Subspace: A Solution to the Farmer's Dilemma

Jeremiah Wagstaff
Subspace Labs
Palo Alto, California
jeremiah@subspace.network

***Abstract*—In an effort to make blockchains more energy-efficient, egalitarian, and decentralized, several new protocols employ consensus based on *Proofs-of-Capacity* (PoC), which replace compute-intensive mining with storage-intensive farming. We observe that PoC consensus introduces a unique mechanism design challenge, referred to as *the farmer's dilemma*, which suggests that existing constructions are not actually incentive compatible. Simply put, farmers must decide whether to allocate scarce storage resources towards *either* maintaining the chain state and history *or* maximizing the amount of space they pledge towards consensus. Rational farmers will always choose the latter, at best becoming light clients, while at worst encouraging pooled farming under a few trusted operators. To resolve this dilemma, we introduce *Subspace*, a PoC blockchain in which farmers maintain neither the state nor the history, while retaining the security properties and decentralization benefits of a full node. Consensus in Subspace is based on proofs of replicated storage of the history of the blockchain itself. Farmers store the history collectively, many times over, with each farmer storing as many replicas as their disk space allows. Consensus and computation are then decoupled, such that farmers only propose an ordering for transactions, while staked executor nodes maintain the state and compute transitions. This separation of concerns significantly reduces the storage and compute overhead needed to operate a farmer, even in an Ethereum-style execution model, allowing for high levels of participation in consensus by ordinary users with commodity hardware.**

I. Background

Nakamoto-style blockchains, such as Bitcoin [1] and Ethereum [2], [3], combine the longest-chain fork-choice rule with a *proof-of-work* (PoW) mining puzzle. These systems are provably secure, with respect to safety and liveness, given an honest majority of miners [4]. Unlike legacy Byzantine Fault Tolerant (BFT) consensus algorithms, participation is both permissionless and scalable. These properties are the standard against which all new blockchain consensus protocols are measured. Unfortunately, the security afforded by PoW comes at a massive cost in electricity. Collectively, miners on Bitcoin and Ethereum consume the energy budget of a medium-sized country, with these numbers steadily increasing as more capital flows into the system. This raises the critical question of whether cryptocurrencies can reach wide scale adoption without adding more fuel to the fire of global warming.

Moreover, while mining was originally envisioned as a democratic and egalitarian process, as expressed by *one-CPU-one-vote*, it quickly became a highly commoditized and centralized enterprise. Today participation in Bitcoin mining instead follows *one-ASIC-one-vote*, assuming a miner also has access to low-cost electricity. Ethereum mining sought to circumvent this by adopting *one-GPU-one-vote*, but this too has proven susceptible to special purpose hardware and still has the tendency to concentrate in regions with low-cost electricity. This raises another key question of whether or not existing cryptocurrencies are actually decentralized, or if we have simply substituted one trusted third-party (financial institutions) for another (mining pools).

These challenges have served as a rallying cry for a diverse group of hackers, researchers, and engineers who have sought to design a sustainable blockchain that holds true to Nakamoto's vision for a more democratic and decentralized future. The most well-known solution to this problem is *proof-of-stake* (PoS), which employs a system of *virtual mining* based on one's wealth, under the adage *one-coin-one-vote*. While PoS clearly solves the sustainability problem, it does not hold true to Nakamoto's vision. It instead reflects a *permissioned* and *plutocratic* alternative, which also exhibits strong tendencies towards centralization. In fact, PoS systems serve to magnify the existing wealth disparity in cryptocurrencies, which are already significantly larger than historically high disparities in global fiat wealth distribution, effectively serving to make the rich even richer.

What is instead needed is a cryptographic proof system based on an underlying resource that is already massively distributed and which does not lend itself to special-purpose hardware. Enter *proof-of-capacity* (PoC)[1], which replaces compute-intensive mining with storage-intensive *farming*, under the maxim *one-disk-one-vote*. Disk-based consensus seems like an obvious choice, as storage hardware has long been commoditized, consumes negligible electricity, and exists in abundance across end-user devices. As it turns out, implementing a PoC such that it does not devolve back into PoW, without resorting to a *permissioned* model, is highly non-trivial, as witnessed by the paucity of live chains to date. Moreover, all existing PoC blockchain designs fail to address a critical mechanism design challenge, to which we turn next.

II. The Farmer's Dilemma

Observe that in any PoC blockchain a farmer is, by-definition, incentivized to allocate as much of its scarce storage resources as possible towards consensus. Contrast this with the desire for all full nodes to reserve storage for maintaining

Research conducted with support from NSF-SBIR Grant 1844037

[1]We use this as an umbrella term, capturing proofs of space, storage, replication, and space-time

both the current state and history of the blockchain. These competing requirements pose a challenge to farmers: do they adhere to the desired behavior, retaining the state and history, or do they seek to maximize their own rewards, instead dedicating all available space towards consensus? When faced with this *farmer's dilemma* rational farmers will always choose the latter, effectively becoming light clients, while degrading both the security and decentralization of the network. This implies that any PoC blockchain would eventually consolidate into a single large farming pool, with even greater speed than has been previously observed with PoW and PoS chains.

