LaKSA: A Probabilistic Proof-of-Stake Protocol

Daniël Reijsbergen[*‡], Pawel Szalachowski[*‡#], Junming Ke[†], Zengpeng Li[*], and Jianying Zhou[*]

[*]Singapore University of Technology and Design, Singapore
[†]University of Tartu, Estonia
[‡]*The authors contributed equally to this work.* [#]*Now at Google.*

{daniel_reijsbergen, jianying_zhou}@sutd.edu.sg
{pjszal, junmingke1994}@gmail.com zengpengli@hotmail.com

***Abstract*—We present Large-scale Known-committee Stake-based Agreement (LaKSA), a chain-based Proof-of-Stake protocol that is dedicated, but not limited, to cryptocurrencies. LaKSA minimizes interactions between nodes through lightweight committee voting, resulting in a simpler, more robust, and more scalable proposal than competing systems. It also mitigates other drawbacks of previous systems, such as high reward variance and long confirmation times. LaKSA can support large numbers of nodes by design, and provides probabilistic safety guarantees in which a client makes commit decisions by calculating the probability that a transaction is reverted based on its blockchain view. We present a thorough analysis of LaKSA and report on its implementation and evaluation. Furthermore, our new technique of proving safety can be applied more broadly to other Proof-of-Stake protocols.**

I. Introduction

One of the main innovations of Bitcoin [39] is Nakamoto Consensus (NC), a protocol for maintaining a distributed data structure called a *blockchain*. In NC, each participating node tries to become the round leader by solving a Proof-of-Work (PoW) puzzle. If a solution is found, the node announces a *block* that contains a hash link to the previous block, the solution of the PoW puzzle, transactions, and metadata. Potential network forks are resolved by following the *longest-chain* rule. In this non-interactive leader election process, a block acts as a medium of transactions, as a leader election proof that is easily verifiable by other nodes, and as confirmation of all previous blocks. Hence, confidence in a given chain is built gradually through the addition of new blocks. The NC protocol can scale to a number of nodes that is vastly too large to be handled by traditional Byzantine Fault Tolerant (BFT) consensus protocols [18], which in every round require significant communication complexity among all nodes. However, NC has critical limitations: an enormous energy footprint, low throughput, and a slow and insecure block commitment process that relies on the absence of large-scale adversarial behavior [25]. Furthermore, its reward frequency and structure encourage centralization, which in turn amplifies its security vulnerabilities [9], [19], [29].

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24164
www.ndss-symposium.org

A promising direction in the mitigation of (some of) these drawbacks involves Proof-of-Stake (PoS) protocols, in which nodes do not vote with their computing power, but with their *stake*, i.e., blockchain tokens. In PoS, new nodes need to obtain stake from existing nodes – although this is more restrictive than Bitcoin's fully permissionless setting, PoS systems promise multiple benefits. The main benefit is reduced energy consumption, as voting is typically based on a cryptographic lottery that relies only on short messages instead of resource-intensive PoW puzzles. Interestingly, as participating nodes and their stake are known upfront, these systems also promise to commit blocks faster and with better safety properties, i.e., similar properties as those in standard BFT protocols.

However, there are many obstacles in the design of new PoS schemes. Several recent systems try to apply the longest-chain rule in the PoS setting. Unfortunately, those proposals mimic Nakamoto's design decisions that introduce undesired effects, e.g., a high reward variance which encourages centralization [27]. Moreover, although they are conceptually similar to NC, whose properties have been investigated in the literature, it has proven challenging to replace PoW-based voting by cryptographic primitives while avoiding fundamental limitations [12]. For instance, in NC it is expensive to work on PoW puzzles on two branches simultaneously because the node will eventually lose its investment on the losing branch. However, it is trivial in PoS systems to vote on multiple conflicting branches – this is informally known as the *nothing-at-stake* problem. Similarly, it is easier to launch *long-range attacks* in the PoS setting, in which an adversary obtains keys of nodes who controlled a majority of stake in the distant past and creates a long alternative chain.

In this work, we present the *Large-scale Known-committee Stake-based Agreement* (LaKSA) protocol. LaKSA is a PoS protocol that addresses the above limitations while remaining conceptually simple and free from BFT-like message complexity. In our protocol, committee members are pseudorandomly and periodically sampled to vote for their preferred views of the main chain. Evidence for these votes is included in the blocks, and users make their own decisions about whether to commit a block and act on the block's transactions – e.g., dispatch a package after receiving a cryptocurrency payment. In LaKSA, clients calculate the *probability* that the block can be reverted given how many votes support the block and its descendants, and commit the block if this probability is low enough. We show that a design with fixed-size pseudorandom committees brings multiple benefits, and although it also introduces a vulnerability to adaptive attackers, we discuss how to mitigate this threat using network-level anonymization techniques from the existing literature [8], [28].

By reducing the degree of interaction between committee members, LaKSA can scale to large numbers of active participants. This is combined with a high frequency of blocks and rewards for participating nodes, which mitigates the tendency towards centralization in NC. The chain selection algorithm in LaKSA aims to gather as much information as possible about the nodes' views, which leads to higher security and faster commit decisions. The CAP theorem [11] restricts distributed systems to select either availability or consistency in the face of network partitions. LaKSA favors availability over consistency; however, in LaKSA partitioned clients would notice that their view of the blockchain does not provide strong safety guarantees for their commit decisions – this is demonstrated through a novel use of hypothesis testing. LaKSA is also augmented with a fair and coalition-safe rewarding scheme, omitted by many competing systems [3], [21], [30].

We emphasize the impact of the major design choices in LaKSA through a detailed comparison to Algorand [30], which is a closely-related PoS protocol. Whereas Algorand's BFT-like block commitment scheme focuses on individual rounds, LaKSA aggregates information from multiple rounds and hence achieves higher security and flexibility in the sense that each user is able to set her own security thresholds. We thoroughly analyze our system, present the efficiency and deployability of LaKSA through an implementation, and discuss alternative design choices and extensions.

Contributions. LaKSA introduces a novel chain selection mechanism and probabilistic commit rules analyzed by a novel application of statistical hypothesis testing. To our best knowledge, LaKSA is the first concrete protocol with fixed-size pseudorandom committees that demonstrates its advantages through a comparison to related work. Finally, the cryptographic sampling procedure is also a new construction.

II. Preliminaries & Requirements

Network & Threat Model. Our system consists of multiple peer nodes who seek to achieve consensus over the state of a distributed ledger. The nodes are mutually untrusting with exactly the same rights and functions – i.e., no node or any subset is trusted. Nodes are identified by their unique public keys, and to avoid Sybil attacks [23] we assume that nodes require stake in terms of the underlying cryptocurrency to participate. We assume that the nodes' stakes are integers that sum to the total stake n. The initial set of nodes and their stake allocation is pre-defined in a genesis block that is known to all nodes. The stake distribution is represented by a mapping between public keys and their corresponding stake units. The system is used by clients who wish to make cryptocurrency transactions and who do not necessarily participate in the consensus protocol.

We assume that participating nodes have loosely synchronized clocks, as the protocol makes progress in timed rounds. Nodes are connected via a peer-to-peer network in which messages are delivered within Δ seconds during periods of network synchrony. We assume a partially synchronous communication model [24], where the network can be asynchronous with random network delays, but synchrony is eventually resumed after an unknown time period called global stabilization time (GST). We restrict this further by assuming that during asynchronous periods, an adversary cannot adaptively move nodes between splits without being detected, although the adversary can be present in all existing splits.

The adversary in our model is able to control nodes possessing $f = \alpha n$ "malicious" stake *units*, for any $\alpha \in [0, \frac{1}{3})$ and where n denotes the total number of stake units; hence, honest nodes possess $n - f = (1-\alpha)n$ stake units. Here, both f and n are integers. Adversarial nodes can be byzantine, i.e., they can fail or equivocate messages at the adversary's will. We assume that $\alpha < \frac{1}{3}$ is a bound in both our adversary and network models [24]. We assume that the adversary's goal is to a) stop the progress of the protocol, or b) cause a non-compromised client to commit a transaction that is reverted with a probability that is higher than a threshold p^* specified by the client. We also discuss adversarial strategies that aim to undermine other protocol properties, such as efficiency, throughput, and fairness.

In our model, honest nodes faithfully execute the protocol, but can be *offline*, meaning that their blocks and messages cannot be received by the rest of network. A node can be offline due to a network fault, or because its client is inactive due to a software or hardware fault. Such faults are usually temporary but can be permanent – e.g., if the node has lost the private key associated with its stake. For LaKSA to satisfy the property of liveness as discussed below, we require honest nodes to be online after the GST. Any node that is offline after the GST is grouped with the adversarial nodes, in line with BFT protocols. The temporary unavailability of honest nodes slows down block commitment, because support for the blocks is accumulated more slowly if fewer votes are received by the block proposers. However, as we discuss in § IV, offline nodes do not make a safety fault – i.e., a committed block being reverted – more likely.

Desired Properties. First, we desire the two following fundamental properties.

Liveness: a valid transaction sent to the network is eventually added to the global ledger and committed by all clients.
Probabilistic Safety: if for a client-specified probability $p^* \in [0, 1]$, the client commits a transaction, then the probability that this transaction is ever reverted is at most p^*.

One important aspect of LaKSA is that, unlike in traditional BFT protocols, our definition of safety is probabilistic. The relaxed safety property allows us to scale the system to thousands of active participants and to propose a simple, robust, and high-throughput system, while still achieving strong client- or even transaction-specific safety.

As we present our work in an open cryptocurrency setting, we also aim to achieve the following additional properties.

Scalability: the system should scale to large numbers of nodes.
High Throughput: the system should provide high throughput for transactions, and in particular, the consensus mechanism should not be a bottleneck for this throughput.

Efficiency: overheads introduced by the system should be reasonable, allowing system deployment on the Internet as it is today, without requiring powerful equipment or resources – e.g., CPU, memory, and bandwidth.

Fairness: an honest node with a fraction β of the total stake, should have a presence of approximately β in the main chain of the blockchain. This is especially important when the presence in the blockchain is translated into system rewards.

Coalition Safety: any coalition of nodes with the total stake $\alpha = \sum \alpha_i$, where α_i is a coalition node's stake, cannot obtain more than a multiplicative factor $(1+\varepsilon)\alpha$ of the total system rewards for some small ε.

Cryptographic Notation. We make standard cryptographic assumptions and we use the following cryptographic constructions. $H(m)$ is a collision-resistant cryptographic hash function, producing a hash value for a message m; $PRF_k(m)$ is a keyed pseudorandom function (PRF), outputting a pseudorandom string for key k and message m; $Sign_{sk}(m)$ is a signature scheme that for a secret key sk and a message m produces the corresponding signature σ; $VrfySign_{pk}(m,\sigma)$ is a signature verification procedure that returns *True* if σ is the correct signature of m under the secret key corresponding to the public key pk, and *False* otherwise.

III. Protocol

A. Blockchain Structure

LaKSA operates through a blockchain, such that each block contains a set of transactions, a link to the previous block, and various metadata. The structure of a block is

$$B = (i, r_i, H(B_{-1}), F, V, Txs, pk, \sigma), \tag{1}$$

where i is the round number (consecutive, starting from 0); r_i is a random value generated by the leader; $H(B_{-1})$ is the hash of the previous valid block through which blocks encode the *parent-child* relationship; V is a set of *votes* that support the previous block (see Eq. 2); F is a set of known, to the leader, forked blocks that have not been reported in any previous known block; Txs is a set of transactions included in the block; pk is the leader's public key; σ is a signature, created by the leader over all previous fields except pk.

Every block B supports its predecessor B_{-1} by including votes of nodes who were elected in B's round and who vouched for B_{-1} as the last block on their preferred chain. A vote has the following structure:

$$v = (i, H(B_{-1}), s, pk, \sigma), \tag{2}$$

where i is the round number; $H(B_{-1})$ is the hash of the previous valid block; s is the stake that the vote creator was elected with as a voter for the round i; pk is the voter's public key; σ is a signature, created by the voter over all previous fields except pk. Essentially, a vote encodes a stake unit with which the voter supports her blockchain view in a given round.

As our blockchain contains blocks that follow a parent-child relation and as it may contain forks, the final rendered data structure is essentially a tree (see an example in Fig. 2). However, within this tree only one branch is considered as the *main chain* whose transactions are strictly ordered. Transactions are initiated by blockchain nodes who wish to transfer their crypto tokens or execute a higher-level logic (e.g., smart contracts). Transactions typically also include fees paid to round leaders for appending them to the blockchain.

B. Voting Round

The protocol bootstraps from the genesis block and makes progress in two-step rounds. The two steps each last Δ seconds, where Δ as defined in § II – for brevity, we treat Δ as a bound on all delays including message generation and processing times. The voting procedure in each round is presented in Alg. 1. At the beginning of round i, each node obtains the round's pseudorandom beacon r and determines the voters and leaders. In § III-F we show concrete instantiations of beacon generation and discuss alternative ways of realizing it.

In the first step of any round i, each node checks whether it can vote in round i by calling $VoterStake()$, which returns the number of stake units that it can use to vote in round i. If a positive number is returned, then the node is called a *voter* in round i and it can vote for the last block of what it believes to be the main chain to support this chain. To do so, it creates a vote v (see Eq. 2) and broadcasts the vote immediately to the network. Other nodes validate each received vote by checking whether a) it is authentic, formatted correctly, and not from the future, i.e., not with a round number that exceeds i, b) it points to a valid previous block, and c) the voter is legitimate, i.e., $VoterStake()$ returns the positive stake amount as declared. After successful verification, votes are added to the pending list of votes that directly support its predecessor block. These votes create a so-called *virtual block* that consists of collected but not yet included votes, and one virtual block can support another virtual block; we discuss this further in § III-D.

