{b}_i$ *respectively* $\mathtt{b}_j$*, then* $\mathtt{b}_i = \mathtt{b}_j$*.*

**Proof.** Let $\mathtt{b}_i, \mathtt{b}_j \in \{0,1\}$ be the output of two (distinct) honest parties $\mathsf{P}_i$, $\mathsf{P}_j$ who each received $n-t$ messages $(\mathtt{baid}, \text{DOUBLE PROPOSAL}, k_i, \mathtt{b}_i)$ and $(\mathtt{baid}, \text{DOUBLE PROPOSAL}, k_j, \mathtt{b}_j)$ respectively. By Lemma 2 different bits cannot be in justified double proposals of the same round, so if $k_i = k_j$, then $\mathtt{b}_i = \mathtt{b}_j$.

Otherwise, w.l.o.g., let $k_i < k_j$. So, in phase $k_i$ there were at least $n-t$ double proposals for $\mathtt{b}_i$ and therefore $\leq 2t < n-t$ double proposals for $\bot$ (i.e. not enough to justify a proposal based on a ticket in phase $k_i+1$). Again by Lemma 2 there cannot be any double proposals for the other bit. Hence $\mathtt{b}_i$ is the only $J_{\mathtt{proposal,k_i+1}}$-justifiable bit. So by Lemma 4 at the latest after phase $k_i+2$ all honest parties will have terminated with output $\mathtt{b}_i$. Thus $\mathtt{b}_j = \mathtt{b}_i$. ☐

To show termination outside the special case of honest pre-agreement we first show that honest parties stay synchronised within the network delay $\Delta_{\mathtt{net}}$ throughout phases of YABBA, and that when the delay is chosen sufficiently large all honest tickets are propagated in time.

**Lemma 6.** *In every phase* $k$*, all honest parties move on to the next phase no later than* $\tau + 2r$*, where* $\tau$ *is the time the last honest proposal was sent and* $r = \max(\Delta_{\mathtt{net}}, \Delta_{\mathtt{YABBA,k}})$*. If additionally the actual network delay* $\Delta_{\mathtt{net}}$ *is smaller than* $\Delta_{\mathtt{YABBA,k}}$*, then all honest parties receive the tickets of all other honest parties before choosing their best ticket.*

**Proof.** Let $\tau_1$ be the last time an honest party is ready to start the first phase. Then no other honest party was ready to start the protocol earlier than $\tau_1 - \Delta_{\mathtt{net}}$. (Otherwise by message propagation, the last party would have started earlier.)

Assume now that in some phase $k$, the last honest party sends a proposal at time $\tau_k$ and all other honest parties send one no earlier than $\tau_k - \Delta_{\mathtt{net}}$. Now all honest parties receive at least $n-t$ proposals no later than $\tau_k + \Delta_{\mathtt{net}}$, and send out their double proposals. Then all honest parties receive $n-t$ double proposals no later than $\tau_k + 2\Delta_{\mathtt{net}}$. Let $r = \max(\Delta_{\mathtt{net}}, \Delta_{\mathtt{YABBA,k}})$. Then all honest parties are done waiting for messages and timeout no later than $\tau_k + \Delta_{\mathtt{net}} + 2r$ and can move on the next phase. Let $\tau_{k+1}$ be the first time an honest party can move on to phase $k+1$. Since the parties were at most $\Delta_{\mathtt{net}}$ out of sync when sending proposals, all other honest parties are done waiting for the timeout no later than $\tau_{k+1} + \Delta_{\mathtt{net}}$. And the messages allowing all parties to begin phase $k+1$ will also propagate no later than $\tau_{k+1} + \Delta_{\mathtt{net}}$.

Finally, the earliest an honest party can finish waiting for timeouts is $\tau_k - \Delta_{\mathtt{net}} + 2\Delta_{\mathtt{YABBA,k}}$. If $\Delta_{\mathtt{net}} \leq \Delta_{\mathtt{YABBA,k}}$ then $\tau_k - \Delta_{\mathtt{net}} + 2\Delta_{\mathtt{YABBA,k}} \geq \tau_k + \Delta_{\mathtt{net}}$, meaning all honest parties receive all honest proposals and tickets before choosing their proposal for the next phase. ☐

Using this synchrony guarantee we show that every time an honest party wins the lottery after the delay is sufficiently large, then the protocol stabilises with probability at least $\frac{1}{2}$.

**Lemma 7.** *If in any phase* $k$ *the network delay is smaller than* $\Delta_{\mathtt{YABBA,k}}$ *and an honest party has the highest lottery ticket, then with probability at least* $\frac{1}{2}$ *all honest parties will have the same* $J_{\mathtt{proposal,k+1}}$*-justified bit* $\mathtt{b}$ *as their proposal in phase* $k+1$*.*

**Proof.** Let $\mathtt{b}$ be the bit proposed by the majority of the honest parties, then $\mathtt{b}$ is the only justifiable bit in a double proposal (Lemma 2). Thus the parties choose their proposals for phase $k+1$ based on receiving a double proposal for $\mathtt{b}$, or by receiving only double proposals for $\bot$ and choosing the proposal with the highest lottery ticket. Note that by Lemma 6 all honest parties will see the winner. As the majority of the honest parties proposed $\mathtt{b}$ independently of their ticket, with probability at least $\frac{1}{2}$ the winner proposed $\mathtt{b}$, in which case all honest parties will choose $\mathtt{b}$ as the proposal in phase $k+1$. ☐

In any phase there is a probability of $\frac{2}{3}$ that an honest party has the highest ticket.

**Corollary 3.** *If in any phase $k$ the actual network delay is smaller than $\Delta_{\mathtt{YABBA,k}}$, then with probability at least $\frac{1}{3}$ all honest parties will have the same $J_{\mathtt{proposal,k+1}}$-justified bit $\mathtt{b}$ as their proposal in phase $k+1$.*

Next we can show that in both network models YABBA satisfies consistency, validity, and termination.

**Theorem 3.** *In $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$, the protocol* YABBA *satisfies consistency, validity, and termination assuming strictly monotone increasing $\{\Delta_{\mathtt{YABBA,k}}\}_{k\in\mathbb{N}}$.*

**Proof.** Consistency follows from Lemma 5 and validity from Corollary 1. The strict monotonicity of $\{\Delta_{\mathtt{YABBA,k}}\}$ implies there is a $k_0$ such that for $k \geq k_0$ $\Delta_{\mathtt{YABBA,k}} \geq \Delta_{\mathtt{net}}$. By Corollary 3, the honest parties achieve pre-agreement in expectation within a constant number of phases after $k_0$ and then terminate within two phases (Lemma 4). □

The above in particular holds for $\{\Delta_{\mathtt{YABBA,k}}\}_{k\in\mathbb{N}} = \{k \cdot \delta\}_{k\in\mathbb{N}}$ for some $\delta > 0$.

**Theorem 4.** *In $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$, the protocol* YABBA *satisfies consistency, validity, and terminates in expectation after constant number of phases that fall in periods of synchrony assuming $\Delta_{\mathtt{YABBA,k}} \geq \Delta_{\mathtt{net}}$ for all $k$.*

**Proof.** Consistency follows from Lemma 5 and validity from Corollary 1. By Corollary 3 and $\Delta_{\mathtt{YABBA,k}} \geq \Delta_{\mathtt{net}}$, there is a chance of at least $\frac{1}{3}$ of achieving pre-agreement among honest parties if the phase falls into a period of strong synchrony. So in expectation, honest parties will get pre-agreement within constant number of such phases. After that they will terminate within two phases (Lemma 4). □

Finally we add a concrete time bound for the periods of synchrony network model, using the very natural choice of $\{\Delta_{\mathtt{net}}\}_{k\in\mathbb{N}}$ for $\{\Delta_{\mathtt{YABBA,k}}\}_{k\in\mathbb{N}}$.