Recall that in any Nakamoto-style blockchain, a new consensus node must synchronize the chain state from genesis, in order to be assured they are actually on the longest valid chain, which implies the availability of the chain history. If a large fraction of nodes stores the history, this data will be readily available, and the network may be considered decentralized. However, as time goes by and the history grows, the storage burden on all full nodes grows as well, and some nodes may choose to prune the history, instead only storing the current state of the chain. This trend was already clear in the Bitcoin network as early as 2014 [5]. If full nodes do not store the history, new nodes must instead rely on altruistic archival nodes or third-party data stores for initial synchronization, resulting in a more centralized network. In a PoC blockchain farmers have nothing to gain by storing the history, but clearly stand to lose out on block rewards, especially as the history grows, consuming a larger fraction of their available disk space.

In order to extend the longest valid chain and collect fees for valid transactions, a farmer must maintain the memoized state of the chain. As the state is often too large to reside in memory, it too must compete with consensus for precious disk space. While perhaps negligible for low-throughput UTXO style chains, state storage is significant for any EVM style chain, or any chain which seeks base layer scalability. Furthermore, all farmers are also required to compute the state transition for each new block as part of the ongoing verification process, imposing a non-negligible computational overhead, which conflicts with the desire for farming to be a lightweight task. The farmer's dilemma then serves to exacerbate the well-known *verifier's dilemma*, by further raising the opportunity cost of verification [6].

If a farmer is willing to adopt a weaker security model, they may instead join a trusted farming pool, whereby they delegate transaction verification and block proposing functions to an operator, while the farmer focuses solely on evaluating the block challenge against their plots[2]. This has the added benefit of drastically reducing the computational overhead required to participate in consensus, which fits with the ideal of many small farmers pledging unused disk space on their home computers. When a farmer finds a valid solution to the block challenge, they send it to the pool operator, who forges the new block in return for a portion of the block reward. As long as the fee is lower than the opportunity cost of local block production, a rational farmer would always choose to join a pool. In PoW blockchains this choice is largely dictated by a desire for a smoother reward function, since, unlike joining a farming pool, joining a mining pool does not increase one's total rewards.

The chief problem with this model is that it is not decentralized. Although the actual consensus hardware is highly distributed, compared to existing PoW mining pools, the operators still present a point of centralization, more akin to validators in delegated or nominated PoS protocols. However, PoS systems at least provide strong penalties for misbehavior, which have worked in practice so far. As farmers in the pooled model are at best acting as light clients, the scope of action for malicious or colluding operators is much higher than in a typical blockchain. The honest majority farmer assumption becomes an honest majority operator assumption. If that assumption does not hold, farmers, and most users, will be unable to distinguish between valid and fraudulent transactions which appear in the longest chain, allowing operators to create coins out of thin air or spend farmer and user funds at will.

PoC blockchain design appears to stuck on the horns of a dilemma. On the one hand, we may abandon the goal of having farmers retain the history, while doing everything possible to minimize the burden of maintaining the state such that that the opportunity costs of running a full node remain negligible, giving farmers little incentive to pool. This leads to a much more limited construction, ruling out stateful smart contracts and even modest base layer scalability. On the other hand, we can abandon Nakamoto's vision and accept pooled consensus as a necessary evil, as has largely been done within the PoW and PoS communities, while at least rejoicing in the fact that participation is now fair and sustainable.

In this work we present a third option, which circumvents the farmer's dilemma without sacrificing the security or decentralization of the network, organized as follows:

1) To prevent farmers from discarding the history, we construct a novel PoC consensus protocol based on proofs-of-storage of the history of the blockchain itself, in which each farmer stores as many provably-unique replicas of the chain history as their disk space allows.
2) To ensure the history remains available, farmers form a decentralized storage network, which allows the history to remain fully-recoverable, load-balanced, and efficiently-retrievable.
3) To relieve farmers of the burden of maintaining the state and preforming redundant computation, we apply the classic technique in distributed systems of decoupling consensus and computation. Farmers are then solely responsible for the ordering of transactions, while a separate class of executor nodes maintain the state and compute the transitions for each new block.
4) To ensure executors remain accountable for their actions, we employ a system of staked deposits, verifiable computation, and non-interactive fraud proofs.