After waiting for Δ seconds to collect and validate votes, nodes execute the round's second step. First, the node checks the output of the $LeaderStake()$ function, and if it is positive then the node is a *leader* in that round. If so, then the node determines the main chain – see the details in § III-D. The node then creates and propagates a new block that has the main chain's last block – which can be a virtual block – as its predecessor and which includes, among other fields (see Eq. 1), all collected votes and the generated random value r_i. A malicious leader can censor – i.e., refuse to include – votes, but we use our incentive mechanism, which is described in § III-G, to discourage this attack.

A node that receives a new block verifies whether a) it is authentic, formatted correctly, and not from the future, b) it points to a valid previous block, c) the votes are correct, and d) the leader is legitimate, i.e., $LeaderStake()$ returns a positive value. If the block is validated, it is appended to its corresponding chain. Besides pointing to the previous block, a leader in its block lists all known forks that were not reported in previous blocks, including pending votes of other blocks.

We propose a concrete instantiation of voter/leader election in § III-C. LaKSA can also be implemented with other procedures (e.g., based on VRFs as discussed in § F) as long as nodes act as leaders and voters in proportion to their stake possession. We do not restrict the roles of a node per round, i.e., a node can both be a voter and leader in a given round, or act in only one of these roles, or none.

Alg. 1: The voting procedure.

```
function VotingRound(i)
    $r \leftarrow RoundBeacon(i)$; $s \leftarrow VoterStake(pk, r)$;
    if $s > 0$ then // check if I am a voter
        $B_{-1} \leftarrow MainChain().lastBlk$; // get last block
        $\sigma \leftarrow Sign_{sk}(i \| H(B_{-1}) \| s)$;
        $v \leftarrow (i, H(B_{-1}), s, pk, \sigma)$; // support vote
        Broadcast($v$);
    Wait($\Delta$); // meanwhile, collect and verify support votes
    if $LeaderStake(pk, r) > 0$ then // check if I am a leader
        $B_{-1} \leftarrow MainChain().lastBlk$; // possibly different block
        $V \leftarrow \{v_a, v_b, v_c, ...\}$; // received $B_{-1}$'s support votes
        $r_i \leftarrow Random()$; $\sigma \leftarrow Sign_{sk}(i \| r_i \| H(B_{-1}) \| F \| V \| Txs)$;
        $B \leftarrow (i, r_i, H(B_{-1}), F, V, Txs, pk, \sigma)$; // new block
        Broadcast($B$);
    Wait($\Delta$); // wait for the next round
```

Alg. 2: Leader/voter election via cryptographic sampling.

```
function VoterStake(pk, r)
    $tmp \leftarrow Sample(q, r, \text{'vote'})$; return $tmp.Count(pk)$;
function LeaderStake(pk, r)
    $tmp \leftarrow Sample(l, r, \text{'lead'})$; return $tmp.Count(pk)$;
function Sample(size, r, role)
    $tmp \leftarrow []$; $res \leftarrow []$;
    for $pk \in stake$ do
        for $s \in \{1, ..., stake[pk]\}$ do
            $tmp.Append(pk)$;
    for $i \in \{1, ..., size\}$ do
        $k \leftarrow PRF_r(i \| role)\ \%\ Len(tmp)$;
        $res.Append(tmp[k])$; $tmp.Delete(k)$;
    return $res$;
```

C. Leader and Voter Election

We propose a method of electing leaders and voters which is based on a novel *cryptographic sampling* method presented in Alg. 2. This method creates an array of all stake units and pseudorandomly samples a fraction from it. The method uses uniquely generated PRF outputs to sample stake units. In a round, leader and voter elections should be independent, thus the $Sample()$ function takes a role parameter – 'lead' and 'vote', respectively – to randomize PRF outputs for these two elections. The function returns a list of sampled public keys – each key corresponds to a stake unit – and is parametrized by the size of the output list. In the following, q denotes how many stake units out of the total stake n are elected every round for the voting committee, and l is an analogous parameter for the number of leaders. The $VoterStake()$ and $LeaderStake()$ functions run $Sample()$ and return how many times the given public key is present in the sampled stake. In App. A, we show that our construction is indistinguishable from random sampling for computationally bounded adversaries. As stake units are sampled uniformly at random, a node with stake s can be elected as a voter between 0 and s times in any given round. For performance reasons it may be desirable to elect one leader per round, which is achieved by setting $l = 1$.

Limitations. The described approach guarantees that in every round the exact same stake fraction is sampled. As a result, nodes are able to more quickly make commit decisions. However, a disadvantage is that an adversary may try to launch an adaptive attack – e.g., (D)DoS – as elected nodes are known before they broadcast their messages. Fortunately, multiple lightweight mechanisms that provide network anonymity are available. For instance, Dandelion [8], [28], which is intended for use in cryptocurrencies, provides formal anonymity guarantees combined with low latency and overheads. Using such a mechanism together with LaKSA would complicate attempts by the adversary to identify the node's IP address, effectively mitigating the mentioned (D)DoS attack.

Another way of addressing this issue in PoS blockchains is to elect nodes using cryptographic primitives with secret inputs (e.g., VRFs as in Algorand [30], or unique signatures). Using this approach, a node's role can be revealed only by this node itself, e.g., by propagating a message. The disadvantage, as we show in § VI, is the resulting variance in the number of elected entities, which slows the block commitment process. In App. F, we show how LaKSA can be combined with VRF-based election and that our commitment scheme is still applicable. An efficient mechanism that combines "secret" election with fixed committee sizes is an open research problem.

D. Chain Selection

LaKSA does not follow the longest-chain rule of Bitcoin's NC – instead, the strength of a chain is expressed by the stake that supports its blocks. To improve the handling of forks, we incorporate the GHOST protocol [42], which improves the throughput and security of NC by utilizing fork blocks in the calculation of the total PoW weight of chains.

Forks and Virtual Blocks. In LaKSA, votes contribute to "weights" of chains and are crucial to determine the main branch. In short, they represent beliefs of stakeholders about their views of the main chain. To compare chains, nodes follow the *most-stake* rule, i.e., the chain which is supported by more stake-weighted votes is chosen. Ties can be broken in LaKSA by selecting the chain whose hash value computed over the concatenated round beacon and whose last leader's public key is smaller. LaKSA allows situations in which no block is added in a round – e.g., when a faulty node is elected as a leader or when the network is temporarily asynchronous – or in which the block contains few or no votes. In such cases, nodes create a virtual block (see § III-B) which is a set of received votes that have not yet been included in the main chain.

Virtual blocks do not have transactions or signatures, unlike 'standard' blocks. During chain selection, nodes do not distinguish virtual from "standard" blocks: if a virtual block is stronger than a conflicting standard block, then nodes will support the virtual block. Voters can support a virtual block by voting for the block's latest standard ancestor (see Fig. 1). In later rounds, leaders can collate non-included votes per round to create a sequence of virtual blocks, of which the latest is used as the predecessor of the newly proposed standard block. The virtual blocks are then transmitted by the leader along with the standard block. A leader who neglects to include votes in her block risks that it is overwritten by another leader who aggregates the non-included votes in virtual blocks. The overwritten block may still be referenced by a later block using GHOST as discussed below, but even then the malicious leader still loses her block reward (see § III-G). If a vote legitimately appears in two conflicting blocks, e.g., in a virtual block and an overwritten but referenced standard block from the same round, then one is disregarded. Since virtual blocks are only created through standard blocks, the commitment procedure of § III-E is only executed on standard blocks.

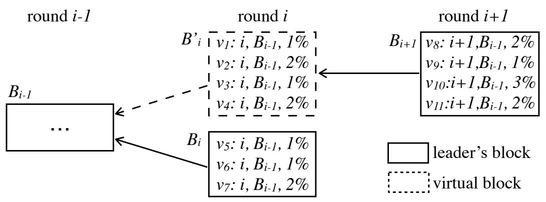


Fig. 1: An example of a virtual block fork.

Alg. 3: The chain selection procedure.

```
function MainChain()
    $best \leftarrow (.lastBlk \leftarrow B_0, .stake \leftarrow Stake(B_0))$;
    while True do
        if $Children(best.lastBlk) = \emptyset$ then
            return best;
        for $B \in Children(best.lastBlk)$ do
            $s \leftarrow TreeStake(B)$;
            if $s > best.stake$ then
                $best \leftarrow (B, s)$; // stronger subtree found
function TreeStake(root)
    $s \leftarrow 0$;
    for $B \in TreeBlocks(root)$ do
        $s \leftarrow s + Stake(B)$; // add stake directly supporting B
    return s;
```

An example is presented in Fig. 1, where $q/n = 10\%$ and voters in round i publish seven votes supporting the block B_{i-1}. The leader of round i creates a block B_i which includes only three votes with 4% of the stake. Therefore, a virtual block B'_i with 6% stake is created and in round $i+1$ all votes implicitly support this block instead of B_i, thus the leader of this round creates a block B_{i+1} that aggregates votes of the virtual block and points to B_{i-1} via B'_i. We emphasize that the block B_{i+1} can also include a pointer to B_i cf. GHOST.

Subtree Selection (GHOST). The proposed protocol is likely to work well when the network is highly synchronous – which can be achieved by choosing a sufficiently high value for Δ. However, Δ must be traded off against transaction throughput: i.e., throughput can only be high if Δ is low. If Δ is small compared to the network latency, then asynchronous periods in which blocks cannot reach nodes before the defined timeouts occur often. This would result in a high stale block ratio that harms the security of system.

In order to prepare the protocol to withstand such situations, we modify and extend GHOST to adapt it to our setting. The chain selection procedure is depicted in Alg. 3. As presented, the *MainChain* procedure starts at the genesis block. Then for each of its child blocks, the algorithm calculates the total stake in the child block's subtree, and repeats this procedure for the child block with the most stake aggregated on its subtree, and so on. When the protocol terminates it outputs the block which denotes the last block of the main branch. The chain selection procedure relies only on the stake encoded in votes and collected votes of virtual blocks – i.e., those not included in any actual block – and includes them in the total stake of their chain.

To illustrate the chain selection procedure we show an example in Fig. 2 where $q/n = 10\%$. In our example, in round i, the leader (i.e., the creator of block E) sees four chains, i.e., those ending with the blocks P, G, D, and J, respectively. To determine the main chain, the leader runs the protocol starting with the genesis block A and computes the stake of its children's subtrees. Block M's subtree stake is 11% whereas B's subtree stake is 15%. Thus, B is selected and the leader repeats the same procedure for this block. Finally, the leader determines A,B,C,D as the main chain, and creates a new block E pointing to it. Moreover, the leader introduces pointers (dashed lines) to known but yet unreported fork blocks (P,G,J). These pointers are required to preserve the integrity of the blockchain view. Note that the chain A,B,C,D is selected as the main one, despite the fact that the stake that voted for this chain is lower than the stake of the chain A,M,N,P. As different chains within a subtree support the same chain prefix, an advantage of combining GHOST with the most-stake rule is that it requires an adversary to compete with the entire subtree and not only its main chain, which makes attacks on safety much more difficult. Repeating the procedure, round $i+1$'s leader extends E's chain and reports on fork blocks K and L.

E. Block Commitment

Block commitment in LaKSA is decided by each node individually. Each node specifies p^*, its risk level, and B, the block to be committed. Given its current view of the blockchain, it then calculates the probability that the target block B can be reverted. Given our chain selection procedure, this question can be reformulated as follows: *what is the probability that an adversary can create any stronger subtree than the corresponding subtree containing B?* If the probability is less than the threshold p^*, then the block is committed.

An adversarial subtree has to be rooted at a block outside B's supporting subtree, as otherwise, it would support B. The stake of the supporting subtree is known to the node and computed simply by $s = TreeStake(B)$. An adversarial subtree can originate from any previous round, so we require that the block B cannot be committed before all of its previous blocks have also been committed. For each block, we hence consider the potential strength of an adversarial subtree that originates in the parent of B and which has k supporting blocks until the current round. As such, the supporting and adversarial branches are in competition during k rounds. The node cannot determine how many stake units support an adversarial subtree, but this knowledge is critical for the security of the commit operation. Therefore, the node splits this stake into the sum of a) *missing stake* which consists of those stake units that may unknowingly contribute to the adversarial branch during a fork, and b) *adversarial stake* which is the sum of the stake

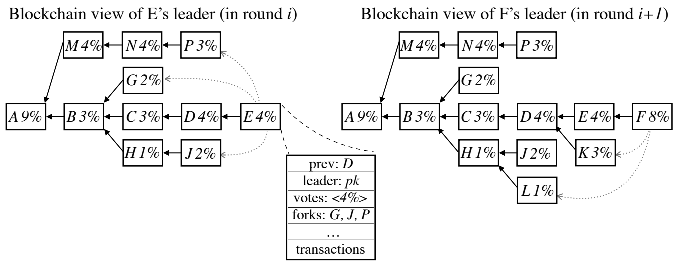


Fig. 2: Example of the evolution of the blockchain when many forks occur. In this figure (and in Fig. 3 and 7), the percentage in each block denotes the supporting stake included in that block.