**Theorem 5.** *In $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ during strong periods of synchrony, if the last honest is party ready to start* $\mathsf{YABBA}(\cdot,\cdot,\{\Delta_{\mathtt{net}}\}_{k\in\mathbb{N}})$ *at time $\tau$, then the last honest party terminates at $\tau + 2(2+k)\Delta_{\mathtt{net}}$ or earlier, where $k$ is expected constant and $k = 0$ in case of pre-agreement.*

**Proof.** By Lemma 6 each phase of YABBA lasts at most $2\Delta_{\mathtt{net}}$. Thus YABBA terminates at the last honest party no later $\tau + 2(2+k)\Delta_{\mathtt{net}}$, where $k$ is expected constant (Theorem 4), and specifically in case of pre-agreement $k = 0$ (Corollary 2). □

## 7 Weak Byzantine Agreement

In this section, we describe a protocol for doing a weak multivalued byzantine agreement, wBA. The protocol is similar to FilteredWMVBA from [11] except it uses YABBA in place of ABBA and omits WeAreDone messages. The idea of the protocol is to first use a protocol called FilteredFreeze from [11] which "freezes" a value such that all parties either have a unique output or $\perp$ as output, and then afterwards use YABBA to decide which was the case.

For completeness, before going into details with wBA, we recap FilteredFreeze from [11], as well as prove a concrete bound for its running time.

## 7.1 Filtered Freeze

The protocol is parametric in an input justification $J$. Each party honest $\mathsf{P}_i$ has a $J$-justified input $\mathtt{p}_i$ and the output of $\mathsf{P}_i$ is justified by justification $J_{\mathtt{dec}}$ (cf. Definition 17). The protocol has the following properties.

**Weak Consistency:** If some honest $\mathsf{P}_i$ and $\mathsf{P}_j$ output decisions $\mathtt{d}_i \neq \bot$ respectively $\mathtt{d}_j \neq \bot$, then $\mathtt{d}_i = \mathtt{d}_j$.

**Validity:** If all honest parties input the same $J$-justified proposal $\mathtt{p}$, then no honest $\mathsf{P}_j$ outputs a decision $\mathtt{p}'$ with $\mathtt{p}' \neq \mathtt{p}$.

**1-Support:** If honest party $\mathsf{P}_i$ outputs decision $\mathtt{d}_i \neq \bot$, then at least one honest party had $\mathtt{d}_i$ as input.

**Termination:** If all honest parties input some justified proposal, then eventually all honest parties output some decision.

Below we describe the justifications that are used in the protocol. Note that we describe the protocol being *run light*. This is different from the description in [11] where explicit justifications are send along messages.

**Definition 14** ([11]). A proposal message $m = (\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$ from $\mathsf{P}_i$ is considered $J_{\mathtt{prop}}$-justified for $\mathsf{P}_j$ if $m$ was sent by $\mathsf{P}_i$ and $\mathtt{p}$ is $J$-justified for $\mathsf{P}_j$.

**Definition 15** ([11]). A filtered proposal message $m = (\mathtt{baid}, \text{FILTERED}, \mathtt{p})$ is considered $J_{\mathtt{filt}}$-justified for $\mathsf{P}_j$ if either there are $J_{\mathtt{prop}}$-justified proposal messages for $\mathtt{p}$ from $t+1$ different parties or there are $J_{\mathtt{prop}}$-justified proposal messages from $n-t$ different parties such that no proposal is contained in more than $t$ of those messages.

**Definition 16** ([11]). A vote message $m = (\mathtt{baid}, \text{VOTE}, \mathtt{v})$ from $\mathsf{P}_i$ is considered $J_{\mathtt{vote}}$-justified for $\mathsf{P}_j$ if is sent by $\mathsf{P}_i$ and either for $\mathtt{v} \neq \bot$ $\mathsf{P}_j$ has collected $J_{\mathtt{filt}}$-justified filtered proposal messages from at least $n-2t$ parties or for $\mathtt{v} = \bot$ $\mathsf{P}_j$ has collected $J_{\mathtt{filt}}$-justified filtered proposal messages $(\mathtt{baid}, \text{FILTERED}, \mathtt{p})$ and $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}')$ (from two different parties) where $\mathtt{p}' \neq \mathtt{p}$.

**Definition 17** ($J_{\mathtt{dec}}$-justification,[11]). A decision message $m = (\mathtt{baid}, \text{ FROZEN}, \mathtt{d})$ is $J_{\mathtt{dec}}$-justified for $\mathsf{P}_j$ if $\mathsf{P}_j$ collected $J_{\mathtt{vote}}$-justified messages $(\mathtt{baid}, \text{VOTE}, \mathtt{d})$ from at least $t+1$ parties.

**Protocol** FilteredFreeze($\mathtt{baid}$, $J$)

Each (honest) party $\mathsf{P}$ has a $J$-justified proposal $\mathtt{p}$ as input. Party $\mathsf{P}$ does the following:

**Propose:**

1. Flood proposal message $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p})$.

**Filter:**

2. Collect proposal messages $(\mathtt{baid}, \text{PROPOSAL}, \mathtt{p}_i)$. Once $J_{\mathtt{prop}}$-justified proposal messages from at at least $n-t$ parties have been collected do the following (but keep collecting proposal messages).
   (a) If your input $\mathtt{p}$ is contained in at least $t+1$ $J_{\mathtt{prop}}$-justified proposal messages, flood filtered proposal message $(\mathtt{baid}, \text{FILTERED}, \mathtt{p})$.
   (b) Else if there is any $\mathtt{p}'$ which is contained in at least $t+1$ $J_{\mathtt{prop}}$-justified proposal messages, flood filtered proposal message $(\mathtt{baid}, \text{FILTERED}, \mathtt{p}')$. Do this for at most one proposal.
   (c) Else flood $(\mathtt{baid}, \text{FILTERED}, \mathtt{p})$.

**Vote:**

3. Collect filtered proposal messages $(\texttt{baid}, \text{FILTERED}, \texttt{p}_i)$. Once $J_{\texttt{filt}}$-justified filtered proposal messages from at at least $n-t$ parties have been collected do the following (but keep collecting filtered proposal messages).
   (a) If $J_{\texttt{filt}}$-justified filtered proposal messages from at at least $n-t$ parties contain the same proposal $\texttt{p}$, flood vote message $(\texttt{baid}, \text{VOTE}, \texttt{p})$.
   (b) Otherwise flood vote message $(\texttt{baid}, \text{VOTE}, \bot)$.

**Freeze:**

4. Collects vote messages $(\texttt{baid}, \text{VOTE}, \texttt{p}_i)$ messages. Once $J_{\texttt{vote}}$-justified vote messages from at least $n-t$ parties have been collected and there is a value contained in at least $t+1$ vote messages do the following.
   (a) If $J_{\texttt{vote}}$-justified vote messages from strictly more than $t$ parties contain the same $\texttt{p} \neq \bot$ output $(\texttt{baid}, \text{FROZEN}, \texttt{p})$.
   (b) Otherwise if $J_{\texttt{vote}}$-justified vote messages from strictly more than $t$ parties contain $\bot$ output $(\texttt{baid}, \text{FROZEN}, \bot)$.
5. Keep collecting vote messages until wBA is terminated (i.e., until $\mathsf{P}_i$ gets an output in wBA). Party $\mathsf{P}_i$ keeps track of all decisions $(\texttt{baid}, \text{FROZEN}, \texttt{p})$ which become $J_{\texttt{dec}}$-justified.

The following lemma is proven in Afgjort.

**Lemma 8** ([11]). *The protocol* FilteredFreeze *satisfies weak consistency, validity, 1-support, and termination. The outputs of honest parties are $J_{\texttt{dec}}$-justified.*

Additionally, we provide the following lemma that provides a concrete bound for the running time of the protocol.

**Lemma 9.** *In both $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\texttt{net}}}$ and $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\texttt{net}}}$, the protocol* FilteredFreeze *terminates within $3\Delta_{\texttt{net}}$ (in a period of synchrony for $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\texttt{net}}}$).*

**Proof.** The protocol consists of phases. In each phase parties flood a message and then wait to collect some of the flooded messages. In the model with hidden bounded delay this requires at most one $\Delta_{\texttt{net}}$ per phase. In $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\texttt{net}}}$, this also holds if protocol is run during a period of synchrony. □

It turns out that FilteredFreeze not only satisfies validity, but has the stronger property that in case of pre-agreement on some block, the only $J_{\texttt{dec}}$-justified output is that block. We state and prove this below.