[2]Trusted pools are altogether different from the pooling approach suggested by Chia, which only serves to smooth out rewards over time, and does not resolve the farmer's dilemma. (www.chia.net/2020/11/10/pools-in-chia.html)

For concreteness, we present this approach within the Ethereum model of a fully-programmable, account-based blockchain, which periodically commits to the state of all accounts within the block header, though we believe many of the proposed techniques could be applied more generally for any Nakamoto-style blockchain.

III. Consensus

Requirements. We wish to create a longest-chain PoC consensus mechanism which incentivizes farmers to retain the blockchain history, in order to (partially) circumvent the farmer's dilemma. Holding to Nakamoto's vision, this mechanism must be permissionless, while ensuring that the chain remains secure, with respect to safety and liveness, as long as honest farmers collectively dedicate more storage than any cooperating group of attacker nodes. To ensure consensus retains the fairness of *one-disk-one-vote* we must discourage farmers from attempting to augment or replace storage with computation, by making this behavior economically irrational.

Proofs-of-Archival-Storage. To accomplish these goals, we base consensus on a *useful* Proof-of-Storage (PoStorage) [7] of the history of the blockchain itself. Farmers first create and store provably unique replicas of the chain history, before responding to random and publicly verifiable storage audits, which allow them to forge new blocks. This stands in contrast to a *useless* Proof-of-Space (PoSpace), as implemented (insecurely) in Burst[3] and proposed by Spacemint [8], Chia [9], and SpaceMesh [10], in which a node stores some randomly generated data. It can instead be understood as a simpler construction of Permacoin [11], paired with a more constrained application of a Filecoin Proof-of-Replication (PoR) [12], in which the scope of the data inputs are limited to the state updates to the blockchain, i.e. the block headers and transaction data. This idea is inspired by the notion of Proof-of-Unique-Blockchain-Storage [5], as originally proposed by Sergio Lerner, but utilized directly for consensus.

Hourglass Schemes. Our solution begins with an hourglass scheme [13] whereby, a prover applies a slow encoding to a file using a public key. Later, a random segment of the encoded file is audited by a verifier. To pass the audit, the prover must demonstrate possession of the encoded segment within a specified timeout. The timeout is tuned such that it is computationally infeasible to encode the file in response to a challenge and still pass the audit, demonstrating that the prover has retained the entire encoded file with high probability. Following Lerner, we may adapt this scheme for the blockchain setting.

Plotting History. To do this, we treat the confirmed history of the ledger as a single large file, divided into an ever-growing set of constant-sized, content-addressed pieces. For each piece, each farmer applies a Pseudo-random Permutation (PRP), using a unique public identifier (ID) as the encoding key, such as the hash of their public key. The resulting encoding is stored to disk, while a commitment, or *tag*, for each encoding is stored within a sorted binary tree. Each farmer now stores a provably unique replica of the ledger, and we define the *replication factor* as the number of unique ledger replicas stored across the network.

[3]https://www.burst-coin.org/

Choice of Permutation. While any cryptographically secure PRP will suffice for the codec, an ideal candidate would be both ASIC resistant and time-asymmetric, without imposing any new security assumptions. It turns out that we may construct such a permutation from the difficulty of computing modular square roots, using the permutation underlying SLOTH (slow-time hash function) as a guide [14]. This has the advantage of a near-optimal encoding time on x86-64 architecture, when using a 64-bit prime, while reducing the decoding time by at least one order of magnitude, significantly lowering the aggregate verification work done across the network for each new block.

Storage Audits. To generate a new block, the network issues an audit, seeking a valid PoR in response to a random challenge derived from the last block. Any tag which lies within a self-adjusting network solution range of the challenge may be used as the basis for a valid PoR. Since farmers store tags within a sorted binary tree, they may efficiently query their entire plot for the nearest tag to each challenge. Note that the quality of the PoR scales logarithmically with respect to the number of encodings under audit, and may be used to deduce the total space pledged to the network by all farmers. Any farmer who seeks to mine encodings on-demand with the same success probability as an honest farmer would then have to create the same number of encodings as were written to disk, within the span of a single block.

Re-introducing Time. To retain a notion of time in the absence of a mining delay, so that we may enforce the timeout of the hourglass scheme, we adopt the standard approach of dividing consensus into discrete time slots and epochs. To achieve clock synchronization without relying on a trusted service, such as Network Time Protocol (NTP), we follow Polkadot by deriving a shared relative clock from the blockchain itself [15], which remains secure as long as clock drift, measured over an epoch, is but a small fraction of the network delay.

Deterring Compression. Note that farmers may simply discard their encodings, while only retaining the much smaller sorted trees and a single copy of the unencoded history, from which they can reproduce any valid encoding on-demand. If a farmer does this many times for different IDs using additional up-front computation, they may magnify their apparent storage, effectively compressing their plot and breaking the fairness of the protocol. To deter this behavior we require that each commitment be based on a salt, which must be updated at a regular interval, determined by the total space pledged to the network and a security parameter L. While a modest interval will deter farmers who seek to break fairness, by making the cost of continuously regenerating encodings larger than simply buying more disks, a shorter interval may be needed initially, to prevent 51% attacks from dedicated CPU mining pools.