Alg. 4: The block commitment procedure.

```
function Commit(lastCommit,p*)
    $B \leftarrow GetMainChainBlockAtRound(lastCommit)$;
    while $B \neq nil$ do
        $s \leftarrow TreeStake(B)$; $k \leftarrow CurrRound() - lastCommit + 1$;
        if $CalculateProb(k,s) \geq p^*$ then // B is rejected
            return;
        $lastCommit \leftarrow lastCommit + 1$;
        $B \leftarrow GetMainChainBlockAtRound(lastCommit)$;
function CalculateProb(k,t)
    if $q^k < MAX\_STAKE$ then
        return $HyperGeomProb(n,\frac{1}{2}(1+\alpha)n,q,k,t)$;
    return $CCBound(n,\frac{1}{2}(1+\alpha)n,q,k,t)$;
function HyperGeomProb(n,u,q,k,t) // hypergeom. sum prob.
    if $k = 1$ then
        $x \leftarrow 0$;
        for $i \in \{t+1,\ldots,q\}$ do
            $x \leftarrow x + HypergeometricPMF(n,u,q,i)$;
        return $x$;
    $x \leftarrow 0$;
    for $i \in \{0,\ldots,q\}$ do
        $x \leftarrow x + HypergeometricPMF(n,u,q,i)$
                $* \ HyperGeomProb(n,u,q,k-1,s-i)$;
    return $x$;
function CCBound(n,u,q,k,t) // the Cramér-Chernoff bound
    return $e^{-k*MaximizeRateFunc(n,u,q,t/k)}$;
function MaximizeRateFunc(n,u,q,t)
    $\lambda_{\max} \leftarrow RateFuncSearchRange(n,u,q,t)$;
    repeat
        for $i \in \{1,\ldots,4\}$ do
            $y_i \leftarrow RateFuncHelper(n,u,q,\lambda_i,t)$;
        $\vec{\lambda} \leftarrow ReduceInterval(\vec{\lambda},\vec{y},n,u,q,t)$;
        $\Delta \leftarrow \lambda_{i_{\max}} - \lambda_0$;
        // $\varepsilon$ is an accuracy threshold set by the user
    until $\Delta < \varepsilon$;
    return $(y_{i_{\max}} - y_0)/2$;
function RateFuncHelper(n,u,q,λ,t)
    return $t\lambda - LogHyp2F1(-q,-u,-n,1-e^{\lambda})$;
```

that the adversary accumulated over the k rounds.

The adversarial stake units can *equivocate*, which means that they are involved in the creation of two or more different blocks or votes within the same round. As such, the stake units that contribute to the adversarial branch can also be present on B's supporting subtree. Furthermore, their exact quantity is unknown, although on average this will be equal to αq. We discuss equivocations in more detail below. We assume the worst-case scenario regarding the missing stake, i.e., that all missing stake – even if it is honest – is placed in an unknown adversarial subtree. Such a situation could occur during asynchronous periods or network splits. The node also conservatively assumes that the adversary would win every tie. However, we emphasize that an adversarial subtree must be internally correct, e.g., it cannot have equivocating votes within it, as otherwise, honest nodes would not accept it.

Hypothesis Testing. To commit B, the node conducts a statistical *hypothesis test* to assert whether most of the network has the same view of the supporting tree as the node. To do so, the node computes a so-called *p-value* that represents the probability of achieving a total supporting stake s over k rounds *under the hypothesis* that fewer stake units are contributing to the supporting branch than to the adversarial branch. We commit B if this p-value is so low that we can safely conclude that this hypothesis is invalid. The function *Commit* in Alg. 4 is called once every round, and as many blocks as possible are committed during every call. To achieve safety, we use the quorum intersection argument present in traditional BFT systems, except in our case it is probabilistic. Namely, an adversary in such a split network could produce equivocating votes on both views at the same time; however, the node can conclude that such a situation is statistically unlikely if she sees that her target block is supported by more than $\frac{2}{3}kq$ stake (see details in § IV-A). That would imply that most of the honest nodes see the target block on the main chain, as any alternative adversarial view cannot obtain more than $\frac{2}{3}kq$ over time (since $\alpha < \frac{1}{3}$).

The process is described by the pseudocode of Alg. 4. At the core of the procedure, the node keeps committing the main chain's subsequent blocks by computing the probability that their corresponding supporting trees are on the "safe" side of the hypothetical fork. Depending on the parameters, the p-value is computed using one of two functions, namely *HyperGeomProb* or *CramerBound* (see Alg. 4). In § IV-A, we discuss these algorithms in more detail and show that they indeed give an upper bound on the error probability. To illustrate this process we present an example in App. E.

To keep our presentation simple, we do not parametrize the commit procedure by the f (or similarly α) parameter. However, it would not change our methodology, and in practice, it may be an interesting feature as nodes could then base commitment decisions on their own adversarial assumptions in addition to their security level p^*.

Equivocation. Although adversarial nodes can freely violate the rules of the protocol, in LaKSA, messages – i.e., blocks and votes – are signed, so adversarial nodes can be held accountable for certain equivocations. In particular, the following actions are detectable: a) equivocating by producing conflicting votes or blocks within the same round, or b) supporting or extending a chain that is weaker than a previously supported or extended chain. The former is a nothing-at-stake attack and it is provable by showing adversary's two signed messages which support or extend different chains in the same round. In the latter case, the adversary violates the chain selection rule. This can be proven by any pair of signed messages (m_1, m_2) which support/extend two different chains C_1 and C_2, respectively, where: $Round(m_1) < Round(m_2) \wedge Stake(C_1) > Stake(C_2)$.

Although prevention of such misbehavior is challenging, solutions that disincentivize it by causing the misbehaving validator to lose all or parts of her deposit/stake have been proposed [14]. Under this approach, the protocol allows honest nodes to submit evidence of equivocation to the blockchain for a reward, e.g., the finder's fee implemented in the smart contract version of Casper FFG [16]. Implementing such a scheme in LaKSA is an interesting direction for future work.

F. Random Beacon

LaKSA relies on a pseudorandom beacon to elect round leaders and voters. It is important for security that these beacons are difficult to be biased by adversaries. The safety analysis in § IV-A relies on the assumption that voters and block proposers on each branch of a fork are sampled proportionally to their stake – if the adversary is able to manipulate the random beacon's committee selection, then this assumption

Alg. 5: The reward procedure.

```
1 function RoundReward(B)
2     $pay(B.pk, R_l)$; // leader's block reward (+ opt. tx fees)
3     for $v \in B.V$ do
4         $pay(v.pk, v.s * R_v)$; // voter's reward
5         $pay(B.pk, v.s * R_i)$; // leader's inclusion reward
```

no longer holds. Furthermore, a beacon that is too predictable is vulnerable to adaptive network-level attacks (e.g., DoS) against voters and round leaders, as the window for such attacks is directly proportional to the time between the creation of a block and the publication of the random beacon.

In this paper we do not propose a new random beacon construction – instead, we rely on previously proposed concepts for a concrete instantiation. For the current implementation, we have followed the approach by Daian et al. [20], where beacons are generated purely from the random values aggregated over "stable" main chain blocks. More precisely, as presented in § III-B, an i-th round leader inserts a random value r_i into its block. For the security parameter κ, which is the number of main chain blocks after which the probability of a roll back is negligible, the random beacon r in the current round j is extracted in a two-step process, using a random oracle, from the previous random values $r_{j-2\kappa}, r_{j-2\kappa+1}, r_{j-2\kappa+2}, \ldots, r_{j-\kappa}$. An adversary can bias the outcome beacon r, but it has been proven [20] that short-time adversarial biases are insufficient to get a long-term significant advantage.

We emphasize that the use of a random beacon for leader/committee selection is not specific to LaKSA and other random beacon constructions can also be used to implement the protocol. Some recent approaches, e.g., DFINITY [31] and RandHound [43], promise scalable randomness and are potentially more bias-resistant than the presented mechanism. However, greater bias resistance may come at the cost of additional computational overhead. We leave the investigation of these schemes combined with LaKSA as future work.

G. Rewards

LaKSA introduces the rewarding scheme presented in Alg. 5. Each voter supporting the previous block of the main chain receives a voter reward R_v multiplied by the number of stake units that the voter was sampled with. The votes of virtual blocks also receive rewards as soon as a block containing them is published. Every leader who publishes a block on the main chain receives a leader reward R_l and an inclusion reward R_i for every stake unit included in the block. Leaders may also receive transaction fees paid by nodes but we omit them as they are application-specific.

The rewarding scheme in LaKSA has several goals. First, it aims to incentivize voters to publish votes supporting their strongest views immediately. Forked or "late" votes and blocks are marked in main chain blocks but they are not rewarded although leaders still have an incentive to include them since they strengthen common ancestor blocks with the main chain. Therefore, voters trying to wait for a few blocks to vote for "past best blocks" would always lose their rewards. Second, it incentivizes leaders to publish their blocks on time, as a block received after the end of the round would not be part of the main chain. After all, voters in the next round would follow their strongest view instead, so the leader would miss out on her rewards. Lastly, the scheme incentivizes block leaders to include all received votes. A leader censoring votes loses an inclusion reward proportional to the stake that was censored. Moreover, the censoring leader weakens her own blocks.

IV. Analysis

A. Probabilistic Safety

In this section we show that a block B is committed only if the probability that a conflicting block is committed is indeed below p^*. To do this, we use a novel proof technique based on *statistical hypothesis testing*. A block is committed by a user when she sees enough supporting stake to conclude that it is unlikely that she is on a branch that includes only a minority of honest users. The main threat is an adversary who wants to cause a safety fault, which means that two conflicting blocks are committed by different honest users. For this to occur, two users who have *different* views of the blockchain must *both* see enough evidence for their blocks to commit.

Our worst-case scenario – i.e., the scenario in which a safety fault is most likely to occur – is when the honest users are split during a fork. The adversary can make a safety fault more likely by voting on both branches of the fork simultaneously. Although equivocation can be punished in retrospect, it cannot be detected while the fork is ongoing and hence cannot be ruled out for our safety analysis. We assume that a fraction $\alpha \in [0, \frac{1}{3})$ of the stake is controlled by the adversary. We furthermore assume that the fork consists of *two* branches, and that the honest stake is split *evenly* among them. The reason is that if one branch is stronger than the other, then it is less likely that a block will be committed on the weaker branch. Similarly, there is no need to consider forks with three or more branches, because the probability of conflicting blocks being committed would be higher if two of the branches were combined. Finally, we do not consider users going offline or votes being withheld by the adversary – this would only make one branch weaker and a safety fault less likely, and therefore constitute a weaker adversarial scenario than the one under consideration.

The user cannot directly observe how many users are on the two branches – however, she *can* observe how many stake units support the block on her branch. The expected stake fraction on her branch in the worst-case scenario is $\alpha + \frac{1}{2}(1-\alpha) = \frac{1}{2}(1+\alpha)$, so $\frac{2}{3}$ if $\alpha = \frac{1}{3}$ (this is the same for the users on the other branch). If she observes that the amount of stake on her branch is considerably higher than this fraction, then she has evidence to conclude that her branch is stronger. In the following, we will formulate the question of whether there is sufficient evidence to commit B as a hypothesis test. Before we proceed, we note that for B to be committed, we require that all preceding blocks have also been committed. Hence, we only focus on B and its supporting subtree, and do not consider any of B's ancestors (see also the example in Fig. 3).

Recall that there are $n \in \mathbb{N}$ stake units of equal weight, where a single node may control many stake units. We draw a committee of size q stake units in every round (e.g., in Fig. 3 it holds that $q = \frac{1}{10}n$). We assume that n and q remain constant throughout the duration of the fork, as these are protocol-level

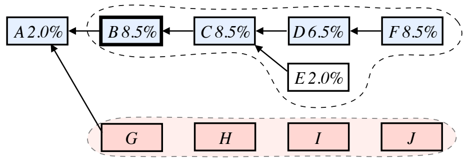


Fig. 3: If the user tries to commit B, we only evaluate the strength of B and its supporting subtree, and implicitly compare it to the branch (G,H,I,J). Even though A is weak, and B might be overturned due to a branch starting from A's predecessor, we require that A has already been committed before committing B, so it need not be considered.

parameters that change infrequently. We denote the number of expected supporting stake units per round on the user's branch by u. We also assume that u does not change during the fork – i.e., that users do not move between branches while the fork is ongoing. We make this assumption because 1) we assume that the adversary is unable to adaptively move users between branches (as stated in § II), and 2) if any of the honest users becomes aware of the other branch then they would be able to detect equivocation and send evidence to users on both branches. We assume that in a given round l, the user is interested in testing whether the amount of stake that supports a block B that appeared in round j is enough to commit it. The *null hypothesis* H_0 asserts that the supporting stake is *at most* equal to the expected worst-case support, i.e.,

$$H_0 : u \leq \frac{1}{2}(1+\alpha)n. \tag{3}$$

In the following, we compute the probability p of observing a given amount of supporting stake in B's subtree *given* that H_0 is true. This probability is commonly called a p-value. If the p-value is below p^*, then we accept the alternative hypothesis $H_1 : u > \frac{1}{2}(1+\alpha)n$ and commit the block. Of course, this experiment may be repeated over the course of several slots until B is finally committed, or if B is dropped after the resolution of the fork. Hence, we use a *sequential* hypothesis test. In the following, we first focus on calculating the probability of observing the data – i.e., the supporting stake for the block B – given the null hypothesis for given rounds m and j, and then extend this to the sequential setting.