**Lemma 10.** *If all honest parties input the same $J$-justified proposal $\texttt{p}$, then no decision message different from $(\texttt{baid}, \text{FROZEN}, \texttt{p})$ can become $J_{\texttt{dec}}$-justified.*

**Proof.** As all honest parties propose $\texttt{p}$, any set of $n-t$ proposal messages will contain at least $t+1$ proposals for $\texttt{p}$ and no more than $t$ proposals for any other block. Thus the only justified filtered message is for $(\texttt{baid}, \text{FILTERED}, \texttt{p})$. It follows that the only justified vote message is $(\texttt{baid}, \text{VOTE}, \texttt{p})$, and consequently the only $J_{\texttt{dec}}$-justified decision message is $(\texttt{baid}, \text{FROZEN}, \texttt{p})$. □

A consequence of this is that whenever there is pre-agreement on the proposals in wBA, then YABBA will terminate in the first phase. (The same is true for ABBA in both WMVBA and FilteredWMVBA in [11].)

## 7.2 wBA

We are now ready to describe the protocol for multivalued byzantine agreement. The notation is adapted to use $0, 1$ as bit values in place of $\bot, \top$ in order to reserve $\bot$ for the undecided double proposal in YABBA. The pre-agreed parameters of the protocol are a unique id $\texttt{baid}$, a justification $J$, and a sequence of delays $\{\Delta_{\texttt{YABBA,k}}\}_{k\in\mathbb{N}}$. Each party $\mathsf{P}_i$ inputs a $J$-justified proposal $\texttt{p}_i$. The output of each honest party is an $J_{\texttt{fin}}$-justified decision $\texttt{d}_i$ (see Definition 19). The idea of wBA is to first call the FilteredFreeze subprotocol to boil down the choice to a unique proposal or $\bot$. Parties then use YABBA to determine which one is the case.

**Justifications.** wBA requires the following justifications. First, we define the $J_{\texttt{in}}$-justification for inputs to YABBA. The idea is that parties input 1 if their FilteredFreeze output is $\texttt{d} \neq \bot$ and 0 otherwise.

**Definition 18** ([11])**.** A bit $\texttt{b}$ is $J_{\texttt{in}}$-justified (*input justified*) for party $\mathsf{P}_i$ if $\mathsf{P}_i$ has a $J_{\texttt{dec}}$-justified tuple $(\texttt{baid}, \text{FROZEN}, \texttt{d})$ where $\texttt{d} \neq \bot$ if and only if $\texttt{b} \neq 0$.

The outputs of wBA are justified as follows.

**Definition 19.** A decision $\texttt{d}$ is considered justified with respect to *final justification* $J_{\texttt{fin}}$ for $\mathsf{P}_i$ if either $\texttt{d} \neq \bot$, $\texttt{d}$ is $J_{\texttt{dec}}$-justified, and 1 is $J_{\texttt{out}}$-justified, or $\texttt{d} = \bot$ and 0 is $J_{\texttt{out}}$-justified.

**Properties.** wBA achieves the following properties.

**Weak Consistency:** If some honest $\mathsf{P}_i$ and $\mathsf{P}_j$ output decisions $\texttt{d}_i \neq \bot$ respectively $\texttt{d}_j \neq \bot$, then $\texttt{d}_i = \texttt{d}_j$.

**Validity:** If all honest parties input the same $J$-justified proposal $\texttt{p}$, then no honest $\mathsf{P}_j$ outputs a decision $\texttt{p}''$ with $\texttt{p}'' \neq \texttt{p}$.

1-**Support:** If honest party $\mathsf{P}_i$ outputs decision $\texttt{d}_i \neq \bot$, then at least one honest party had $\texttt{d}_i$ as input.

With the right sequence of input delays it also achieves.

**Termination:** If all honest parties input some justified proposal, then eventually all honest parties output some decision.

**Protocol** wBA$(\texttt{baid}, J, \{\Delta_{\texttt{YABBA,k}}\}_{k\in\mathbb{N}})$

Party $\mathsf{P}_i$ with $J$-justified input $\texttt{p}_i$ does the following.

1. Run FilteredFreeze$(\texttt{baid}, J)$ with input $\texttt{p}_i$. Denote by $\texttt{d}_i$ the $J_{\texttt{dec}}$-justified output for $\mathsf{P}_i$ from FilteredFreeze.
2. Run YABBA$(\texttt{baid}, J_{\texttt{in}}, \{\Delta_{\texttt{YABBA,k}}\}_{k\in\mathbb{N}})$ with input $\texttt{b}_i$ where $\texttt{b}_i = 0$ if $\texttt{d}_i = \bot$ and $\texttt{b}_i = 1$ otherwise. Denote by $\texttt{b}_i'$ the output of YABBA for $\mathsf{P}_i$.
3. If $\texttt{b}_i' = 0$, then terminate and output $\bot$ (which is $J_{\texttt{fin}}$-justified), otherwise (if $\texttt{b}_i' = 1$), once $\mathsf{P}_i$ has a $J_{\texttt{dec}}$-justified decision message $(\texttt{baid}, \text{FROZEN}, \texttt{d}_i)$ with $\texttt{d}_i \neq \bot$ (from FilteredFreeze) it terminates and outputs $\texttt{d}_i$.

*Remark* 3. In order to produce a witness of finality as in [11], it can be redefined as a set of messages $J_{\texttt{fin}}$-justifying a decision. Otherwise (in line with [11]) one could send $(\texttt{baid}, \text{WeAreDone}, \texttt{d}_i)$ messages at the end of wBA, and collect $t + 1$ of these before terminating.

All properties except termination follow directly from the Theorem for FilteredWMVBA in [11].

**Lemma 11** ([11])**.** *The protocol* wBA *satisfies consistency, validity,* 1*-support.*

We now show that wBA additionally achieves termination in the $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$ and $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ models respectively, when instantiated with the appropriate delays.

**Theorem 6.** *In $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$, the protocol* wBA*, satisfies consistency, validity, and termination assuming strictly monotone increasing $\{\Delta_{\mathtt{YABBA,k}}\}_{k\in\mathbb{N}}$.*

**Proof.** All properties except termination follow from Lemma 11. Termination follows from Theorem 3 and Lemma 8 (Section 7.1). ☐

**Theorem 7.** *In $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$, the protocol* wBA*, satisfies consistency, validity, and terminates in expectation after constant number of* YABBA *phases that fall in periods of synchrony assuming $\Delta_{\mathtt{YABBA,k}} \geq \Delta_{\mathtt{net}}$ for all $k$.*

**Proof.** All properties except termination follow from Lemma 11. Termination follows from Theorem 4 and Lemma 8 (Section 7.1). ☐

We finally show concrete time bounds for termination in each of the network models.

**Lemma 12.** *In $\mathcal{N}_{\mathsf{HBD}}^{\Delta_{\mathtt{net}}}$, if all honest parties input the same $J$-justified proposal* p *they terminate with output* p *within $3\Delta_{\mathtt{net}} + 2\max(\Delta_{\mathtt{net}}, \Delta_{\mathtt{YABBA,1}})$ of the last honest input*

**Proof.** The validity property of FilteredFreeze implies that all honest parties will terminate FilteredFreeze within $3\Delta_{\mathtt{net}}$ (see Lemma 9, Section 7.1) with output p. Moreover the only $J_{\mathtt{in}}$-justified input for YABBA is 1 (see Lemma 10, Section 7.1). Thus YABBA will terminate within $2\max(\Delta_{\mathtt{net}}, \Delta_{\mathtt{YABBA,1}})$ given Lemma 3. ☐

As this is independent of timeouts, the same property holds for $\mathsf{wBA}(\cdot, \cdot, \{\Delta_{\mathtt{net}}\}_{k\in\mathbb{N}})$ in the $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ model, even outside periods of synchrony, but in that case the time bound will be $3\delta + 2\max(\Delta_{\mathtt{net}}, \delta)$ where $\delta$ is the maximal network delay experienced during that period.