Preventing Grinding. To prevent the winning farmer from malleating on the contents of a block, in an attempt to influence subsequent audits in their favor, we follow Spacemint, and make this impossible by separating the canonical PoR and the malleable transaction data into two distinct sub-chains, while basing audit challenges solely on the content of the proof chain.

Constraining Simulation. In the event of an honest fork, a farmer could easily solve on both branches, effectively doubling their apparent space. If all farmers do this publicly, it will take longer to reach consensus on the longest chain, and the network becomes susceptible to a disastrous balancing attack, possible with only a small fraction of storage resources [16]. If farmers instead do this privately, they may magnify their apparent storage resource up to a factor e (2.718), allowing them to more easily engage in selfish farming and double-spend with only 27% of the total space [9]. Fortunately, we can make the advantage of these *simulation* or *nothing-at-stake* attacks negligible, by again following Spacemint, and recycling the same challenge over several consecutive audits [8], [16], [17], while recognizing that the interval need not be that large [18].

Handling Equivocation. A farmer still has the ability to extend both branches of a fork, using the same proof, in attempt to "hedge their bets". While this behavior reflects the most rational choice for the farmer, if left unchecked, it would result in unbounded perpetual forks and prevent the network from ever achieving consensus on a single longest chain. To remedy this, we again follow Spacemint, noting that equivocation is both detectable and punishable. If a farmer does sign two blocks using the same proof at the same height with different parents, they will forfeit their rewards and see their ID blacklisted from participation in consensus, effectively burning their plot.

Detecting Long-Range Attacks. Observe that in any PoC blockchain, an attacker may generate a *heavier* chain using only the average amount of space pledged over the chain's lifetime, by rewriting history from some fork deep in the chain's past [8], [9]. This *history-rewriting* attack may then be used to trick new farmers, or those who have been offline since before the fork, as well as light clients, into believing that the rewrite is in-fact the longest chain. Note that under proof-of-space consensus, any revised chain is indistinguishable from the honest chain, as the content of plots, and the resulting proofs, are random and independent of any particular chain history. However, in a proof-of-archival-storage chain, such as Subspace, these chains are distinguishable, unless the long-range attacker continuously revises their plot to reflect the new history they are creating. An indistinguishable history rewriting attack would therefore require both significant storage and (only partially parallelizable) computation resources, greatly reducing, if not ruling out, the feasibility of such an attack. In order to detect the weaker attack, a node need only sample the average height from which a valid solution was derived, while simply favoring the chain with the highest average height.

Security. By modeling the evaluation of the plot as a random process, similar in principle to the evaluation of a Verifiable Random Function (VRF) in PoS, we may extend Nakamoto's analysis, defined formally in the backbone protocol [4], and later adapted to the PoS setting [17], to show that we have the same security guarantees. Specifically, that no single party with less than one-half of available storage resources may generate a longer chain, resulting in the formal properties of safety and liveness. Following Spacemesh, our protocol only retains these qualities under the assumption that farmers are economically rational, such that they will always choose farming over mining, given that farming is the *cheaper* option [10]. This condition may be easily maintained, since the cost of mining is configurable within the protocol.

IV. Storage

Given that some farmers may only be able to store a partial replica of the history, while others may be able to store it many times over, we shall require a scheme which ensures the consistency of storage over time. It must be uniform, such that on average each piece is stored the same number of times across the network. It must be durable, such that with high probability. no single piece may be forgotten, whether accidentally or through malicious intent. It must be retrievable, both in full and for any single piece, in a manner which balances requests evenly across all farmers, allowing the overhead of serving history to remain negligible. It must also be stored in a manner which is efficiently verifiable, as farmers must not be expected to either synchronize or retain the full history. Finally, this must all work in the permissionless setting, without any central coordination, while accounting for the dynamic availability of farmers and the uneven growth of the history over time.

A. Load Balancing

To achieve a uniform distribution of pieces across the network, we employ a technique inspired by consistent hashing [19]. Recall that each farmer has a self-assigned ID, as the hash of their public key, which we may map to a ring over the domain of the hash function. If the farmer is unable to store a full replica within its plot, it favors pieces whose ID are closest to its own ID on the ring, up to the sector size afforded by its storage. Otherwise, it simply stores as many full replicas, and one partial, as its storage allows. This scheme has the benefit of *scattering* the history, such that a large fraction of the network would have to be taken offline in order to erase any single piece, while further ensuring that as more farmers join the network the average replication factor for any piece will approach the replication factor of the ledger.