Let X_i be the number of stake units held by supporting users sampled in round i. Since the cryptographic sampling algorithm draws without replacement, we know that X_i has a *hypergeometric* distribution [32], whose probability mass function is given by

$$\mathbb{P}(X_i = x) = \binom{u}{x}\binom{n-u}{q-x}\Big/\binom{n}{q}.$$

The *total* number of supporting units (where a single unit may feature in more than one different rounds) that is drawn in rounds $j+1,\ldots,m$ is then given by $T = \sum_{i=j+1}^{m} X_i$. Since $X_1, X_2, \ldots$ are all identically distributed, we know that T has the distribution of $k = m-j$ independent hypergeometric random variables. Hence, to calculate the probability of observing a total amount of supporting stake t given the null hypothesis, we can compute the tail probability $\mathbb{P}(T \geq t) = \sum_{i=t}^{kq} \mathbb{P}(T = i)$ under the assumption that $u = \frac{1}{2}(1+\alpha)n$. After all, of all possible values for u included under H_0, this choice would give the highest probability. Unfortunately, a sum/convolution of hypergeometric random variables does not have a probability mass function that is easy to compute. It can be expressed as

$$\mathbb{P}(T = t) = \sum_{x_2=0}^{q}\sum_{x_3=0}^{q}\cdots\sum_{x_k=0}^{q}\prod_{i=2}^{k}\mathbb{P}(X_i = x_i)\,\mathbb{P}\left(X_1 = t - \sum_{i=2}^{k} x_i\right).$$

This expression is derived in Lemma B.1 in the appendix. The intuition behind it is as follows: for all possible sequences $(x_2, x_3, \ldots, x_m)$, we calculate the probability of observing it, and then multiply this by the probability that X_1 is exactly t minus the sum of the sequence. This is what is implemented in Alg. 4: the recursion encapsulates the repeated sum, while the final computation of $\mathbb{P}(X_1 = t - \sum_{i=2}^{k} x_i)$ is done in lines 14 through 18. Numerically, it can be simplified somewhat – e.g., it is not necessary to consider sequences whose sum already exceeds t. However, to compute this probability it is inescapable that the number of operations is roughly proportional to q^k, which is very large even for moderate values of k. This is the intuition behind the if-statement in line 10 of Alg. 4: the exact computation is only performed if $q^{m-j} = q^k$ is small enough, i.e., below some threshold that depends on the node's processing power.

A less computationally expensive approach would to find an approximation for the distribution of T. Good candidates include a single hypergeometric (with parameters kq, ku, and kq instead of n, u, and q), a binomial (for which we draw *with* instead of *without* replacement), a Poisson (which approximates the binomial distribution), or a normally distributed (by the Central Limit Theorem) random variable. However, in the following we instead focus on the Cramér-Chernoff method [10], which is a general method for bounding the probability $\mathbb{P}(X \geq x)$ of a random variable X. The reasons for this are twofold. First, it gives us a strict upper bound on the probability of interest regardless of the parameter choice, which is safer than an asymptotically valid approximation. Second, it can easily be generalized to settings where the distribution of the random variables is slightly different. We have also investigated other methods for bounding the tail probabilities of T, e.g., the Chernoff-Hoeffding bound, and the approaches of [36] and [44]. However, we found that the Cramér-Chernoff bounds were the sharpest while still being computationally feasible.

The Cramér-Chernoff method's basis is Markov's inequality [10], which states that for any nonnegative random variable X and $x > 0$, it holds that

$$\mathbb{P}(X \geq x) \leq \mathbb{E}(X)/x.$$

It can then be shown [10] that for any $\lambda \geq 0$,

$$\mathbb{P}(X \geq x) = \mathbb{P}(e^{\lambda X} \geq e^{\lambda x}) \leq \mathbb{E}(e^{\lambda X})e^{-\lambda x} = e^{-(\lambda x - c_X(\lambda))},$$

where $c_X(\lambda) = \log(\mathbb{E}(e^{\lambda X}))$. Since this holds for all $\lambda \geq 0$, we can choose λ such that the bound is sharpest. Let $r_X(x) = \sup_{\lambda \geq 0}(\lambda x - c_X(\lambda))$. It then holds that $\mathbb{P}(X \geq x) \leq e^{-r_X(x)}$. In particular, if $T = X_1 + \ldots + X_k$, such that all X_i, $i \in \{1,\ldots,k\}$ are mutually independent and have the same probability distribution as X, then

$$c_T(\lambda) = \log(\mathbb{E}(e^{\lambda T})) = \log\left(\prod_{i=1}^{k}\mathbb{E}(e^{\lambda X_i})\right) = kc_X(\lambda)$$

and hence

$$r_T(kx) = \sup_{\lambda \geq 0}(\lambda kx - kc_X(\lambda)) = kr_X(x),$$

and therefore $\mathbb{P}(T \geq t) \leq e^{-kr_X(t/k)}$. Common names for

r_X include the *Legendre-Fenchel transform* [1] or the large-deviations *rate function* of X after its use in Cramér's theorem for large deviations [10], [22]. In our context, t/k is the average supporting stake accumulated per round. As shown in Lemma B.3 in the appendix, if t/k is higher than the expected worst-case supporting stake, which in our case is qu/n, then the probability that the adversary wins decreases exponentially in k. In fact, the function r_X represents the exponential rate at which this probability decays (hence the name "rate function").

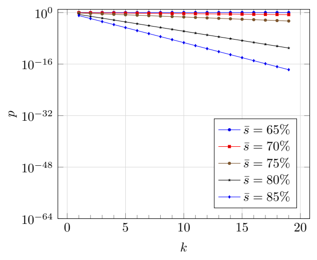
As an example, let $n = 1500$, $u = 1000$, and $q = 150$. We find that $r_X(\lfloor \frac{3}{4}q \rfloor) = r_X(112) \approx 2.50$. This means that the probability under the null hypothesis that we observe an average support of 75% of the committee for k rounds in a row decreases at least exponentially with a factor of $e^{-2.50} \approx 0.082$ per round. This is displayed in Fig. 4, for $\bar{s} = \frac{t}{kq} = 75\%$. After 15 rounds, the probability of observing support from, on average, 75% of the committee is below 10^{-16} under H_0.

The function $c_X(\lambda)$ does not have a convenient closed-form expression: it is equal to ${}_2F_1(-q, -u, -n, 1-e^{\lambda})$ [47], where ${}_2F_1$ is the Gaussian or ordinary hypergeometric function. Hence, the rate function has to be evaluated numerically. Since $c_X(\lambda)$ is convex on $\lambda \geq 0$ (see also Lemma B.2 in the appendix), there is only one local optimum and we can therefore use some variation of golden-section search to find it. The procedure in Alg. 4 is as follows: we first determine a search range for λ by widening a base interval until we observe that the function decreases at some point in the interval. We then iteratively narrow the interval until its width is below some accuracy threshold. Computation of the rate function is still not trivial, although it depends on the size of q: for $q = 150$, we have found that roughly 15 instances can be computed per second on a MacBook Pro with a 2.5 GHz Intel Core i7 processor, compared to $280/s$ for $q = 30$ and $0.6/s$ for $q = 750$. Still, although there is no theoretical limit on the number of blocks that can be committed in a single round via the *Commit* function in Alg. 4, in practice it is unlikely that more than several blocks are committed in a single round, as the evidence for committing is accumulated slowly. Furthermore, evaluations of $r_X(x)$ can be cached to speed up the *Commit* function in later rounds.

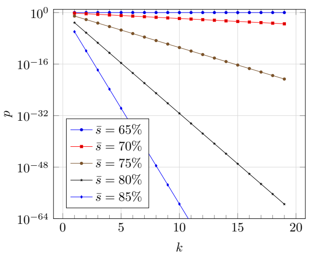
Fig. 4 depicts the evolution of the bounds if the same average supporting stake $\bar{s}$ is observed over k rounds consecutively. It is clear that, as expected, larger committee sizes mean that large deviations are less likely, and therefore that the user is able to commit sooner. Hence, there is a trade-off between security and efficiency. Even if $q = 30$ and on average 24 (i.e., $\bar{s} = 80\%$) stake units vote to support a block, then the p-value decreases by almost 75% per round (i.e., $e^{-r_X(24)} \approx 0.263$).

Although we have so far focused on a single test conducted in round m about a block B in slot j, in practice the test for B will be performed multiple times if there is not enough evidence to commit immediately. Each time the test is performed, there is some probability of error, so we need to account for this accumulation. The most straightforward way to achieve this is as follows: if we conduct the test for the ith time, we use $p^* \cdot \gamma^i$ for some $\gamma \in (0,1)$ as our threshold. The total probability of error after a potentially infinite number of trials is then bounded by $\sum_{i=1}^{\infty} p^* \gamma^i = \frac{\gamma}{1-\gamma} p^*$. For example, if

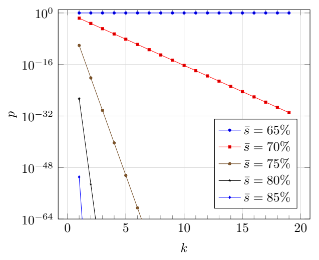
[1]Strictly speaking, only after broadening the range of the supremum.



(a) $q = 30$



(b) $q = 150$



(c) $q = 750$

Fig. 4: The Cramér-Chernoff bound p for the probability that a block B is overturned, assuming that B and its supporting subtree in the $k-1$ subsequent rounds contain the votes of a fraction $\bar{s}$ of the committee. In these figures, $n = 1500$, $u = 1000$, and q varies. Clearly, when the committee size is larger, the p-values vanish faster and nodes are able to commit sooner.

$\gamma = \frac{9}{10}$, then we need to multiply our probability threshold by $\frac{1-\gamma}{\gamma} = \frac{1}{9}$ (e.g., if we originally had set a threshold of 10^{-16} then we would now use $\approx 1.11 \cdot 10^{-17}$), and the threshold for the p-value would become lower (so harder to meet) by 10% each round. This is not a major obstacle: as mentioned before, even for a committee size of 30 and an average supporting stake of 80%, the p-value decreases considerably faster than by 10% per round, and the multiplicative factor of $1/9$ is met after only 2 additional rounds.

Using the above, we can prove the following theorem.

Theorem IV.1. *LaKSA achieves probabilistic safety.*

Proof: A safety fault occurs when two honest users on conflicting branches commit a block on their branch. If the honest stake is split evenly between the two branches, then the probability that both users commit is $(p^*)^2$, which is much lower than p^*. [2] In the case where the stake is *not* evenly split among the two branches, then if the user is on the weaker branch, the probability that she ever commits B is below p^*. If the user is instead on the stronger branch, then the probability that the honest users on the weaker branch ever commit is below p^*. Hence, the probability of both B and a conflicting block being committed is below p^* in all settings. Our safety property is therefore satisfied. ■

Long-Range Attacks. An additional benefit of the commitment scheme in LaKSA is its potential to act as a first line of defense against *long-range attacks*. A long-range attack occurs when the adversary obtains a set of keys that control a large

[2]It can be even lower if there is negative correlation between the stake on the two branches, e.g., if the implementation of the random beacon is such that the same committee is chosen for different blocks in the same round.

majority of stake in the distant past. Unlike PoW, blocks in PoS protocols are not computationally hard to create, which means that it is feasible for an adversary to create an extremely long branch by herself (e.g., 10^6 blocks or more). If the adversary forks the chain in the far past and creates a branch that is stronger than the main branch, then the attacker can convince honest nodes to revert a large number of their blocks in favor of adversarial blocks. We assume that the system's users will eventually be able to agree that the original branch is the "correct" branch after a period of out-of-band consultation. However, in the meantime honest nodes are vulnerable to double-spend attacks as they may accept transactions that conflict with transactions on the eventual main chain.

A user can mitigate long-range attacks by instructing her client to *finalize* committed blocks, i.e., to not revert them without a manual reset. However, if the node commits a block on the losing branch of a temporary fork, then it will remain out-of-sync with the rest of network until the user intervenes. This again leaves the node vulnerable to double-spend attacks, and its votes and blocks will not earn rewards on the main chain. It is therefore only profitable to use the above defense mechanism if its expected gains outweigh its costs. Although not all quantities involved in these costs and gains are clear a priori, the user *does* have a bound on the probability of committing a block on the losing branch of a fork, namely p^* as discussed above. In the following, let C_1, p_1, and C_2 be the user's guesses for the expected total cost of a long-range attack, the per-block probability of a long-range attack,[3] and the expected total cost of being out-of-sync with the network until a manual reset, respectively. Then the gains of this defense mechanism (i.e., nullifying the cost of long-range attacks) outweigh its costs if $C_1 p_1 > C_2 p^*$, i.e., if $p^* < \frac{C_1}{C_2} p_1$.

In practice, C_1 and C_2 will depend strongly on the nature of the user. For example, as witnessed by the recent 51% attacks on Bitcoin Gold and Ethereum Classic,[4] cryptocurrency exchanges are a valuable target for double-spend attacks – after all, exchanges routinely process high-value transactions and their actions (e.g., transactions on another blockchain) are often irreversible. These nodes can reflect this by choosing a high value for C_1. By contrast, a user who controls stake units will miss out on voting and block creation rewards when she is out-of-sync with the network, so she will judge C_2 to be much higher than C_1 if she is unlikely to be targeted by attackers.