**Lemma 13.** *In $\mathcal{N}_{\mathsf{SPS}}^{\Delta_{\mathtt{net}}}$ during a period of synchrony, if the last honest party starts $\mathsf{wBA}(\cdot, \cdot, \{\Delta_{\mathtt{net}}\}_{k\in\mathbb{N}})$ at time $\tau$, then the last honest party terminates no later than $\tau + 5\Delta_{\mathtt{net}} + k \cdot \Delta_{\mathtt{net}}$, where $k$ is expected constant and $0$ in case of pre-agreement.*

**Proof.** Assume the last honest party starts wBA at time $\tau$. Then all honest parties will terminate FilteredFreeze within $3\Delta_{\mathtt{net}}$ (see Lemma 9) and move on to YABBA. Now Theorem 5 implies that the last honest party terminates no later $\tau + 3\Delta_{\mathtt{net}} + (2+k)2\Delta_{\mathtt{net}}$, for expected constant $k$. If however there was pre-agreement on a proposal p, then by Lemma 10 the only $J_{\mathtt{in}}$-justified bit for YABBA is 1. With only one justified input YABBA terminates after a single phase at time $\tau + 5\Delta_{\mathtt{net}}$ (see Lemma 3). ☐

The bound in case of pre-agreement even holds outside periods of synchrony, but then it will instead be 6 consecutive rounds of communication.

## 8 Security analysis of Enig

With the sub-protocols analysed we are ready to prove the security properties of $\mathsf{Enig}_{\mathtt{HBD}}$ and $\mathsf{Enig}_{\mathsf{SPS}}$.

### 8.1 Properties of Enig in Partial Synchrony with HBD

The consistency of $\mathsf{Enig}_{\mathtt{HBD}}$ follows immediately from the consistency property of wBA.

**Corollary 4** (Consistency)**.** *If two honest parties have $\mathsf{Final}_r \neq \bot$ in round $r$, they have the same value for $\mathsf{Final}_r$.*

**Theorem 8** ($n-3t$-validity)**.** *Any finalised block has at least $n-3t$-validity.*

**Proof.** By the 1-support property of $\mathsf{wBA}(\cdot, \mathtt{J}_{\text{VOTE},n-2t}, \cdot)$, when an honest party outputs a decision different from $\bot$, that decision is $\mathtt{J}_{\text{VOTE},n-2t}$ justified. By Lemma 1 a block that is $\mathtt{J}_{\text{VOTE},n-2t}$-justified has $n-3t$ honest prevotes in its subtree. $\square$

**Theorem 9** (Finalisation of an honest stem)**.** *If the first honest message of round $r$ is sent at time $\tau$ then there is a $(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$, $H$, s.t. $H \preceq \mathsf{Final}_r$.*

**Proof.** If any honest party sends a message for round $r$ at time $\tau$, then all other honest parties are ready to send their prevote at time $\le \tau + \Delta_{\mathtt{net}}$. This means all honest parties will send their prevotes for their best block at the time somewhere between $\tau$ and $\tau + \Delta_{\mathtt{net}}$. Denote by $H$ the intersection of all the honest prevotes. Note that $H$ is a $(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$. We will show show that $H \preceq \mathsf{Final}_r$.

As $\mathsf{Final}_r$ is the output of $\mathsf{wBA}(\cdot, \mathtt{J}_{\text{VOTE},n-2t}, \cdot)$ it must be $\mathtt{J}_{\text{VOTE},n-2t}$-justified (by 1-support of $\mathsf{wBA}$). So by Lemma 1 $\mathsf{Final}_r$ has $n-3t$ honest prevotes in its subtree, which when $n > 3t$ implies that either $H \preceq \mathsf{Final}_r$ (and we are done) or $\mathsf{Final}_r \preceq H$ as they have at least one common successor. Hence, it suffices to show that $\mathsf{Final}_r \not\prec H$. Let $S$ be any set of prevotes from $n-t$ parties $\mathtt{J}_{\text{VOTE},n-2t}$-justifying $\mathsf{Final}_r$. Then $\mathsf{Final}_r \in \mathsf{G}(S, n-2t)$, which implies $\mathtt{supported}(\mathsf{Final}_r, S)$ and for any block $b$ we have, $\mathsf{Final}_r \prec b \implies \neg\mathtt{supported}(b, S)$. As $H$ is in the prefix of all honest prevotes it is supported in all sets of prevotes from $n-t$ parties, specifically $\mathtt{supported}(H, S)$ holds. But $\mathsf{Final}_r \prec H$ would imply $\neg\mathtt{supported}(H, S)$, a contradiction. We conclude $H \preceq \mathsf{Final}_r$. $\square$

Next, we state two theorems ensuring that $\mathsf{Enig}_{\mathtt{HBD}}$ terminates. The first states that $\mathsf{Enig}_{\mathtt{HBD}}$ will terminate no matter the circumstances, while the second ensures fast termination when there is already pre-agreement.

**Theorem 10** (Termination)**.** *Each round of $\mathsf{Enig}_{\mathtt{HBD}}$ eventually terminates.*

**Proof.** If the first honest party has advanced to round $r$ at time $t$. Then all honest parties advance to round $r$ at time $\tau + \Delta_{\mathtt{net}}$ or earlier and immediately send their prevote. All honest parties consequently receive at least $n-t$ prevotes at the latest at time $\tau + 2\Delta_{\mathtt{net}}$, allowing them to send a vote in the first iteration of the loop. Assuming the first honest party is ready to run the $\mathtt{i}^{th}$ iteration of the loop at time $\tau_{\mathtt{i}}$, then all parties ready at time $\tau_{\mathtt{i}} + \Delta_{\mathtt{net}}$. The first party will await votes until $\tau_{\mathtt{i}} + \max(\Delta_{\mathtt{net}}, \mathtt{i} \cdot \delta)$, while all parties are done waiting for votes at time $\tau_{\mathtt{i}} + \Delta_{\mathtt{net}} + \max(\Delta_{\mathtt{net}}, \mathtt{i} \cdot \delta)$ allowing them to start an instance of $\mathsf{wBA}$. When $\mathtt{i} \cdot \delta \ge 2\Delta_{\mathtt{net}}$ this implies all honest parties receive the votes of all other honest parties before starting $\mathsf{wBA}$. If additionally the lottery is won by an honest party, then all honest parties have the same proposal in $\mathsf{wBA}$, and by the validity property it terminates with that proposal as output. In iterations where either of these conditions are not satisfied, we will not assume anything about the outcome of $\mathsf{wBA}$ except that it eventually terminates allowing parties to move on to the next iteration. But as long as the round has not terminated eventually $\mathtt{i} \cdot \delta \ge 2\Delta_{\mathtt{net}}$ will hold for all future iterations. After this point only an expected constant number of iterations are needed before the best lottery ticket is won by an honest party, as this is expected to happen with constant probability in every iteration. We conclude that each round will eventually terminate. $\square$

**Theorem 11** (Pre-agreement causes constant time termination)**.** *If all honest parties have the same chain when starting a finalisation round, then they terminate with that block within $4\Delta_{\mathtt{net}} + 3\max(\Delta_{\mathtt{net}}, \delta)$ of the last honest party starting that round.*

**Proof.** Assume all honest parties prevote for the same block before time $\tau$. This means they are ready to send a vote before time $\tau + \Delta_{\mathtt{net}}$, and start $\mathsf{wBA}$ before time $\tau + \Delta_{\mathtt{net}} + \max(\Delta_{\mathtt{net}}, \delta)$.

By Theorem 8 and Theorem 9 we know that all justified votes have $P_{n-2t\text{-validity}}$ and extend an honest stem. When all honest prevotes are on the same block $B$, then $B$ is the only block with both properties. It follows that all honest parties choose the same proposal in $\mathsf{wBA}$ and by Lemma 12 that it terminates before $\tau + \Delta_{\mathtt{net}} + \max(\Delta_{\mathtt{net}}, \delta) + 3\Delta_{\mathtt{net}} + 2\max(\Delta_{\mathtt{net}}, \delta)$. □

As most (if not all) NSC protocols rely on the majority of the parties not only being honest but also having received the most recent honest block, we conjecture that the conditions of the theorem apply most of the time.

## 8.2 Properties of Enig in Partial Synchrony with SPS

The consistency of $\mathsf{Enig}_{\mathsf{SPS}}$ follows immediately from the consistency property of $\mathsf{wBA}$.