As the history grows, farmers scan new pieces, retaining only those which fall within their (shrinking) self-assigned sector, while evicting those on the boundaries. This means that each farmer will have re-computed one-half of its plot every time the history doubles in size. However, unlike initial plotting, this computation is amortized over a much longer time period, such that it imposes negligible additional overhead. To minimize re-plotting for early farmers, the network will be seeded with several terabytes of data at genesis, comprising the history of a few leading blockchains.

While a *lazy* farmer could choose to deviate from the above strategy, we note that there is little incentive for doing so, as it will have no impact on their reward function. At best, it will save them a negligible amount of computational overhead. A better strategy would be to instead increase their storage capacity in step with the history growth, such that their sector size remains constant, allowing them to retain their otherwise valid encodings along the boundaries.

A farmer could also employ a sybil strategy, in which they generate many IDs, allowing them to create many small partial replicas from the same small slice of the history. While this still has no impact on their reward function, it could significantly reduce the bandwidth costs of the initial plotting phase. To deter this behavior, and instead encourage farmers to store as many full replicas as possible, we require that each farmer salt the challenge with their ID. This simple modification forces a farmer to maintain a unique search tree and process a separate audit for each ID, increasing their computational overhead linearly in the number of IDs. We therefore expect rational farmers to avoid sybil farming, as the one-time bandwidth savings will quickly be overshadowed by the higher constant cost of participation in consensus.

B. *Durability*

To obtain a level of durability many orders of magnitude greater than simple replication, we apply the well-known technique of erasure coding [20]. Concretely, at the beginning of each new epoch, the history for some past epoch may now be used as the basis for a valid solution. We start by chunking each block from the newly confirmed epoch into a constant-sized batch of k source pieces. We then apply an erasure code with rate one-half, to obtain an additional k parity pieces. Note that farmers only retain and encode source or parity pieces which falls within their sector of the ring. Finally, we construct a Merkle tree over all n pieces in the batch. Since batches will rarely align neatly with individual blocks, we aggregate pieces over many blocks until the desired batch size is obtained. We then construct a much smaller *state header chain* which simply commits to the Merkle root of each batch. This allows a winning farmer to prove that some encoding, which satisfies the audit, is actually derived from the confirmed history, without requiring farmers or verifiers to maintain the history themselves. Instead they are only required to maintain the much smaller state header chain, which may be further compressed to just the Merkle roots after initial synchronization.

C. *Retrievability*

To ensure the history remains fully retrievable, and therefore available, we employ a simplified version of a Distributed Hash Table (DHT). Observe that the notion of distance between piece and farmer IDs on the ring is equivalent to the XOR metric, a key component of a Kademlia DHT [21]. If we simplify a K-DHT, such that each farmer only announces its ID and contact information, any node may then retrieve a piece by simply locating a farmer whose ID is close enough to the piece by XOR distance, before requesting the piece from that farmer. This reduces the K-DHT from a generic key-value store into a kind of decentralized domain name service. The storage overhead for the DHT on each node is tiny, even for a large number of farmers, and would require only a few hops at most to locate any particular piece.

To allow for bandwidth efficient and load balanced fetching of a farmer's sector during the initial plotting phase, we may adapt the clever approach pioneered by the BitTorrent network, by partitioning, multiplexing, and streaming requests across many *nearby* peers. Based on the success of BitTorrent, we expect that an optimistic *tit-for-tat* piece sharing policy, combined with the significantly reduced overhead needed to operate a DHT node and the expected load-balancing of requests across all nodes, will lead a sufficiently large fraction of farmers to share their pieces without any direct financial incentives [22]. A similar approach employed by Arweave, a blockchain-based permanent storage network, has worked in practice for several years [23].

Note that this scheme only allows for retrieval of individual pieces if their ID is previously known, as they are content-addressed. We may sidestep this issue, if we instead address pieces by the hash of their index, as they are created, allowing nodes to more easily fetch specific periods of the history. We may also extend this schema with a secondary layer of resolvers, in order to allow for the efficient retrieval of particular blocks, transactions, or any arbitrary object by ID, as is often by required by light clients. To accomplish this, each farmer allocates a small, configurable amount of memory, in which they retain a mapping between those resolvers closest to their ID and the index of the piece where the underlying data is stored. A light client can then query some object by ID, obtaining its storage index, before fetching the full piece and extracting out the required data. Farmers may optionally maintain an ephemeral cache of those items that are most popular within the DHT itself.