In general, if the user deems long-range attacks to not be likely or costly, or if she deems the costs of being out-of-sync to be high, then she will judge the defense mechanism to be worth it only if p^* is low. As can be seen from Figure 4, the p-value of our test decreases rapidly over time, so even if p^* is extremely low, then blocks will be committed in a timely manner – e.g., for $q = 30$ and 24 supporting stake per block on average, p^* reaches $2^{-256} \approx 10^{-77.06}$ after only 133 rounds. Furthermore, the user is free to set a lower threshold for finalization than for regular commitment decisions, and compare the calculated p-value for each block to both thresholds. Hence, she can set the threshold for finalization very low, and still be safe from long-range attacks from fewer than 10^3 blocks in the past at negligible expected cost.

[3]To illustrate this: if the user expects long-range attacks to occur once per year, and if there are 10^6 blocks per year, then $p_1 = 10^{-6}$.

[4]See also the online references here and here.

B. Liveness

In our analysis we assume the partially synchronous network model, where after the GST the network is synchronous for t rounds. We follow the same setup in § IV-C and § IV-D. To prove liveness we use two lemmas (see App. C). In short, they state that with increasing t, after t rounds, a) the main chain includes at least $t(1-\alpha)q$ stake units (Lemma C.1), and b) the main chain includes m blocks that the honest participants voted for, where $t \geq m \geq \lceil t(1-2\alpha) \rceil$ (Lemma C.2). As stated in § II, honest nodes can be temporarily offline before but not after the GST, and permanently offline honest nodes are grouped with the adversarial nodes. Since $\alpha < \frac{1}{3}$, some $\varepsilon > 0$ must exist for which $\alpha = \frac{1}{3} - \varepsilon$, and we assume that γ is chosen such that $\gamma e^{-rx((\frac{2}{3}+\varepsilon)q)} < 1$, as we discuss below.

Theorem IV.2. *LaKSA achieves liveness.*

Proof: We show that the probability that within $t = 2t'$ rounds an honest node appends a block to the main chain and the block is committed by every honest node is overwhelming, with t increasing.

By using Lemma C.2, we obtain that the m' – which increases proportionally with t' – blocks of the main chain were supported by, and thus visible to, the honest majority. Now, to violate liveness, the adversary has to be elected as a leader for all of those contributed m' rounds.[5] The adversary is elected as a round leader with the probability proportional to her stake possession $\alpha < \frac{1}{3}$. Therefore, the probability that at least one honest node adds a block in those m' visible rounds is $1 - \alpha^{m'}$, which with increasing t' goes to 1.

Now, after the first t' rounds, the block will be on the main chain, thus honest nodes over the next t' rounds will be supporting the block by their stake of the total amount at least $t'(1-\alpha)q$ units (see Lemma C.1) with t' increasing. We assume that $\alpha < \frac{1}{3}$, which means that there exists some $\varepsilon > 0$ such that $\alpha = \frac{1}{3} - \varepsilon$. This means that the expected amount of supporting stake per round will be $(\frac{2}{3} + \varepsilon)q$, even if the adversarial nodes never vote. By the weak law of large numbers, the probability that the average supporting stake after t' rounds is not above $\frac{2}{3}Q$ goes to zero as t' goes to infinity. We know by Lemma B.3 that if the observed average supporting stake is greater than the mean, the rate function evaluated at this average is greater than 0. By substituting the threshold $\alpha = \frac{1}{3}$ into (3), we find that $u \leq \frac{2}{3}n$, and that the expected per-round supporting stake under the null hypothesis qu/n equals at most $\frac{2}{3}q$. Since we therefore observe more than what is expected under the null hypothesis on average, the per-round decay rate $e^{-rx((\frac{2}{3}+\varepsilon)q)}$ will be less than 1, and if γ (which can be set arbitrarily close to 1) is such that $\gamma e^{-rx((\frac{2}{3}+\varepsilon)q)} < 1$ then the upper bound on the test p-value goes to 0 as t' goes to infinity. As such, there will be some t' at which the p-value goes below p^*, which means that the user can commit. This completes the proof for liveness. ■

In practice, the length of t depends on the values of n, q, α, γ, and p^*, and can be derived from the bound given in § IV-A.

[5]In such a case the adversary could append blocks with no transactions, append them to a different (weaker) fork, or not publish them at all.

C. Fairness

We define $R(t)$ as the total expected reward of an honest node with β stake fraction during t synchronous rounds after GST. In the following, we analyze the fairness of LaKSA.

Theorem IV.3. $R(t) = \Theta(t)$

Proof: Every round (see Alg. 5), a node has the expected voter's reward $e_v = \beta q R_v$, and the expected leader's reward of $e_l = \beta(R_l + qR_i)$ (contributions of inclusion rewards R_i may vary and in the worst case e_l can be βR_l). As shown in Lemma C.2, honest nodes participate in n blocks of the main chain during t rounds, with n growing proportional to t. We assume that an adversary can censor the node's votes (removing her voter rewards) but that will not happen in $n(1-\alpha)$ rounds on average (i.e., when an honest leader is elected). Thus $n(e_l + e_v) \geq R(t) \geq n(e_l + e_v(1-\alpha))$, and given the bounds for n (Lemma C.2), the following holds

$$t(e_l + e_v) \geq R(t) \geq \lceil t(1-2\alpha) \rceil (e_l + e_v(1-\alpha)). \qquad \blacksquare$$

Another advantage of our reward scheme is that it rewards active nodes frequently. In every round, q stake units are rewarded for voting and one leader obtains the leader reward. The rewards are given uniformly at random and proportional to stake possession. In an example setting where $n = 10,000$, $q = 200$, and $l = 1$, even a node with a small stake possession $\beta = 1$ would receive a voter reward every 50 rounds and a leader reward every 10^4 rounds, on average. With a realistic round time $2\Delta = 5s$, this would be 250 seconds and 13.9 hours, respectively. By contrast, systems that award only leaders would not give these frequent voter rewards. For instance, in Bitcoin (where only leader awards exist and rounds last 10 minutes on average) the node would receive a reward every 70 days on average. By combining frequent rewards with fairness (Theorem IV.3), LaKSA minimizes the reward variance which is seen as the root cause of creating large mining pools which introduce centralization into the system.

D. Rewards and Incentives

To reason about rewards in LaKSA, we use a definition similar to the one by Pass and Shi in [41]. We call a protocol's reward scheme *β-coalition-safe* if, with overwhelming probability, no $\beta' < \beta$ fraction coalition can gain more than a multiplicative factor (1 + ε) of the total system rewards. Intuitively, it means that with overwhelming probability, for any coalition of nodes, the fraction of their rewards (within a time period) is upper-bounded by $(1+\rho)R(t)$, while a solo node is guaranteed to receive at least $(1-\rho)R(t)$. Thus, the multiplicative increase in the reward is $\frac{1+\rho}{1-\rho} \leq 1+\varepsilon$.

Using this definition, we show how relationships between rewards in LaKSA influence its coalition-safety. As shown in Theorem IV.3, the total reward of a node with β stake fraction during t synchronous rounds is between $n(e_l + e_v)$ and $n(e_l + e_v(1-\alpha))$. Thus $\frac{1+\rho}{1-\rho} = \frac{n(e_l+e_v)}{n(e_l+e_v(1-\alpha))}$ and the following holds: $\rho = \frac{\alpha e_v}{2e_l + e_v(2-\alpha)}$. We also have $\frac{n(e_l+e_v)}{n(e_l+e_v(1-\alpha))} \leq (1+\varepsilon)$ owing to $\frac{1+\rho}{1-\rho} \leq (1+\varepsilon)$, resulting in $\varepsilon \leq \frac{\alpha e_v}{e_l + e_v(1-\alpha)}$.

As we want to minimize ρ and ε, owing to $e_v = \alpha q R_v$, and $e_l = \alpha(R_l + qR_i)$, we wish to minimize $\frac{\alpha q R_v}{2(R_l+qR_i)+qR_v e_v(2-\alpha)}$ and $\frac{\alpha q R_v}{R_l+qR_i+qR_v e_v(1-\alpha)}$, thus, R_l should be large enough to make ρ and ε smaller, i.e., $R_l \gg R_v$ and $R_l \gg R_i$.

We suggest that R_v is higher than R_i. The intuition behind this choice is that as voter rewards R_v are received frequently when compared with inclusion rewards R_i received only by a leader. A too high R_i would contribute to a high reward variance and could incentivize nodes to join their stake into pools (as in PoW systems), introducing centralization risks. Therefore, with $R_v > R_i$, we propose the following relationships between rewards in LaKSA: $R_l \gg R_v > R_i$.

Our rewarding scheme (§ III-G), aims to incentivize a) voters to broadcast their votes immediately, b) leaders to broadcast their blocks immediately, and c) leaders to include all received votes in their blocks. We informally reason about its incentive-compatibility in a setting with synchrony, a single leader elected per round, and rational nodes (optimizing only for their rewards). If a voter does not publish its vote on time, it will not reach the round's leader, who subsequently cannot include this vote in her block, thus the voter loses her reward R_v. Similarly, if the leader does not publish her block immediately, it is not received by other nodes, who in the next round will vote for a virtual block, ignoring the late block and hence not awarding the rewards R_v and R_i to the leader. If the leader publishes blocks but without some of the received votes, then the leader is losing the inclusion reward R_i proportional to the stake of the ignored votes.

E. Scalability and Parametrization

In order to evaluate how different configurations influence LaKSA's performance and security, we have built a LaKSA simulator and conducted a series of experiments. For our simulations, we introduced 5000 active nodes with one stake unit each (thus $n = 5000$) and a single leader per round ($l = 1$). The number of nodes is comparable with the estimated number of all active Ethereum nodes [1]. To model the network latencies while propagating messages we used the data from https://ethstats.net/, which in real-time collects various information (like the block propagation time) from volunteer Ethereum nodes. In this setting, we ran simulations for different values of q and Δ, where each simulation run was for 10000 blocks. We measured the block and vote stale rates ψ_b and ψ_v, which respectively describe how many blocks and votes do not make it onto the main chain.

The obtained results are presented in Fig. 5. We see that if Δ is too short, then protocol performance decreases as stale rates for both blocks and votes are high. Besides influencing performance (fewer blocks denote lower throughput), high stale rates degrade security as it takes longer to commit a block (see § IV-A). However, when Δ is slightly increased, the stale rates improve significantly. We found that choosing Δ between 3-4 seconds may be good for the considered network conditions. We confirm our results in § V. Moreover, with increasing q, we see only a mild increase in the stale rates.

V. Implementation and Evaluation

Implementation. To test and evaluate LaKSA we fully implemented it. Our implementation is based mainly on Python. For implementing the p2p networking stack, we use the Twisted framework while we deployed Ed25519 signatures [7]. The

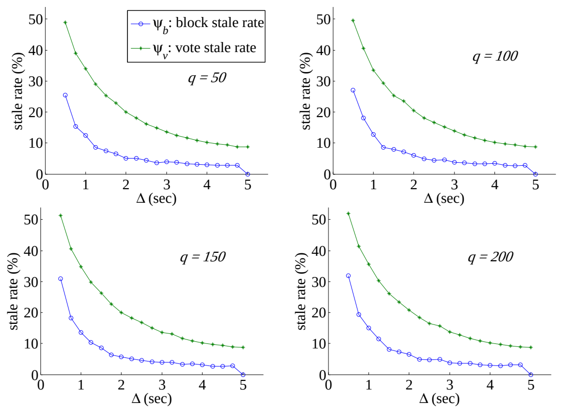


Fig. 5: Simulated stale rates.

advantage of doing so is that is produces short public keys and signatures (32 and 64 bytes, respectively). In our implementation, we have encoded a vote in 80 bytes, and votes are included within their respective blocks. Although the number of votes is linear, even for large values of q the introduced bandwidth overhead should be acceptable. For instance, 1000 support votes with our encoding would consume 80kB, which constitutes only 8% of a 1MB block or 4% of a 2MB block.

Evaluation. Equipped with our implementation we conducted a series of experiments in a real-world setting. We set up a testbed consisting of physical machines geographically distributed among 15 different locations from five continents. We used these machines to run 100 LaKSA nodes in total. We used a simple p2p flooding where every node peered with up to five peers. We first investigated the throughput of the network and we obtained an average end-to-end throughput of around 10 Mbps. To introduce a conservative setting and better express real-world heterogeneous network conditions we did not optimize our network by techniques like efficient message dissemination or geographical peer selection.

In our setting, all our 100 nodes are elected as voters (i.e., $q = 100$) while only one node per round is elected as leader. From the performance point of view, such a setting would be for instance equivalent to the setting when there are 2000 nodes in the system and $q/n = 5\%$. During the execution of our experiments, we noticed an imbalance between the durations of the protocol steps. Namely, votes constitute only a small part of the blocks that actually carry transactions, so if the nodes spend too much time waiting for votes then this would not allow us to saturate the network. To maximize the throughput we introduced separate waiting times for votes and blocks – i.e., Δ_1 and Δ_2 respectively, where $\Delta_1 < \Delta_2$, instead of a single value Δ for both.

We have run a series of experiments with different Δ_1, Δ_2, and block size parameters and our results are presented in Tab. I. Namely, we measure the following three performance indicators: the block and vote stale rates ψ_b and ψ_v as defined in § IV-E, and the *goodput* – i.e., the number of kilobytes available for potential applications per second. The block size introduces a natural trade-off between the latency and goodput. We observe that if we increase the block size from 1MB (which is the default for Bitcoin) to 2MB, the block stale rates remain similar and the goodput is roughly doubled. However, if we

Tab. I: Performance results for different parameters.