**Corollary 5** (Consistency). *If two honest parties have* $\mathsf{Final}_r \neq \perp$ *in round $r$, they have the same value for* $\mathsf{Final}_r$*.*

**Theorem 12** ($n-2t$-validity). *Any finalised block has at least $n-2t$-validity.*

**Proof.** By the 1-support property of $\mathsf{wBA}(\cdot, \mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}, \cdot)$, when an honest party outputs a decision different from $\perp$, that decision is $\mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}$ justified. A block that is $\mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}$-justified is also $\mathsf{J}_{\textsc{Vote},n-t}$-justified and thus by Lemma 1 has $n-2t$ honest prevotes in its subtree. □

**Lemma 14.** *All $\mathsf{J}_{\textsc{Vote},n-t}$-justified blocks are on a chain.*

**Proof.** Any $\mathsf{J}_{\textsc{Vote},n-t}$-justified block has at least $n-2t$ honest prevotes in its subtree (see Lemma 1), and thus any two will have at least $n-3t \geq 1$ honest prevotes in common. Since they are in the prefix of the same block, one must be a prefix of the other. □

**Theorem 13** (Finalisation of an honest stem). *If the first honest message of round $r$ is sent at time $\tau$, and the round is in a strong period of synchrony then there is a* $(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$*, $H$, s.t. $H \preceq$* $\mathsf{Final}_r$*.*

**Proof.** We fix a specific $(\tau, \Delta_{\mathtt{net}})$-$\mathtt{HonestStem}$, $H$, to be the intersection of all blocks included honest prevotes of round $r$. By the 1-support property of $\mathsf{wBA}(\cdot, \mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}, \cdot)$, when an honest party outputs a decision different from $\perp$, that decision is $\mathsf{J}_{\mathsf{Qualified}\textsc{Vote}}$ justified. This means that $\mathsf{Final}_r$ has blocks voted for by at least $t+1$ parties in its prefix. Specifically some honest party voted for a $\mathsf{J}_{\textsc{Vote},n-t}$-justified block, $b$, where $b \preceq \mathsf{Final}_r$. By Lemma 15 the sender of $b$ received all honest prevotes, meaning $b \in \mathsf{G}(S, n-t)$ for some set $S$ containing all honest prevotes. The set of all honest prevotes ensures that $H$ is $\mathsf{J}_{\textsc{Vote},n-t}$-justified, and hence by Lemma 14 this implies that either $b \preceq H$ or $H \preceq b$. As $b$ is the output of $\mathsf{G}(S, n-t)$ this implies that $H \preceq b$. Thus $H \preceq \mathsf{Final}_r$. □

**Lemma 15** (Synchronous rounds). *When a finality round is contained in a strong period of synchrony, then all honest parties receive the prevotes of all other honest parties before voting, and the "qualified votes" of all other honest parties before starting* $\mathsf{wBA}$*. Additionally all parties are ready to start* $\mathsf{wBA}$ *no later than $6\Delta_{\mathtt{net}}$ after the first honest party sends a prevote.*

**Proof.** Let $\mathsf{P}$ be the first honest party ready to prevote in a round. Let $\tau$ be the time $\mathsf{P}$ is ready to prevote. By strong synchrony, all other honest parties are ready to prevote at time $\tau + \Delta_{\mathtt{net}}$ and all honest prevotes are received by all honest parties no later than $\tau + 2\Delta_{\mathtt{net}}$. As $\tau$ was the first time an honest party could prevote, all honest parties receive all honest prevotes before voting at some time no earlier than $\tau + 2\Delta_{\mathtt{net}}$.

The first honest party sending their qualified vote, $\mathsf{P}'$, must send it at some time $\tau' \geq \tau + 2\Delta_{\mathtt{net}}$. By strong synchrony all other honest parties will be ready to send their qualified vote no later than $\tau' + \Delta_{\mathtt{net}} \geq \tau + 3\Delta_{\mathtt{net}}$ (notice that they are not waiting in the vote step as it started at $\tau + \Delta_{\mathtt{net}}$ or earlier). This means that all honest parties receive the qualified votes of all other honest parties at time $\tau' + 2\Delta_{\mathtt{net}}$ or earlier. As $\tau'$ was the earliest possible time to send a qualified vote, all honest parties receive every honest qualified vote before entering $\mathsf{wBA}$.

The last honest vote is sent no later than $\tau + 3\Delta_{\mathtt{net}}$ and received by everyone no later than $\tau + 4\Delta_{\mathtt{net}}$. This prompts a qualified vote and wait for $2\Delta_{\mathtt{net}}$, meaning every honest party enters $\mathsf{wBA}$ no later than $\tau + 6\Delta_{\mathtt{net}}$. □

Finally, we state that in a synchronous period then $\mathsf{Enig}_{\mathsf{SPS}}$ terminates in expected constant time. Again we defer the proof to **??** due to space constraints.

**Theorem 14** (Expected constant time termination)**.** *If the first honest party is ready to prevote at time $\tau$ and the round is contained in a strong period of synchrony, then the round terminates in time at most $\tau + (11 + k)\Delta_{\mathtt{net}}$, where $k = 0$ in case of pre-agreement or if the best ticket is won by an honest party. Otherwise $k$ is expected constant.*

**Proof.** Lemma 15 gives us that all honest parties are ready to start $\mathsf{wBA}$ at time $\tau + 6\Delta_{\mathtt{net}}$. By Lemma 13 all honest parties have then terminated $\mathsf{wBA}$ no later than $\tau + 6\Delta_{\mathtt{net}} + 5\Delta_{\mathtt{net}} + k \cdot \Delta_{\mathtt{net}}$ for expected constant $k$, where $k = 0$ and the output is a block in case of honest pre-agreement. If all honest parties gave a prevote for the same block, then this block is the only justifiable proposal and all parties must use it as input to $\mathsf{wBA}$. If there was not pre-agreement but an honest winner then all honest party choose the vote of that winner as their proposal. In either of these two cases the protocol terminates at $\tau + 11\Delta_{\mathtt{net}}$.

In the remaining case we do not assume anything about the output of $\mathsf{wBA}$ but simply note that it is expected constant time, and the expected number of rounds before an honest party wins the lottery is constant. □

## 9 On Proving Universally Composable Security

In this section we give an overview of how to prove that our two finality layers are secure in a composable sense. For this we choose to work in the UC-framework [6], and wishes to show that our finality layers *UC realises* our ideal functionality $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$. Clearly, both protocols are dependent upon time which is not present in the standard UC-framework. So before providing our actual theorems and sketch the proofs we discuss a solution for adding this to UC.

**Time.** In order to allow the set of properties, that our functionalities are parametrised over, to also include canonical blockchain properties, our functionality need the to have access to time. Furthermore, we need channels which ensures delivery within a certain time. Therefore we adapt the notion of time from TARDIS [2]. For completeness we provide a brief recap here.

TARDIS models time via. a global functionality called a "ticker". This is functionality is written $\bar{\mathcal{G}}_{\text{Ticker}}$. The ticker allows the environment to progress time, under the restriction that it ensures that all parties are ready to do progress. A party declares itself when it is ready to progress by providing this as an input to $\bar{\mathcal{G}}_{\text{Ticker}}$. Importantly, the ticker does not expose any information about the current time to the parties them self - only ideal functionalities can query the ticker to observe whether or not a tick has happened. This ensures that even though that parties are to notify the ticker about that they are ready to progress, they are themselves oblivious to the passing of time. Therefore using this modelling of time does not impose any specific synchrony model on protocols.

We adopt the convention from TARDIS that instead of letting the functionalities query the $\bar{\mathcal{G}}_{\text{Ticker}}$ as the first thing they are activated leave this implicit and only describe their behaviour in case that a "tick" has happened via an activation rule *Tick*.

Having time available within the UC model, it is easy to define flooding functionalities providing the guarantees of the network models discussed in Section 2.2. We let $\mathcal{F}_{\mathsf{FloodBD}}$ denote a network functionality that respects Definition 2 and let $\mathcal{F}_{\mathsf{FloodSPS}}$ denote a functionality that respects Definition 1.