D. *Cost of Storage*

To ensure the history does not grow beyond total network storage capacity, we modify the transaction fee mechanism such that it dynamically adjusts in response to the replication factor. Recall that in Bitcoin, the base fee rate is a function of the size of the transaction in bytes, not the amount of BTC being transferred. We extend this equation by including a multiplier, derived from the replication factor. This establishes a mandatory minimum fee for each transaction, which reflects its perpetual storage cost. The multiplier is recalculated each epoch, from the estimated network storage and the current size of the history. The higher the replication factor, the cheaper the cost of storage per byte. As the replication factor approaches one, the cost of storage asymptotically approaches infinity. As the replication factor decreases, transaction fees will rise, making farming more profitable, and in-turn attracting more capacity to the network. This allows the cost of storage to reach an equilibrium price as a function of the supply of, and demand for, space.

V. Computation

Farmers seek to dedicate all available disk space to consensus while preforming as little redundant computation as possible. However, to be assured that they remain on the longest *valid* chain they must compute all intermediate state transitions, which implies they also maintain the state. As the burden of maintaining the state and computing transitions grows larger, both the farmer's and verifier's dilemmas will present themselves, leading farmers to sacrifice security for higher rewards at a lower cost, by either becoming light clients or joining a trusted farming pool. In order to resolve both dilemmas, we require a method which relieves farmers of this burden, while still allowing them to be certain that they are extending the longest valid chain. Critically, this method must not degrade the liveness, fairness, or safety of block production.

The solution we propose follows the classic technique in distributed systems of decoupling consensus and computation. In this system, farmers are solely responsible for providing *subjective* and *probabilistic* consensus over the ordering of transactions, while a separate class of executor nodes compute the *objective* and *deterministic* result of that ordering. Executors are selected through a stake-based election, separate from block production, and analogous to the block finalization technique proposed by Casper FFG [24]. Executors are incentivized by sharing transactions fees with farmers, while being held accountable through a system of non-interactive fraud proofs [25] and slashing [26].

While ostensibly similar to, and certainly influenced by, *Flow* [27]–[29], our approach is far simpler (two, not four classes of nodes), retains compatibility with Nakamoto consensus (as opposed to BFT style consensus), and maintains the honest majority security assumption. We also take inspiration from the authors of Truebit [30], who first recognized that optimistic off-chain computation, with fallbacks to on-chain verification, could be used to realize a trustless decentralized mining pool, which is largely what we accomplish here. However, unlike Truebit, we employ a system of *non-interactive* fraud proofs, while resolving the verifier's dilemma in an altogether different manner. Unlike protocols such as ChainSpace [31] and LazyLedger [32], which achieve decoupling by delegating computation to clients, our system retains global state, allowing for cross-contract calls and composability of applications.

Note that this approach is entirely different from hybrid PoC / PoS consensus mechanisms employed by other storage-based blockchains. Filecoin [33] requires staking proportionate to one's storage as a *pre-condition* for farming, sacrificing the permissionless nature and dynamic availability of Nakamoto consensus. Mass [4] provides a dual incentive structure whereby anyone may produce blocks and earn rewards by either staking or farming. We instead clearly distinguish between a permissionless farming mechanism for block production and permissioned staking mechanism for block finalization.

A. Overview

We begin by restricting the role of farmers to simply providing a subjective ordering over *potentially valid* transactions. To do this, a farmer collects transactions as normal, while only verifying that the sender has provided a valid signature and is able to cover the fee. When a farmer finds a PoR which satisfies the storage audit, they will bundle all valid transactions into a new block, while committing to the last valid state root proposal they observe. Since farmers no longer compute the new state root themselves, they do not need to maintain the code, state, or account balances for any contracts. Instead they must only maintain the much smaller set of balances (and nonces) for all Externally Owned Accounts (EOAs). This has the added benefit of allowing blocks to propagate faster (and reducing the probability of forks) as farmers need only verify the PoR and whether or not the block contains potentially valid transactions, before relaying the block to their peers.

A separate class of executor nodes will then maintain the full state and apply transactions, returning the new *proposed* state root. For each new block, a small constant number of executors are chosen through a stake-weighted election. Anyone may participate in execution by syncing the state and placing a small deposit. To determine if they are elected, each executor will use the PoR hash as the input to a VRF linked to their deposit and weighted by their proportion of the total stake. Regardless of whether or not they are elected, each executor will then apply each transaction to the state, with all appropriate state transition functions, in the order specified by the block. As each transaction is executed, they will incrementally commit to an intermediate state root, forming an execution trace, while also compiling an aggregate diff of all EOA balances.

Note that any non-elected executor who skips the state update procedure will be unable to propose a new valid state root, in the event they are elected for some future block. All elected executors will then broadcast an execution receipt (ER), consisting of the VRF proof, the final state root, a Merkle root of all intermediate state roots, an execution trace, and the EOA balances diff. Upon arrival and validation of the ER, each farmer will optimistically apply the balances diff, while noting the new proposed state root to expect (or embed) in the next block.