		1MB Blocks			2MB Blocks			4MB Blocks		
Δ_1	Δ_2	ψ_b	ψ_v	Gput.	ψ_b	ψ_v	Gput.	ψ_b	ψ_v	Gput.
s	s	%	%	KB/s	%	%	KB/s	%	%	KB/s
1.0	4.0	9.5	19.6	179	14.3	33.5	343	49.0	41.1	204
3.0	3.0	0	5.8	167	2.5	22.3	325	16.5	26.7	550
2.0	3.0	0	6.1	200	0	23.0	400	37.1	34.4	336
2.5	3.0	0	4.1	182	0	20.6	364	31.2	32.1	371
2.0	4.0	0	2.0	167	0	5.4	333	31.6	38.6	412
1.5	4.0	1.4	1.9	179	0	5.3	364	48.4	39.0	267
3.5	3.0	0	1.4	154	0	2.9	308	2.0	7.1	603
4.0	3.0	0	1.5	143	0	2.2	286	0	5.6	571

increase the block size to 4MB then the stale rates are so high for small Δ_1 that throughput is not meaningfully higher than for 2MB blocks. As depicted, LaKSA offers a goodput between 200 and 600 KB/s, with a round latency between 5 and 6.5 seconds, respectively. In Bitcoin, where the size of a 2-in-2-out transaction is around 450 bytes, this would roughly correspond to between 450 and 1300 transactions per second. In particular, we found $\Delta_1 = 1.5s$ and $\Delta_2 = 4.0s$ as a promising configuration for 2MB blocks.

Further, we investigated the computation costs incurred by LaKSA nodes. We used a single core of Intel i7 (3.5 GHz) CPU to compute vote and block processing times. On average, it takes only 53.25 μs to create a signed vote, 17.22 ms to create a new block with 100 votes, and 10.08 ms to validate a received block including 100 votes.

VI. Comparison to Algorand

In this section we provide a detailed comparison to Algorand [30], which is a closely-related PoS protocol. Algorand shares several similarities with LaKSA: 1) the protocol operates as a sequence of rounds, 2) leaders and committee members are selected in each round based on a random beacon that changes between rounds, 3) the likelihood that a node is selected as a leader or committee member is proportional to its number of stake units, 4) committee members vote on blocks proposed by the leaders, and 5) blocks are committed if they receive a sufficient number of votes.

Despite these similarities, there are two main differences between Algorand and LaKSA. The first is that instead of the cryptographic sampling procedure of Alg. 2, Algorand elects leaders and committee members in every round by running a VRF [38] over the round's random beacon. As a result, the number of leaders and committee members in each round is random instead of fixed, and the added variance makes block commitment less secure. Second, instead of the voting and sequential hypothesis testing procedures of Alg. 1 and 4, Algorand uses a bespoke Byzantine agreement protocol (BA$\star$) to commit blocks. However, BA$\star$ includes many steps during which no transactions are added to the blockchain, and the security analysis for BA$\star$ is more restricted than for LaKSA. There are other differences – e.g., rewards and fairness are not discussed in [30] – but due to space restrictions we only elaborate on the two main differences. A summary of the technical details of Algorand can be found in App. D.

We first investigate what the impact would be if Alg. 2 of LaKSA were replaced by Algorand's committee selection procedure, so that l and q are no longer fixed. The main consequence is that the additional variance due to the committee

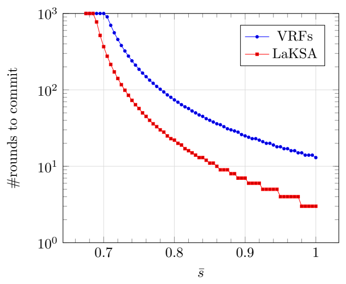


Fig. 6: Number of rounds needed before a block can be committed given an average supporting stake fraction per round $\bar{s}$ for fixed committee sizes (LaKSA) and VRFs. It takes around 3-4 times longer to commit using VRFs.

sizes makes it harder to make block commitment decisions using hypothesis testing. This difference in terms of variance is made explicit in Tab. III of App. F. In particular, the quantity of supporting stake per round X was previously hypergeometric (drawing q samples without replacement from a population of size n of which u support the user's branch), whereas in the setting with VRFs it is binomial with a considerably larger population size (drawing n samples such that each is in the committee with probability q/n and supportive of the user's branch with probability u/n). The variance is typically around 3-4 times higher in the new setting if the adversary controls one third of the stake (i.e., $u/n = \frac{2}{3}$).

As we can see from Fig. 6, the higher variance translates into block commitment times that are also roughly 3-4 times higher. For this experiment, we calculated numerically for different values of the average supporting stake per round $\bar{s}$ how many rounds it would take to commit in a setting where $p^* = 10^{-64}$, $n = 1500$, $u = 1000$, $q = 150$, and $\gamma = 0.99$ (as discussed in § IV-A). If X is binomially distributed, then $T = \sum_{i=n+1}^{m} X_i$ is also binomially distributed, so we do not need to use the Cramér-Chernoff method to bound the probability $\mathbb{P}(T \geq t)$ for the VRF setting because we can calculate it directly. However, the looseness of the Cramér-Chernoff bound is clearly offset by the VRF method's higher variance. It can also be seen that if $\bar{s}$ is high enough, then LaKSA can commit the blocks very quickly. For example, if $\bar{s}$ is at least equal to 98%, then LaKSA commits within 3 rounds despite the very high security requirement (i.e., a safety error probability per round of 10^{-64}). As long as $\bar{s}$ is above 86%, LaKSA commits within 10 rounds – with 5.5 seconds per round (see § V), this takes 55 seconds and therefore less time than the (on average) hour for 6 confirmations in Bitcoin.

Although the above demonstrates that VRF-based committee selection increases the number of rounds needed to commit a block using hypothesis testing, the value proposition of Algorand's BA$\star$ algorithm is that it commits blocks after a single round. However, there are several important caveats. The first is that even in the best-case scenario, the two phases of BA$\star$ take 2 steps each, so a round in Algorand takes at least $\lambda_{\text{PRIORITY}} + 4 \cdot \lambda_{\text{STEP}} = 85$ seconds (see also App. D), which is the same as 15 rounds of LaKSA. No transactions are added during the BA$\star$ steps, whereas in LaKSA transaction-containing blocks are confirmed by other transaction-containing blocks. The second is that the security analysis relies on larger committee sizes (10,000 stake units in the final round) and a weaker adversary (who only controls 20% of the stake) than LaKSA. Finally, each user in LaKSA is able to set her security threshold p^* individually depending on her risk tolerance, whereas Algorand has a fixed set of security parameters that determine block commitment.

VII. Related Work

Besides Algorand, Bitcoin's NC [39] has inspired a variety of alternative protocols that aim to build on its core strengths while mitigating its weaknesses, i.e., wastefulness, a slow and unclear commitment process, low transaction throughput, and a tendency to centralization. In this section, we compare LaKSA to some of its most prominent alternatives, and highlight instances where they copy drawbacks from NC or introduce new ones. An overview is presented in Tab. II.

Tab. II: Comparison of different blockchain protocols. HotStuff is a pure BFT design that can also be applied in a PoS setting.

Protocol	Node Selection	Block Commitment	#Leaders/Voters Per Round
NC [39]	PoW	Chain-Based	$\Theta(1)$
Peercoin [35]	PoW/PoS	Chain-Based	$\Theta(1)$
Tendermint [37]	PoS	BFT Voting	$\Theta(n)$
Casper FFG [14]	PoS	BFT Voting	$\Theta(n)$
HotStuff [46]	—	BFT Voting	$\Theta(n)$
Algorand [30]	PoS	BFT Voting	$\Theta(q)$
LaKSA	PoS	Chain-Based	$\Theta(q)$

The first PoS protocols that tried to extend or modify NC by taking stake into consideration resulted in hybrid PoS-PoW systems [35], [6]. The first of these, PPCoin (later renamed to Peercoin) [35], extends Bitcoin by also granting the ability to propose blocks to nodes who hold on to tokens instead of spending them (i.e., increasing their *coin age*). Bentov et al. instead propose a protocol [5] in which new blocks generate randomness that is used to coordinate the extension of the blockchain in the next several rounds. Other PoS protocols [26], [45] try to emulate the mining process of NC by using unique digital signatures. All of these systems mitigate Bitcoin's energy waste; however, they share multiple other drawbacks with NC, such as an unclear commitment process and a tendency to centralization via high reward variance.

To address the low throughput and unclear commitment process in NC, several other approaches replace NC's longest-chain rule by more established BFT consensus algorithms. The first approach to do so is Tendermint [37], in which leaders are elected proportionally to their stake using a round-robin procedure. The other nodes run a protocol that is based on Practical Byzantine Fault Tolerance (PBFT) [18] to agree on whether to commit the proposed block. Like most BFT protocols, PBFT works in the presence of f adversarial nodes as long as $n \geq 3f + 1$. After a block is proposed, all nodes vote across two phases (*prepare* and *commit*), and a block is committed if at least $2f + 1 \approx \frac{2}{3}n$ nodes vote to approve the block in both phases. To avoid attacks on leaders, which are known in advance, Tendermint proposes a lightweight network-level anonymity solution [37].

Casper FFG [14] is another PBFT-based PoS protocol, and functions as a finality-providing overlay for PoW or PoS blockchains. One of its main observations is that the two phases of PBFT can be encoded in a blockchain as vote messages for two subsequent blocks. Casper FFG is designed for an open setting with a dynamic set of nodes, and includes incentive-based protection against equivocation – i.e., misbehaving nodes risk losing their deposits. In both Tendermint and Casper FFG, all nodes send their vote message to all other nodes in each voting round – as such, when n grows large the communication complexity becomes a bottleneck. HotStuff [46] considerably reduces the message complexity of protocols such as Tendermint and Casper FFG through the use of threshold signatures. However, in HotStuff all participating nodes still vote in every round of the protocol.

Another approach to reduce the message complexity of BFT voting in each round is to draw a committee from the total set of nodes instead of requiring that all nodes vote. However, the safety properties of PBFT-based approaches do not straightforwardly carry over to this setting. The most obvious generalization of PBFT to random committees is to require that a block receives more than $\frac{2}{3}q$ votes instead of $\approx \frac{2}{3}n$ votes across two rounds for it to be committed, but this can lead to high safety fault probabilities. As discussed in § VI, Algorand [30] introduces a bespoke BFT algorithm called BA$\star$, which addresses the above problem by 1) requiring a higher fraction than $\frac{2}{3}$ of supporting votes to commit ($0.74q$ in the final round), 2) requiring large committee sizes, and 3) assuming limited adversarial strength.

A. *Message Complexity per Round*

NC has a message complexity of $\Theta(n)$ in each round, as leaders are elected without requiring any messages, and a leader sends her block to the other $n-1$ nodes. PBFT-like protocols such as Tendermint and Casper FFG have a message complexity of $\Theta(n^2)$, because all nodes except the leader vote in each round, and each vote is sent to every other node. In HotStuff, voters only send their votes to the leader, and the leader then sends the block including an aggregated signature to the nodes, resulting in a message complexity of $\Theta(n)$. Algorand has a message complexity of $\Theta(qn)$, as each committee member sends a message to the other $n-1$ nodes. Since the leader is not known – which is inescapable in the first phase, as the winners of the VRF-based election are unknowable by design – this cannot be reduced using the approach of HotStuff. In the implementation of LaKSA as presented in Alg. 1, the communication complexity is also $\Theta(qn)$ since each vote is sent to $n-1$ nodes. However, since the leader is known in each round, this can be reduced to $\Theta(n)$ using the approach of HotStuff. One adverse effect is that this makes the creation of virtual blocks in the case of offline or malicious leaders more complicated – essentially, voters must also send their votes to the leaders of later rounds.

B. *Other Related Work*

Brown-Cohen et al. [12] analyze longest-chain PoS protocols and show their fundamental limitations in preventing PoS-specific attacks. These results do not directly apply to LaKSA, since chain selection in LaKSA fully depends on votes (and not leader-driven chain length) and a GHOST-like rule which cannot be expressed in the framework of [12]; both aspects are mentioned by the authors of [12] as limitations. The Ouroboros family of protocols [34], [21], [3], [33] also uses a PoS approach with committee voting. Ouroboros shares the limitations of longest-chain PoS protocols as reported by Brown-Cohen et al. [12]. Moreover, the protocol leaves rewards and incentives as future work. Cardano, a cryptocurrency built upon Ouroboros, encourages users to either join or create stake pools [2] (see also [13]), thus encouraging centralization by design. Under the Ethereum 2.0 roadmap, Ethereum's legacy PoW chain will be phased out in favor of a novel PoS chain that features Casper FFG and sharding. The *beacon chain*, which will eventually coordinate the shard chains, was launched in December 2020. The beacon chain's block proposal mechanism [15] shares some similarities with LaKSA, e.g., it is also chain-based, with *attestations* taking the place of votes in the fork-choice rule. In the beacon chain protocol, time is divided into epochs that consist of 32 rounds – in each epoch, the set of active participants is pseudorandomly shuffled and divided among the epoch's slots to ensure that each node votes once per epoch.

Finally, several recent surveys [17], [4], [40] present an overview of recently proposed blockchain protocols.