**UC Protocols from $\mathsf{Enig}_{\mathtt{HBD}}$ and $\mathsf{Enig}_{\mathtt{SPS}}$.** The protocols $\mathsf{Enig}_{\mathtt{HBD}}$ and $\mathsf{Enig}_{\mathtt{SPS}}$ are protocols that describes how to rounds of finality. However, in order to implement $\mathcal{F}_{\mathsf{FinTree}}$ it must additionally forward any request to see the current tree of a party GETTREE to the underlying tree $\mathcal{F}_{\mathsf{Tree}}$. We let $\pi_{\mathsf{Enig}_{\mathtt{HBD}}}$ and $\pi_{\mathsf{Enig}_{\mathtt{SPS}}}$ denote any two implementations respecting this requirement, using $\mathsf{Enig}_{\mathtt{HBD}}$ and $\mathsf{Enig}_{\mathtt{SPS}}$ respectively. Furthermore, in these protocols, each time a party floods a message we assume that this is done through the respective flooding functionality ($\mathcal{F}_{\mathsf{FloodBD}}$ or $\mathcal{F}_{\mathsf{FloodSPS}}$).

**Properties.** The functionality $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$ is parametrised by a set of safety properties $\mathbb{P}$. We consider the following safety properties.

$P_{\textbf{Consistency}}$: Only blocks that are actually in the tree of party can be declared final, and any two SETFINAL inputs are chain-forming.

$P_{k\textbf{-validity}}$: Any SETFINAL input has $k$-validity.

$P_{\textbf{HonestStem}}$: For any SETFINAL input there exists a time s.t. $\tau$ s.t. the input extends $(\tau, \Delta_{\mathtt{net}})$-`HonestStem`.

We now define the set of properties that our two finality layers provides:

$$\mathbb{P}_{\mathtt{HBD}} := \{P_{\text{Consistency}}, P_{(n-3t)\text{-validity}}, P_{\text{HonestStem}}\}$$
$$\mathbb{P}_{\mathtt{SPS}} := \{P_{\text{Consistency}}, P_{(n-2t)\text{-validity}}, P_{\text{HonestStem}}\}.$$

As we cannot introduce functionalities with exponential running time in the UC model (this does not compose) we define the class of *poly-time-checkable* properties.

**Definition 20** (Poly-time-checkable properties)**.** We say that a property $P$ is poly-time-checkable if there exists and algorithm that for any trace $t$ in polynomial time decides $P(t)$.

We note that the properties described in $\mathbb{P}_{\mathtt{HBD}}$ and $\mathbb{P}_{\mathtt{SPS}}$ are poly-time-checkable.

**Theorems and Proofs.** In [17, Chapter 3] it is shown that to prove that any protocol which leaks all I/O behaviour to an adversary, *securely* realises a functionality, it is enough to prove that it *correctly realises* the functionality. We note that $\mathcal{F}^{\mathbb{P}}_{\mathsf{FinTree}}$ has the property that all I/Os are leaked to the adversary and we will therefore make use of this theorem when proving our theorems below.

We now state our results in the UC-model and provide a sketch of how it can be proven.

**Theorem 15.** *Let $\mathbb{P}$ be a set of polynomial time checkable properties with $P_{FinalityRespecting} \in \mathbb{P}$. We have that*

1. *the protocol $\pi_{\mathsf{Enig}_{\mathtt{HBD}}}$ using $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ UC-emulates $\mathcal{F}^{\mathbb{P}\cup\mathbb{P}_{\mathtt{HBD}}}_{\mathsf{FinTree}}$,*

2. *and the protocol* $\pi_{\mathsf{Enig}_{\mathtt{SPS}}}$ *using* $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ *UC-emulates* $\mathcal{F}^{\mathbb{P}\cup\mathbb{P}_{\mathtt{SPS}}}_{\mathsf{FinTree}}$.

**Proof** (Sketch)**.** We start out proving Item 1. By the note above it is sufficient to argue that $\pi_{\mathsf{Enig}_{\mathtt{HBD}}}$ correctly realises $\mathcal{F}^{\mathbb{P}\cup\mathbb{P}_{\mathtt{HBD}}}_{\mathsf{FinTree}}$. We first note that during any execution the `Exploded`-flag of $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ never will be set. Corollary 4 ensures that the output of any final round will be equal for all parties. This, in combination with that the next round only starts after the previous round ends, that $\mathcal{F}^{\mathbb{P}}_{\mathsf{Tree}}$ enforces that $\mathtt{Pos}_i$ is below the last final block (by $P_{\text{FinalityRespecting}}$) ensures that $\mathtt{Exploded} = \perp$ for the entire execution. Furthermore, $\pi_{\mathsf{Enig}_{\mathtt{HBD}}}$ relays GETTREE-requests directly to the underlying functionality. Hence, as $\mathcal{F}_{\mathsf{Tree}}$ respects all properties in $\mathbb{P}$ when $\mathtt{Exploded} = \perp$, then the composed protocol also respects all properties in $\mathbb{P}$. What is left is thus to show that the properties in $\mathbb{P}_{\mathtt{HBD}}$ are respected. This is ensured by Corollary 4 and Theorem 8 and 9.

The proof for Item 2 follows the proof for Item 1 but instead uses Corollary 5 and Theorem 12 and 13. ☐

## References

[1] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi. Communication complexity of byzantine agreement, revisited. In *PODC*, pages 317–326. ACM, 2019.

[2] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner. TARDIS: A foundation of time-lock puzzles in UC. In *EUROCRYPT (3)*, volume 12698 of *Lecture Notes in Computer Science*, pages 429–459. Springer, 2021.

[3] E. Blum, A. Kiayias, C. Moore, S. Quader, and A. Russell. The combinatorics of the longest-chain rule: Linear consistency for proof-of-stake blockchains. In S. Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1135–1154. SIAM, 2020.

[4] V. Buterin and V. Griffith. Casper the friendly finality gadget. *CoRR*, abs/1710.09437, 2017.

[5] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang. Combining GHOST and casper. *CoRR*, abs/2003.03052, 2020.

[6] R. Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In M. I. Seltzer and P. J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999.

[8] T. H. Chan, R. Pass, and E. Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptol. ePrint Arch.*, page 981, 2018.

[9] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[10] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[11] T. Dinsdale-Young, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi. Afgjort: A partially synchronous finality layer for blockchains. In *SCN*, volume 12238 of *Lecture Notes in Computer Science*, pages 24–44. Springer, 2020.

[12] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[13] J. A. Garay, J. Katz, R. Kumaresan, and H. Zhou. Adaptively secure broadcast, revisited. In C. Gavoille and P. Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 179–186. ACM, 2011.

[14] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 291–323. Springer, 2017.

[15] P. Gazi, A. Kiayias, and A. Russell. Tight consistency bounds for bitcoin. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 819–838. ACM, 2020.

[16] M. Hirt and V. Zikas. Adaptively secure broadcast. In H. Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 466–485. Springer, 2010.

[17] J. B. Nielsen. *On protocol security in the cryptographic model.* PhD thesis, 2003.

[18] A. Stewart and E. Kokoris-Kogia. GRANDPA: a byzantine finality gadget. *CoRR*, abs/2007.01560, 2020.

# Appendix

## A Proof of Stake Lotteries

Our protocol can be analysed in the UC model using the model of lotteries in [10]. We by far use all the details of the model, so the proof will hold for most PoS lotteries. We here for completeness recap the parts of the model we need.

All proof-of-stake blockchains rely on some sort of lottery. In each round, or for each job to be carried out, each party gets a ticket which has some weight. We will here identify each lottery by a lottery identifier $\mathsf{lid}$. They take the same role as the slots in [10]. They just name the individual lotteries. Very simplified the tickets are a verifiable random function of the lottery identifier $\mathsf{lid}$ and the party identifier $\mathsf{P}$, we write $\mathsf{ticket}_{\mathsf{lid},\mathsf{P}}$. This hides many important details. The ticket for instance also depend on a nonce which is needed for proving security in the case of a dynamic stake distribution. The weight of the ticket is scaled with the party's amount of stake. This makes it hard to predict which party will win $\mathsf{lid}$. It at the same time makes it easy to prove if one won the lottery and ensures that the probability that the largest ticket is held by an honest party is proportional to the fraction of honest stake.