B. Retaining Liveness

Considering the asynchronous nature of the network, the presence of byzantine actors, and the probabilistic nature of both block production and execution, we cannot expect that

[4]https://massnet.org

these two processes will always alternate smoothly. In the event that a farmer is chosen to produce a new block before they receive a valid ER for the last block, we still allow them to publish the block without an ER.[5] This disruption could occur for several possible reasons: the new block was created soon after its parent, either before execution could be completed, or the ER could fully propagate across the network; no executor was actually elected to finalize the block; or the elected executor/s chose to withhold the ER, in an effort to attack the liveness of execution.

To recover from this event, we simply re-run the executor election for each new block, while allowing any newly elected executor to include all past ERs required to catch up, the new ER for the latest block, and a cumulative EOA balances diff. In the meantime, farmers may continue to verify new transactions pessimistically, i.e., based on the lowest possible EOA balance for the pending transactions set. While this may result in some transactions being falsely flagged as illegal [6] it will not allow an illegal transaction to make its way into a new block.

It is important to note that, as each executor election is independent and random, the likelihood of this event recurring decays exponentially for each new block. We also set the election threshold such that on average a small constant number of executors are elected each round, in order to significantly decrease the probability that this event even occurs at all. To handle the case where an adversary with deep pockets consumes a super-majority of the stake without actually running an executor, in order to attack the liveness of execution, we allow the election threshold to dynamically adjust itself in accordance with the observed availability of executors.

C. Preserving Fairness

Farmers split transaction fee rewards evenly with all executors, based on the expected number of ERs for each block.[7] For example, if 32 executors are elected, the farmer will take half of the all transaction fees, while each executor will take 1/64. A farmer is incentivized to include all ERs which finalize execution for its parent block because doing so will allow it to claim more of its share of the rewards for its own block. For example, if the farmer only includes 16 out of 32 expected ERs, it will instead receive 1/4 (not 1/2) of total rewards, while each of the 16 executors will still receive 1/64. Any remaining shares will then be escrowed within a treasury account under the control of the community of token holders, with the aim of incentivizing continued protocol development. Overall, unclaimed funds should remain negligible.

In the event more than 32 ERs are included, the farmer's share will remain the same, while executors will be diluted accordingly, in order to ensure that the dynamic election threshold maintains the correct balance. Critically, this reward sharing may not be retroactive, else it would incentivize selfish executors to withhold ERs for certain farmers who produce subsequent blocks, with whom they may collude. We note that this scheme is also compatible with a delegated or nominated staking mechanism, but unlike many proof-of-stake consensus protocols, has no strict upper bound on the number of delegates or nominees.

D. Ensuring Validity

In the event that any elected executor proposes an invalid state transition, all honest executors and full validating nodes will immediately recognize this and react accordingly. Recall that unlike farming, execution is entirely objective and deterministic. For any block and initial state, there exists one, and only one, valid final state and execution receipt, upon which all honest executors will arrive.

Any honest executor may then easily inspect the execution trace for the point of divergence, and compile a fraud proof consisting of:

1) The initial state for all accounts touched by the transaction.
2) All state transition functions (i.e., the contract code).
3) A commitment to the valid output state.
4) Merkle inclusion proofs for each above.

Any node, given the execution trace and fraud proof can determine the validity of the ER by:

1) Verifying the Merkle inclusion proofs for the input state and code against the last valid intermediate state root.
2) Applying all state transition functions to the input state.
3) Verifying the output state matches the commitment, and that the Merkle inclusion proof is valid for a *different* intermediate state than was witnessed by the ER. .

If the fraud proof is valid, the executor who proposed the invalid ER will see their entire deposit confiscated and placed within the protocol treasury. For this to occur, both the invalid ER and the fraud proof must be included in the chain. This means that farmers must reference all proposed ERs they observe, when authoring a new block, while knowing that all invalid ERs will quickly be sanitized out of the ledger. This will result in a (retained) re-org of the chain of execution receipts, without impacting the ordering of transactions. Instead of including the entire ER, farmers need only record the much smaller header, consisting of the VRF proof and a Merkle root over all intermediate state transitions.

In the expected case, where executors are economically rational, all ERs will be valid, and fraud proofs will not be needed. If executors are dishonest, a separate fraud proof would need to be generated for each invalid ER. Any malicious executor who controlled a majority of stake could then impose a limited denial of service (DoS) attack on the chain by creating many different, invalid ERs, forcing honest nodes to generate the corresponding fraud proofs and occupying precious block space. If we make the minimum executor stake deposit larger than the expected cost of simply occupying the

[5]While the farmer may instead choose to wait for a valid ER (in the case of a network delay), they do so at the risk of generating a fork and forfeiting their block reward.