VIII. Conclusions

In this work we have proposed LaKSA, a novel PoS consensus protocol dedicated to cryptocurrencies. Surprisingly, through its simple construction LaKSA provides a robust, scalable, and secure consensus mechanism. Our scheme extends the notion of probabilistic safety – in particular, clients base block commitment decisions on the probability of the block being reverted given the total observed support for it. These decisions are made more precise thanks to a lightweight committee voting scheme that allows large numbers of nodes to participate and express their beliefs about the blockchain.

In this work we have presented the core concept behind LaKSA and its properties. In the future, we plan to extend and analyze the system in a more dynamic setting and with adaptive adversaries. In particular, we believe that the ideas that are present in recent protocols [30], [20], [21] can be successfully applied in LaKSA, enhancing the protocol further. Another interesting research problem is to find an efficient election protocol combining the advantages of the proposed schemes (i.e., "secret" but deterministic election). We also plan to further study the economic aspects of the reward scheme and their influence on the security of the system.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by the Ministry of Education, Singapore, under its MOE AcRF Tier 2 grant (MOE2018-T2-1-111), by the National Research Foundation (NRF), Prime Minister's Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2016NCR-NCR002-028) and administered by the National Cybersecurity R&D Directorate, and by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Programmtic Programme (Award A19E3b0099).

References

[1] Ethereum mainnet statistics. https://www.ethernodes.org/.
[2] Shelley incentivized testnet - Cardano. https://staking.cardano.org/.

[3] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas. Ouroboros Genesis: Composable proof-of-stake blockchains with dynamic availability. In *ACM CCS*, 2018.

[4] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. SoK: Consensus in the age of blockchains. In *ACM AFT*, 2019.

[5] I. Bentov, A. Gabizon, and A. Mizrahi. Cryptocurrencies without proof of work. In *Financial Crypto*, 2016.

[6] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld. Proof of activity: Extending Bitcoin's proof of work via proof of stake. *IACR ePrint*, 2014.

[7] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2), 2012.

[8] S. Bojja Venkatakrishnan, G. Fanti, and P. Viswanath. Dandelion: Redesigning the Bitcoin network for anonymity. *ACM POMACS*, 1(1), 2017.

[9] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *IEEE SP*, 2015.

[10] S. Boucheron, G. Lugosi, and P. Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

[11] E. A. Brewer. Towards robust distributed systems. In *PODC*, 2000.

[12] J. Brown-Cohen, A. Narayanan, A. Psomas, and S. M. Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *ACM EC*, 2019.

[13] L. Brünjes, A. Kiayias, E. Koutsoupias, and A.-P. Stouka. Reward sharing schemes for stake pools. *arXiv preprint arXiv:1807.11218*, 2018.

[14] V. Buterin and V. Griffith. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[15] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang. Combining GHOST and Casper. *arXiv preprint arXiv:2003.03052*, 2020.

[16] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras. Incentives in Ethereum's hybrid Casper protocol. *IEEE ICBC*, 2019.

[17] C. Cachin and M. Vukolić. Blockchain consensus protocols in the wild. *arXiv*, 2017.

[18] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *OSDI*, 1999.

[19] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, et al. On scaling decentralized blockchains. In *Financial Crypto*, 2016.

[20] P. Daian, R. Pass, and E. Shi. Snow White: Robustly reconfigurable consensus and applications to provably secure proofs of stake. In *Financial Crypto*. 2019.

[21] B. David, P. Gaži, A. Kiayias, and A. Russell. Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT*, 2018.

[22] A. Dembo and O. Zeitouni. *Large Deviations Techniques and Applications*. Springer-Verlag, 1998.

[23] J. R. Douceur. The Sybil attack. In *International workshop on peer-to-peer systems*, 2002.

[24] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2), 1988.

[25] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[26] L. Fan and H.-S. Zhou. A scalable proof-of-stake blockchain in the open setting (or how to mimic Nakamoto's design via proof-of-stake). Technical report, IACR ePrint, 2017.

[27] G. Fanti, L. Kogan, S. Oh, K. Ruan, P. Viswanath, and G. Wang. Compounding of wealth in proof-of-stake cryptocurrencies. In *Financial Crypto*, 2019.

[28] G. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *ACM POMACS*, 2(2), 2018.

[29] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. In *ACM CCS*, 2016.

[30] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *ACM SOSP*, 2017.

[31] T. Hanke, M. Movahedi, and D. Williams. DFINITY technology overview series, consensus system. *arXiv*, 2018.

[32] N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*. John Wiley & Sons, 2005.

[33] A. Kiayias and A. Russell. Ouroboros-BFT: A simple Byzantine fault tolerant consensus protocol. *IACR Cryptol. ePrint Arch.*, 2018:1049, 2018.

[34] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, 2017.

[35] S. King and S. Nadal. PPCoin: Peer-to-peer crypto-currency with proof-of-stake. 2012.

[36] A. Klenke and L. Mattner. Stochastic ordering of classical discrete distributions. *Advances in Applied Probability*, 42(2), 2010.

[37] J. Kwon. Tendermint: Consensus without mining. 2014.

[38] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *IEEE FOCS*, 1999.

[39] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[40] C. Natoli, J. Yu, V. Gramoli, and P. Esteves-Verissimo. Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. *arXiv*, 2019.

[41] R. Pass and E. Shi. Fruitchains: A fair blockchain. In *ACM PODC*, 2017.

[42] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in Bitcoin. In *Financial Crypto*, 2015.

[43] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable bias-resistant distributed randomness. In *IEEE SP*, 2017.

[44] K. Teerapabolarn. Binomial approximation for a sum of independent hypergeometric random variables. *Global Journal of Pure and Applied Mathematics*, 4(5), 2015.

[45] X. Wang, G. Kamath, V. Bagaria, S. Kannan, S. Oh, D. Tse, and P. Viswanath. Proof-of-stake longest chain protocols revisited. *arXiv*, 2019.

[46] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *ACM PODC*, 2019.

[47] D. Zwillinger and S. Kokoska. *CRC standard probability and statistics tables and formulae*. CRC Press, 2000.

Appendix A
Cryptographic Sampling Proof

Definition 1 (Pseudorandom sampling)**.** *PRF is a pseudorandom sampling and mapping $\{0,1\}^n \times \{0,1\}^s \rightarrow \{0,1\}^n$ if it is collision-resistant and satisfies the following requirements: 1) For any $k \in \{0,1\}^s$, PRF is a bijection from $\{0,1\}^n$ to $\{0,1\}^n$. 2) For any $k \in \{0,1\}^s$, there is an efficient algorithm to evaluate $PRF_k(x)$. 3) For any PPT distinguisher $\mathscr{D}$:*

$$|Pr(\mathscr{D}^{PRF_k(\cdot)}(1^n)=1) - Pr(\mathscr{D}^{f_n}(1^n)=1)| < negl(s), \quad (4)$$

where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and f_n is chosen uniformly at random from the set of permutations on n-bit strings.

Corollary 1. *If the output of $PRF_r(\cdot)$ is indistinguishable from the uniform distribution, then the result of $PRF_r(\cdot)\%N$ for any N is indistinguishable from the uniform distribution.*

In particular, a pseudorandom sampling family is a collection of pseudorandom functions, where a specific sampling may be chosen using a key as a *salt*.

Lemma A.1. *We say that PRF is an unpredictable pseudorandom sampling, if no PPT adversary can distinguish the unit-pair from a uniform random distribution.*

Proof: For all PPT distinguishers $\mathscr{D}$:
$|Pr(\mathscr{D}^{PRF_k(\cdot)}(1^n) = 1) - Pr(\mathscr{D}^{f_n}(1^n) = 1)| < negl(s)$,

We define the following two distributions. $\mathscr{X}$ is the distribution $o_i^1, o_i^2, \cdots, o_i^S$, where $o_i^j \leftarrow PRF_{r_s}(i\|role)$, and $\mathscr{Y}$ is the uniform distribution. Then the two distributions $\mathscr{X}$ and $\mathscr{Y}$ are computationally indistinguishable. Below, we prove it by contradiction.

We assume that $\mathscr{D}$ is a PPT adversary who can distinguish $\mathscr{X}$ from $\mathscr{Y}$ with non-negligible advantage. For $1 \le i \le S+1$, S is the number of stake units, we introduce intermediate distributions $\mathscr{X}_i$ that are given by $\overline{h}^1, \cdots, \overline{h}^{i-1}, h^i, \cdots, h^S$, where h^i is as above and $\overline{h}^i$ is uniformly chosen from $\mathbb{Z}_q$. Hence we obtain $\mathscr{X}_1 = \mathscr{X}$ and $\mathscr{X}_{S+1} = \mathscr{Y}$. By assumption, $\mathscr{D}$ can distinguish $\mathscr{X}_1$ from $\mathscr{X}_{S+1} = \mathscr{Y}$ with noticeable (or overwhelming) advantage ε and so, by a standard hybrid argument, there is some i such that $\mathscr{D}$ can distinguish $\mathscr{X}_i$ from $\mathscr{X}_{i+1}$ with some noticeable advantage at least ε/S. It is then easy to see that $\mathscr{D}$ gives a distinguisher. That is, Eq. 4 does not hold. By assumption, no such distinguisher exists, and hence the lemma is proved. ■

Corollary 2. *If the pseudorandom sampling distribution $PRF_r(1\|role)\|\cdots\|PRF_r(Q\|role)$ is indistinguishable from the uniform distribution, then the sampling distribution $PRF_r(1\|role)$ % $Len(tmp_1)\|\cdots\|PRF_r(Q\|role)$ % $Len(tmp_Q)$ is indistinguishable from uniform, where tmp_i for $i \in [1,Q]$.*

We remark that tmp_i is dynamically changing for $i \in [1,Q]$, but it does not effect the indistinguishability result because $PRF_r(i\|role)$ is pseudorandom, so $PRF_r(i\|role)\%N$ is also pseudorandom.

Appendix B
Safety Analysis Lemmas

Lemma B.1. *Let $X_1, \ldots, X_k$ be independent and identically distributed, such that each X_i takes values on $\{0,1,\ldots,q\}$. Let $T_k = \sum_{i=1}^k X_i$. Then*

$$\mathbb{P}(T_k = t) = \sum_{x_2=0}^{q}\sum_{x_3=0}^{q}\cdots\sum_{x_k=0}^{q}\prod_{i=2}^{k}\mathbb{P}(X_i = x_i)\mathbb{P}\left(X_1 = t - \sum_{i=2}^{k} x_i\right).$$

Proof: We prove this using mathematical induction. For the base case, we have that $\mathbb{P}(T_2 = t) = \sum_{x_2=0}^{q}\mathbb{P}(X_2 = x_2)\mathbb{P}(X_1 = t - x_2)$ from the definition of the convolution of two random variables. For the induction step we have that

$$\begin{aligned}\mathbb{P}(T_k = t) &= \sum_{x_k=0}^{q}\mathbb{P}(X_k = x_k)\mathbb{P}(T_{k-1} = t - x_k)\\ &= \sum_{x_k=0}^{q}\mathbb{P}(X_k = x_k)\sum_{x_2=0}^{q}\sum_{x_3=0}^{q}\cdots\sum_{x_{k-1}=0}^{q}\\ &\quad \prod_{i=2}^{k-1}\mathbb{P}(X_i = x_i)\mathbb{P}\left(X_1 = t - x_k - \sum_{i=2}^{k-1} x_i\right)\\ &= \sum_{x_2=0}^{q}\sum_{x_3=0}^{q}\cdots\sum_{x_k=0}^{q}\prod_{i=2}^{k}\mathbb{P}(X_i = x_i)\mathbb{P}\left(X_1 = t - \sum_{i=2}^{k} x_i\right),\end{aligned}$$

where the first equality follows the definition of the convolution of two random variables, the second equality is the induction step, and the third equality holds because we are allowed to interchange the summations (as they are all over a finite set of elements). This proves the lemma. ■

The following well-known result about the convexity of c_X is needed at two different points in the text (namely for the validity of golden-section search, and for Lemma B.3).

Lemma B.2. *Let X be a random variable. Its cumulant-generating function $c_X(\lambda) = \log(\mathbb{E}(e^{\lambda X}))$ is convex.*

Proof: See, e.g., [10], or Cosma Shalizi's lecture notes on stochastic processes (week 31), which can be found online. ■

The following lemma asserts that when the per-round supporting stake does not exceed the expected value of our distribution (which in our case is given by qu/n), then we cannot establish a meaningful bound using the Cramér-Chernoff method. However, if it *does*, then the rate function is positive, which means that the bound (and therefore the p-value) goes to zero as the number of rounds increases.

Lemma B.3. *Let X be a random variable. Then its large-deviations rate function $r_X(t)$, defined as $r_X(t) = \sup_{\lambda \ge 0}(\phi_t(\lambda)) = \sup_{\lambda \ge 0}(\lambda t - c_X(\lambda))$, has the following properties: 1) if $t \le \mathbb{E}(X)$, then $r_X(t) = 0$, and 2) if $t > \mathbb{E}(X)$, then $r_X(t) > 0$.*

Proof: By definition, $c_X(0) = \log(1) = 0$. Hence, $\phi_t(0) = 0$ for all t, so by the definition of the supremum it must hold that $r_X(t) = \sup_{\lambda \ge 0}(\phi_t(\lambda)) \ge 0$. In the following, we will investigate $\phi_t'(0)$, i.e., the derivative of $\phi_t(\lambda)$ at $\lambda = 0$. If $\phi_t'(0) \le 0$, then by the convexity of c_X we know that $\phi_t'(\lambda) \le 0$ for all $\lambda \ge 0$, which proves property 1. However, if $\phi_t'(0) > 0$ then there must exist a point $\lambda^* > 0$ such that $\phi_t(\lambda^*) > \phi_t(0) = 0$, and the supremum must at least be equal to this value. Hence, we can demonstrate both properties of the lemma using $\phi_t'(0)$.