How the probability of winning scales with the relative stake can vary. In [10] a party with relative stake $\alpha$ wins the lottery (has the highest ticket) with probability

$$\phi(\alpha) = 1 - (1 - f)^{\alpha}$$

for a tweakable hardness coefficient $f$. This means that the probability of having the highest ticket is not linear in $\alpha$ but slightly concave. This function is chosen to have the property that $1 - \phi(\alpha + \beta) = (1 - \phi(\alpha))(1 - \phi(\beta))$. This ensures that the probability of winning the lottery does not depend on whether you have all you stake on one account or several accounts. Ensuring that a party $\mathsf{P}$ with probability $\alpha_{\mathsf{P}}$ wins with probability $\phi(\alpha_{\mathsf{P}})$ is done simply by defining that a party wins is the output of the VPRF is below a threshold which is a fraction $\phi(\alpha_{\mathsf{P}})$ of the output range of the VPRF.

It is important that the lottery identifiers $\mathsf{lid}$ are fixed by the protocol. If they are dynamically chosen by the adversary it could pick the $\mathsf{lid}$ giving better tickets for corrupted parties. It follows by inspection of our protocols that all $\mathsf{lid}$'s are fixed values out of the control of the adversary.

We will use PoS lotteries for two purposes. First of all we will use them for compiling PR protocols by selecting parties for "small" committees for a round. We will also use them within PR protocols. When a PR protocol is compiled, then for each round a small committee of size $n$ is elected. Once this happened and each committee members sends a message it can be convenient to be able to do leader election within the committee.

### A.1 For Compiling PR Protocols

We first discuss how to use PoS lotteries to compute PR protocols. Instead of electing a unique winner for a $\mathsf{lid}$, one can elect a set of winners. To do this one uses a tweakable threshold $T$. A party is called a winner if the weight of its ticket is below $T$. For a lottery identifier $\mathsf{lid}$ we use $\mathcal{P}(\mathsf{lid})$ to denote the set of parties with a ticket below $T$.

In [10, Definition 6] the authors define for $\mathsf{lid}$ the characteristic value

$$w_{\mathsf{lid}} = \begin{cases} \perp & \text{if } \mathcal{P}(\mathsf{lid}) = \emptyset \\ 0 & \text{if } |\mathcal{P}(\mathsf{lid})| = 1 \text{ and the winner is honest} \\ 1 & \text{if } |\mathcal{P}(\mathsf{lid})| > 1 \text{ or some winner is malicious.} \end{cases}$$

We call a lottery *inactive* if $\mathcal{P}(\mathsf{lid}) = \emptyset$. We call it *active* if $\mathcal{P}(\mathsf{lid}) \neq \emptyset$.

As part of the analysis in [10] it is shown that if there is fraction $1/2 + \epsilon$ honest stake for a positive $\epsilon$, then the hardness of the protocol can be set such that for each $\mathsf{lid}$ independently it holds that $\Pr[w_{\mathsf{lid}} = 0] > \Pr[w_{\mathsf{lid}} = 1]$. In other words, when the lottery is active, then typically it is won by a single honest party. The price is that $\Pr[w_{\mathsf{lid}} = \bot] = 1 - \beta$ for some small positive constant $\beta$, i.e., there is a large fraction of inactive slots $\mathsf{lid}$. One essentially just sets the hardness of the lottery such that the probability that there is more than a single winner is below some sufficiently small $\epsilon$. This analysis easily generalises to other fractions than $1/2$. In particular, if there is fraction $2/3 + \epsilon + \gamma$ honest stake for positive $\epsilon, \gamma$, then the hardness of the lottery can be set such that $\Pr[w_{\mathsf{lid}} = 0] > (2 + \epsilon) \Pr[w_{\mathsf{lid}} = 1]$ and $\Pr[w_{\mathsf{lid}} = \bot] = 1 - \beta$ for positive constant $\beta$. This in turn implies that $\Pr[w_{\mathsf{lid}} = 0] > 2 \Pr[w_{\mathsf{lid}} = 1] + \alpha$ and $\Pr[w_{\mathsf{lid}} = \bot] = 1 - \beta$ for positive constants $\alpha, \beta$. We will assume a lottery with this property.

To avoid confusion we add a comment to the use of two slack values $\alpha$ and $\beta$. Below we essentially only need honest super majority, $\Pr[w_{\mathsf{lid}} = 0] > 2 \Pr[w_{\mathsf{lid}} = 1]$, but we need to leave a slack $\alpha$ to do a Chernoff bound.

## A.2 Leader Election

Another simpler use of PoS lotteries is leader election. Say a small committee of $n$ parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ has been elected, where parties are elected according to stake. I.e., if we let $\mathcal{C}$ be the committee then

$$\Pr[\mathsf{P} \in \mathcal{C}] \propto \phi(\alpha_{\mathsf{P}}) \ .$$

Say that we want one party $\mathsf{P}_\ell$, the leader, to execute a special role. We want that $\mathsf{P}_\ell$ is honest with constant probability. This is easy to do. Let $\mathsf{cid}$ be an identifier uniquely naming the committee. Let all parties $\mathsf{P}$ generate a ticket $\mathsf{ticket}_{(\mathsf{cid},\textsc{Lead}),\mathsf{P}}$. All parties send along $\mathsf{ticket}_{(\mathsf{cid},\textsc{Lead}),\mathsf{P}}$. Now *define* the leader to be the small committee member with the largest $\mathsf{ticket}_{(\mathsf{cid},\textsc{Lead}),\mathsf{P}}$. When the small committee members have been elected according to stake, as is the case for our applications here, then the leader election lottery should not be scaled with stake, each $\mathsf{P}$ should win with the same probability $1/n$. I.e., if we let $\mathcal{C}$ be the small committee with size $n$ and let $\mathsf{L}$ be the leader, then we should ensure that

$$\Pr[\mathsf{P} = \mathsf{L} \mid \mathsf{P} \in \mathcal{C}] \approx 1/n \ .$$

This is easy to do with the lottery in [10], as one will not operate with a threshold hold depending on $\alpha_{\mathsf{P}}$. All parties simply send along $\mathsf{ticket}_{(\mathsf{cid},\textsc{Lead}),\mathsf{P}}$ and the winner is the one with the highest ticket. Now note that

$$\Pr[\mathsf{P} \in \mathcal{C} \wedge \mathsf{P} = \mathsf{L}] = \Pr[\mathsf{P} = \mathsf{L} \mid \mathsf{P} \in \mathcal{C}] \cdot \Pr[\mathsf{P} \in \mathcal{C}] \propto 1/n \cdot \phi(\alpha_{\mathsf{P}}) \propto \phi(\alpha_{\mathsf{P}}) \ .$$

This in particular gives us that if there is a positive fraction of honest parties on the small committee, then an honest party is elected with positive probability, which is all we need.

Note that if the leader election had been scaled according to stake of committee members then the probability of $\Pr[\mathsf{P} \in \mathcal{C} \wedge \mathsf{P} = \mathsf{L}]$ would essentially have been proportional to $\phi(\alpha_{\mathsf{P}})^2$, which would allow the adversary an advantage by concentrating stake. This would not directly be insecure for our applications, as we only need a positive constant probability that an honest party wins $(\mathsf{cid}, \textsc{Lead})$. It would however slow down liveness, as liveness of some of our protocols depend on having an honest leader. It might also indirectly affect security by leading to concentration of stake. We therefore promote the flat leader election within already elected committees.

## B Player-Replaceable Protocols

We assume that the protocols proceed in rounds where each party has a role to act in each round. In [9] Micali introduces the notion of a *player-replaceable protocol*. This is a syntactic notion on a protocol saying that each party in the protocol consists of one sub-party $\mathsf{P}_i^R$ for each round $R$ and that each $\mathsf{P}_i^R$ only sends messages once and that $\mathsf{P}_i^R$ cannot pass secret state to future sub-parties $\mathsf{P}_i^{R'>R}$. Furthermore, $\mathsf{P}_i^R$ and $\mathsf{P}_i^{R'}$ can be independently corrupted. The idea behind player-replaceable protocols is that when they are deployed each role $\mathsf{P}_i^R$ can be executed by a fresh party elected using a proof-of-stake lottery. This gives strong mitigation against denial-of-service attack. All our protocols are player replaceable.

Because we assume a model with periods of asynchrony we need the following slightly more restrictive class of PR protocols than [9], which we call *symmetric PR* (SPR). This has to do with how a PR protocol can be compiled into a real-world protocol using a proof-of-stake lottery. For comparison, let us first look at how this is done in a synchronous network.

In a synchronous network we could assign a party to a role $\mathsf{P}_i^R$ as follows. All parties $\mathsf{P}$ compute a ticket $\mathsf{ticket}_{(R,i),\mathsf{P}}$ using the proof-of-stake lottery and sends the ticket. Here we use $\mathsf{lid} = (R, i)$ as the lottery identifier. To avoid too much communication a cut-off hardness can be set and only tickets heavier than this threshold will be flooded. Set the threshold such that except with negligible probability there is at least one ticket below the threshold, i.e., $\Pr[w_{\mathsf{lid}} = \perp] = \mathsf{negl}$. Now use that the network is synchronous and wait long enough that all honest tickets to have arrived at all honest parties. Along with the ticket a party sends the message $m$ it would compute in the protocol if it was to win the role $\mathsf{P}_i^R$. For each role use the message $m$ sent by the party with the largest ticket. When the party with the largest ticket is honest this guarantees that all honest parties agree on $m$ and we therefore get an honest and consistent execution of the role $\mathsf{P}_i^R$.

Consider now the case where the network is asynchronous. Consider running the above protocol for role assignment. Even if the party with the highest ticket for a role is honest the adversary can use network delays to assign a corrupted winner to the role as long as there is a single corrupted party with a ticket heavy enough to be flooded. And even if *all* honest parties with a ticket heavy enough to be flooded are honest the adversary can use message delays to have different honest parties adopt different winners, as long as there is more than one tickets heavier that the threshold. This essentially renders all roles corrupted. A solution to this problem would be to set the hardness such that there is at most one heavy enough ticket per role. Then the honest parties can asynchronously wait for a ticket for each role. When there is a single winner of the role and that winner is honest, then the role is honest. It is, however, impossible to set the hardness such that typically there is at most one winner and at the same time at least one winner. If one sets the hardness such that typically there is at most one winner when there is a winner, then most of the time there will be *no* winner of the role. A problem similar to the one discussed above is faced in [10], where the analysis is done in a so-called semi-synchronous model. This is the motivation for the definition of the definition of the characteristic value $w_{\mathsf{lid}}$. We can use the same ideas here.

We first describe how to compile a PR using a PoS lottery. Then we analyse what kind of model this gives us. We run a PR protocol in the asynchronous setting as follows. We use a population size $N$, a small committee size $n << N$, a threshold $T$, and a corruption threshold $t < n/3$. We discuss how to set these below. Assume a proof-of-stake lottery where

$$\Pr[w_{\mathsf{lid}} = 0] > 2\Pr[w_{\mathsf{lid}} = 1] + \alpha$$

and $\Pr[w_{\mathsf{lid}} = \perp] = 1 - \beta$ for some positive constants $\alpha$ and $\beta$.

In each round there are ground population of $N$ names $\mathsf{Q}_1^R, \ldots, \mathsf{Q}_N^R$, which we call the ground population roles. For each role $\mathsf{Q}_i^R$ all parties $\mathsf{P}$ compute a ticket $\mathsf{ticket}_{(R,i),\mathsf{P}}$ and flood the ticket if it is below the threshold $T$. It sends along the message $m$ that it should compute for the role. Now all parties wait to receive $n-t$ messages from distinct roles. This is done asynchronously. The party simply deadlocks until it received messages from winners of $n-t$ distinct roles. We will set up the framework such that this will not kill liveness. There will always be $n-t$ honest winners, so $n-t$ distinct messages will eventually arrive.

We now discuss how to set the parameters $N$, $T$, $n$ and $t$. Say that $\mathsf{Q}_i^R$ is *crashed* if $w_{(R,i)} = \perp$, say that $\mathsf{Q}_i^R$ is *honest* if $w_{(R,i)} = 0$, and say that $\mathsf{Q}_i^R$ is *malicious* if $w_{(R,i)} = 1$.

Let $H$ be the number of the $N$ roles which are honest. Let $M$ be the number of roles which are malicious. Clearly

$$\mathbb{E}[H] = \Pr[w_{(R,i)} = 0]N$$
$$\mathbb{E}[M] = \Pr[w_{(R,i)} = 1]N \ .$$

It follows that

$$\mathbb{E}[H] - 2\mathbb{E}[M] > \alpha N \ .$$

Let

$$h = \mathbb{E}[H] - (\alpha/5)N$$
$$m = \mathbb{E}[M] + (\alpha/5)N \ .$$

It follows from a Chernoff bound that for all positive $\alpha$ we can set $N$ to be a large enough, and linear in the security parameter, to ensure that

$$\Pr[H \le h] = \mathsf{negl}$$
$$\Pr[M \ge m] = \mathsf{negl} \ .$$

Note that we have

$$h - 2m > (2\alpha/5)/N \ .$$

It in particular follows that

$$\Pr[H - 2M > (2\alpha/5)N] = \mathsf{negl} \ , \tag{1}$$

which we use below. By now we have a model where in each round there are at least $h$ honest active roles and at most $m$ malicious active roles, and $h > 2m$. However, it turns out that we also need a known upper bound $n$ on how many roles might be active.

Let $n = \mathbb{E}[H] + \mathbb{E}[M] + (\alpha/5)N$. Call $n$ the *small committee size.* This is what we will think of as the size of the committee in each round of the SPR protocol. It is on purpose slightly larger than the expected number of active parties $H + M$. It follows from a Chernoff bound that for all positive $\alpha$ we can set $N$ to be a large enough, and linear in the security parameter, to ensure that

$$\Pr[H + N \ge n] = \mathsf{negl} \ .$$

When $H + N < n$ we think of it as there being a committee of size $n$ but $n - H - T$ of the members are crashed. For mental clarity we can even promote $n - H - M$ of the inactive ground population roles to be small committee members and then immediately declare them crashed. This is just a definitional hack, there is of course no way for the parties in the protocol to know who these crashed small committee member are.

Let $t = n - h$. This is the number of malicious or crashed small committee members, so we can think of this as $t$ corrupted small committee members. We have that

$$\begin{aligned} h &= \mathbb{E}[H] - (\alpha/5)N \\ t &= n - h = (\mathbb{E}[H] + \mathbb{E}[M] + (\alpha/5)N) - (\mathbb{E}[H] - (\alpha/5)N) = \mathbb{E}[M] + (2\alpha/5)N \ . \end{aligned}$$

It follows that

$$h - 2t = \mathbb{E}[H] - (\alpha/5)N - 2\mathbb{E}[M] - (4\alpha/5)N = \mathbb{E}[H] - 2\mathbb{E}[M] - \alpha > 0 \ .$$

This gives a model with three known thresholds $n$, $h$ and $t$, where $h > 2t$ and $n = t + h$. In each round the number of live roles is at most $n$, the number of corrupted roles is less than $t$, and the number of honest parties is at least $h$.

We call a protocol secure in the above model a *symmetric* PR protocol, as it is only guaranteed that in each round there exist $n > 3t$ parties $\mathsf{P}_1^R, \ldots, \mathsf{P}_n^R$ for which it is guaranteed that $2t + 1$ are honest, but the honest parties might not agree who these parties are, it can be any subset of the ground population. The only way to discover small committee members is to see a message from them.

Note, however, that in a SPR protocol an honest party can wait for messages from $h = n - t$ roles without deadlocking as $H \geq h$ except with negligible probability and the $h$ honest parties will each send their message. Furthermore, if two honest parties both wait for $h = n - t = 2t + 1$ messages, then because there are at most $n$ live roles, except with negligible probability, they will have heard from at least $(n - t) - t = t + 1$ common roles. To see this, note that there are at most $t$ small committee members that a given party did not hear from. So there are at most $2t$ small committee members that one of the two parties did not hear from. Finally, if two honest parties have heard from $t + 1$ common small committee members, then since there are at most $t$ corrupted small committee members, they will in turn have heard from at least one common honest small committee member. These will be the central properties exploited when proving liveness and safety of our SPR protocols.

In the above discussion we focused on compiling SPR protocol using PoS lotteries. It seems like a harder problem to compile SPR protocols using proof-of-work in an asynchronous setting. In a PoW lottery any role will eventually be won by some party solving the puzzle. Indeed, any role will eventually be won by some corrupted party. So if the adversary controls the network completely, it can make all roles of a round be run by corrupted parties by delaying messages long enough to solve all puzzles. We leave the compilation of asynchronous SPR protocols to PoW lotteries as interesting future work.