We find that

$$\phi_t'(\lambda) = \frac{d}{d\lambda}\left(\lambda t - \log(\mathbb{E}(e^{\lambda X}))\right) = t - \frac{\frac{d}{d\lambda}\mathbb{E}(e^{\lambda X})}{\mathbb{E}(e^{\lambda X})}$$

because of the chain rule. Hence,

$$\phi_t'(0) = t - \left.\frac{\frac{d}{d\lambda}\mathbb{E}(e^{\lambda X})}{\mathbb{E}(e^{\lambda X})}\right|_{\lambda=0} = t - \mathbb{E}(X)$$

because $\mathbb{E}(e^0) = 1$, and $\left.\frac{d}{d\lambda}\mathbb{E}(e^{\lambda X})\right|_{\lambda=0} = \mathbb{E}(X)$ because of the fundamental property of the moment-generating function[6] that $\frac{d^n}{d^n\lambda}\mathbb{E}(e^{\lambda X})\Big|_{\lambda=0} = \mathbb{E}(X^n)$.

Obviously, if $\phi_t'(\lambda) = t - \mathbb{E}(X)$ then $\phi_t'(0) \leq 0$ if $t \leq \mathbb{E}(X)$ and $\phi_t'(0) > 0$ otherwise, which proves the lemma. ■

Appendix C
Liveness Analysis Lemmas

Lemma C.1. *The main chain after t rounds includes at least $t(1-\alpha)q$ stake units while t increases.*

Proof: While t increases, the expected number of "honest" stake units elected per round is $(1-\alpha)q$. When the network is synchronous, honest nodes vote for the same blocks and their votes are delivered on time. Thus, with the failing (i.e., not voting) adversary, the strongest chain will obtain $t(1-\alpha)q$ of the supporting stake. An adversary can cause honest nodes to change their current main chain, e.g., by showing a stronger fork, but according to our chain selection rule (§ III-D) such a chain would need to contain at least the same amount of supporting stake as their main chain. Thus, the final main chain would include at least $t(1-\alpha)q$ stake units. ■

Lemma C.2. *With increasing t, the main chain after t rounds includes m blocks that the honest participants voted for, where*

$$t \geq m \geq \lceil t(1-2\alpha) \rceil.$$

Proof: Following Lemma C.1, honest nodes after t would agree on the main chain C that has at least $t(1-\alpha)q$ stake units. Let us assume that after t rounds there exists an adversarial chain C' with total stake of $t\alpha q + s$, where $t\alpha q$ is adversarial stake and s is the stake shared between C and C' (i.e., the chains have common blocks/prefix). C' can overwrite C only if: $t\alpha q + s \geq t(1-\alpha)q$, giving the bound for the shared stake: $s \geq t(1-\alpha)q - t\alpha q$. As every block contains maximally q stake units, s can be contributed in minimum $\lceil \frac{t(1-\alpha)q - t\alpha q}{q} \rceil$ blocks shared between C and C'. Thus, $n \geq \lceil t(1-2\alpha) \rceil$.

Note, that n is upper bound by $t \geq n$, as the main chain cannot contain more than t blocks in t rounds. ■

Appendix D
Technical Details of Algorand

In each round of Algorand, each node uses a VRF with the user's private key to calculate a hash of the random beacon concatenated with a "role" index. Each round in Algorand consists of two phases, and there are three roles: block proposer, initial ("step") committee member, and final committee member. The value of the calculated hash determines how many of the node's stake units qualify for each role. For any node, let N be the number of stake units that it controls, and N_{ROLE} be the number of stake units that are assigned to a given role. Alg. 1 of [30] then ensures that N_{ROLE} is binomially distributed with $\mathbb{P}(N_{\text{ROLE}} = n) = \binom{N}{n} {p_{\text{ROLE}}}^{n}(1-p_{\text{ROLE}})^{N-n}$, where p_{ROLE} is the probability for each single stake unit that it is assigned to ROLE. The probabilities are chosen such that the expected total numbers of block proposers, initial committee members, and final committee members per round equal τ_{PROPOSER}, τ_{STEP}, and τ_{FINAL}, respectively.

[6]https://en.wikipedia.org/wiki/Moment-generating_function

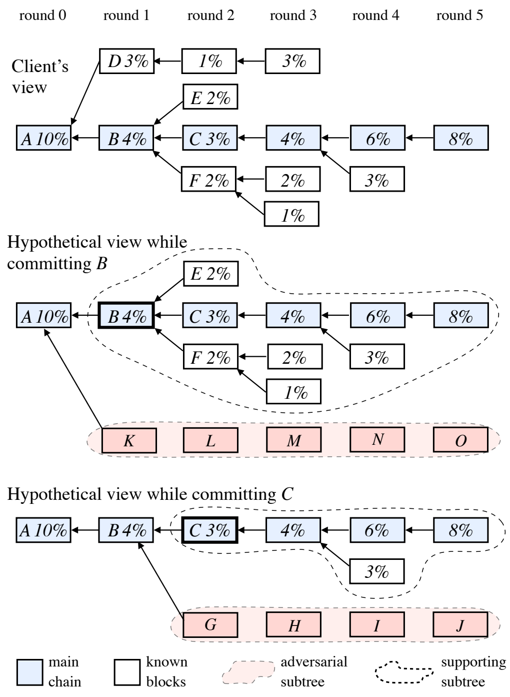


Fig. 7: An example of the client's view of the blockchain and possible adversarial subtrees. (Percentage of a block denotes the stake directly supporting this block.)

Each leader proposes a block such that there is a priority relation between the proposed blocks that is obvious to all nodes. The committee members wait a fixed amount of time per round ($\lambda_{\text{PRIORITY}}$) to receive blocks. Next, they initiate Phase 1 in which they vote for the highest-priority block that they are aware of (Alg. 7 of [30]). If a block receives more than $T_{\text{STEP}} \cdot \tau_{\text{STEP}}$ votes across two rounds of voting ("steps"), it advances to Phase 2. Phase 2 (Alg. 8 of [30]) consists of a sequence of steps in which the initial committee members either vote for the block from Phase 1 or an empty block. If either option receives more than $T_{\text{STEP}} \cdot \tau_{\text{STEP}}$ votes during a step, it is subjected to a final vote among the final committee members. If the block receives more than $T_{\text{FINAL}} \cdot \tau_{\text{FINAL}}$ votes in the final vote, then it is committed by the nodes. Each step in either phase of BA⋆ takes λ_{STEP} time units. The Algorand white paper provides a set of benchmark parameters for BA⋆ (see Fig. 4 of [30]) to ensure that the probability of a safety fault is sufficiently small (less than 10^{-7} over 1000 rounds) if the adversary controls at most 20% of the stake: $\lambda_{\text{PRIORITY}} = 5$ seconds, $\lambda_{\text{STEP}} = 20$ seconds, $\tau_{\text{PROPOSER}} = 26$, $\tau_{\text{STEP}} = 2000$, $\tau_{\text{FINAL}} = 10000$, $T_{\text{STEP}} = 0.685$, and $T_{\text{FINAL}} = 0.74$.

Appendix E
Block Commitment Example

To illustrate the block commitment process (§ III-E) better we present a blockchain example in Fig. 7 where it holds that $q = \frac{1}{10}n$ and a client wishes to commit the block C which is on the main chain (denoted in blue). To overwrite the block, an adversary would need to overwrite C's or B's supporting subtree. However, since block B must have already been committed, we do not need to consider any subtree starting from A (although we show it in the figure). As described, the client computes the p-value for the hypothesis that more stake units contribute to the adversarial subtree than to the supporting subtree under the worst-case conditions, and commits if this is low enough.

In our case, if the client wishes to commit C, then the adversarial subtree that could overwrite the supporting tree directly originates from block B and has $k = 4$ potential blocks, i.e., G, H, I, J. (We do not consider the topologies of adversarial subtrees, as only the stake that they aggregate decides their strengths.) The supporting subtree has 24% of the stake. The missing stake in this case is $4 \cdot 10\% - 24\% = 16\%$. However, the exact amount of missing stake that contributes to an adversarial chain may depend on the random beacon implementation (e.g., block J and the main chain block in the same round may have different committees if the beacon depends on some information in the blocks from the preceding rounds). Hence, we do not use this information directly. To be able to conclude with some degree of certainty that enough stake units are supporting B, we need the supporting stake to be comfortably above $\frac{2}{3}kq$. If the calculated p-value is acceptable for the client, the client commits B.

Appendix F
Election via VRF

A verifiable random function (VRF) [38] allows for the generation of a pseudorandom output from a message and a secret key. The output can then be publicly verified by any party with the corresponding public key. Algorand [30] proposed a VRF-based method for committee and leaders election dedicated to PoS systems. The core of the procedure is the cryptographic sortition algorithm (see § 5 of [30]). Every node runs the procedure locally to find out how many of its stake units were sampled within the round. The likelihood of being elected is proportional to the stake possession. This is similar to our cryptographic sampling construction (§ III-C), and Algorand's sortition can be applied in LaKSA almost directly, as we sketch below. The main voting procedure is similar as previously, i.e., as in Alg. 6. The main difference is to use the sortition algorithm to check whether the node is a leader and voter in the round. Leaders and voters add proofs to their blocks and votes, proving that they were indeed elected in the given round. These proofs have to be checked by other nodes by running *VerifyRole*.

Alg. 6: The voting procedure with VRF election.

```
function VotingRound(i)
    $r \leftarrow RoundBeacon(i)$; $w \leftarrow stake[pk]$;
    $(h,\pi,s) \leftarrow Sort(sk,r,q,\text{'vote'},w,n)$; // Alg. 1 in [30]
    if $s > 0$ then // check if I'm a voter
        $B_{-1} \leftarrow MainChain().lastBlk$; // get last block
        $\sigma \leftarrow Sign_{sk}(i \| H(B_{-1}) \| h \| \pi \| s)$;
        $v \leftarrow (i, H(B_{-1}), h, \pi, s, pk, \sigma)$; // support vote
        Broadcast($v$);
    Wait($\Delta$); // meantime collect and verify support votes
    $(h,\pi,s) \leftarrow Sort(sk,r,L,\text{'lead'},w,n)$; // Alg. 1 in [30]
    if $s > 0$ then // check if I'm a leader
        $B_{-1} \leftarrow MainChain().lastBlk$; // possibly different block
        $V \leftarrow \{v_a, v_b, v_c, ...\}$; // received $B_{-1}$'s support votes
        $r_i \leftarrow Random()$;
        $\sigma \leftarrow Sign_{sk}(i \| r_i \| H(B_{-1}) \| F \| V \| Txs \| h \| \pi \| s)$;
        $B \leftarrow (i, r_i, H(B_{-1}), F, V, Txs, h, \pi, s, pk, \sigma)$; // new block
        Broadcast($B$);
    Wait($\Delta$); // wait for the next round
function VerifyRole(r, role, pk, h, π, s, size) // vrfy. leader/voter
    $w \leftarrow stake[pk]$;
    $s' \leftarrow VerifySort(pk,h,\pi,r,size,role,w,n)$; // Alg. 2 in [30]
    return $s = s'$;
```

The main advantage of this solution is that voters and leaders are unpredictable and become known only after publishing their messages. This eliminates some classes of adaptive attacks as an adversary cannot target nodes at the beginning of the round. On the other hand, q and l are not constant in this setting and become random variables instead. In particular, if X is the number of stake units per round, then the variance in the hypergeometric distribution of § IV is given by

$$\mathrm{Var}_{\mathrm{HPG}} = q\frac{u}{n}\frac{n-u}{n}\frac{n-q}{n-1}.$$

By contrast, if we use VRFs then every stake unit is chosen as part of the committee with probability $\frac{q}{n}$ and part of the supporting branch with probability $\frac{u}{n}$. Hence, the number of supporting stake drawn using VRFs in a single round is binomial with sample size n and probability $u/n \cdot q/n$. The corresponding variance is

$$\mathrm{Var}_{\mathrm{BIN}} = n\frac{uq}{n^2}\left(1 - \frac{uq}{n^2}\right).$$

The difference between these two quantities is illustrated in Tab. III, which contains for different selections of n, q, and u the mean of X (which equals qu/n for both distributions), the variances of the hypergeometric distribution ($\mathrm{Var}_{\mathrm{HPG}}$) and the binomial distribution ($\mathrm{Var}_{\mathrm{BIN}}$), and the ratio of the latter to the former. We observe that the variance of the binomial distribution is consistently between 3 and 4 times larger than for the hypergeometric distribution.

Tab. III: Means and variances of the two different probability distributions (hypergeometric and binomial) for X for different parameters.

n	u	q	mean	$\mathrm{Var}_{\mathrm{HPG}}$	$\mathrm{Var}_{\mathrm{BIN}}$	ratio
150	100	15	10.0	3.02	9.33	3.09
150	100	75	50.0	8.39	33.33	3.97
1500	1000	75	50.0	15.84	48.33	3.05
15000	10000	75	50.0	16.58	49.83	3.0
1500	1000	750	500.0	83.39	333.33	4.0
15000	10000	2000	1333.33	385.21	1214.81	3.15