[6]With respect to the inability for the sender to cover the transaction fee

[7]We use this rate for explanatory purposes, while noting that in order to minimize the plutocratic nature of PoS, executor shares should be smaller in practice.

block space with a transaction (i.e., the equivalent gas cost) then it would always be more rational to DoS the network by simply creating large transactions with sufficient fees to store some arbitrary data or perform some arbitrary computation, as can already be done today on Ethereum.

E. Addressing the Verifier's Dilemma

Most schemes which rely on fraud proofs employ bounties, as an incentive to circumvent the tricky verifier's dilemma. We note that since all fraud proofs are canonical for the same invalid ER, any reward mechanism could easily be front-run. Instead of forwarding a fraud proof as is, any executor could just as easily claim it as their own. Ultimately, this would create an incentive for farmers and executors to collude, such that the farmer who creates the block would claim the fraud proof as their own. While this could be constrained by requiring a deposit to participate as a verifier, it would not solve the problem. We might also try to elect one or more verifiers under a similar scheme, but this would significantly degrade the liveness of fraud proof generation.

Instead, we rely on the fact that all executors may act as verifiers at negligible additional cost, as they are already required to maintain the valid state transitions in order to propose new ERs. If we further require them to reveal fraud in order to protect their own stake and claim their share of the rewards, in the event that they themselves are elected, then we can provide a more natural solution to the verifier's dilemma. We achieve this by similarly punishing any executor who extends an invalid ER without first demonstrating fraud. It is worth noting that farmers also have an incentive to include both invalid ERs and their corresponding fraud proofs in a new block, as without them, any new valid ERs for the parent block will not be accepted, reducing the farmers share of its available rewards.

Since multiple executors will be elected each round, lazy executors could still perhaps wait for some other more eager executors to generate the fraud proof first. If we also require that each ER include a commitment to each fraud proof, for each invalid ER witnessed by the previous block, then we can ensure that executors may not release the ER until they have first created (or seen) all required fraud proofs. Given the network delay and a stochastic block production process, any elected executor who waits for all valid fraud proofs before releasing their own ER risks being left out of the next block and missing out on their share of the rewards. It is therefore rational for an executor to do the negligible and rare extra work required to generate the fraud proof locally in order to release their own ER as soon as possible.

In addition to executors, any full node may also monitor the network and generate fraud proofs, by virtue of the fact that no deposit is required to act as verifier. Since fraud proofs are canonical, any node may generate and propagate them locally, without consuming any more bandwidth than if they had simply received them and passed them on. This has the benefit of speeding up fraud proof propagation, while further strengthening the security guarantees. By allowing anyone to propose a fraud proof we are able to ensure the validity of the chain in the face of a dishonest majority of executors, allowing the system remain secure as long as a single honest executor remains connected to at least one honest farmer.

F. Maintaining Safety

We define safety as the property that a transaction, once included in the ledger, may not subsequently be reverted. Nakamoto consensus only provides probabilistic safety, through the k-deep confirmation rule, which states that for a given adversarial fraction of consensus resources, the probability that a transaction may be reverted decreases exponentially with each new block. Our goal is to simply retain this property.

To do this, we must first distinguish between *illegal* and *invalid* transactions. Strictly speaking, any transaction which circumvents the DoS resistance mechanism of the transaction fee is illegal, and should never be included in the ledger. Farmers enforce legality by ensuring that a transaction has a valid signature and can cover the specified fee. We then define a valid transaction as one which results in a valid state transition. Executors enforce the validity of transactions by applying them deterministically in the order specified by farmers. It is important to recognize that invalid transactions may certainly be legal, and by no means imply malicious intent, as their execution is not guaranteed to reflect the same global state witnessed at their creation. Accordingly, invalid transactions are simply ignored by all honest executors.

It then follows that dishonest executors may only *temporarily* influence the *apparent* safety of the chain by making an invalid transaction appear valid, but they may not revert a transaction, as they have no way to exclude it from the pre-determined set of potentially valid transactions. Farmers alone have the ability to do this, through an intentional fork, which, following Nakamoto's analysis, may only be accomplished with negligible probability for sufficient k [1]. Moreover, even if many invalid ERs are included in the chain, the time it takes to recognize them as such will always be less than k. Therefore, our safety guarantees remain the same, even in the presence of a dishonest majority of executors, given that we maintain an honest majority of farmers.

Recall that when evaluating an ER, farmers only check to ensure the VRF proof is valid, as they have no way of knowing if it does indeed reflect a valid state transition. Even if they knew it did not, they would still need to include the invalid ER, so that it could be referenced by its corresponding fraud proof. Since we expect a small constant number of ERs for each block, and presuppose economically motivated executors, we expect that in the average case all ERs will agree, allowing farmers to optimistically apply the EOA balances diff and continue collecting new transactions. In the event that one or more ERs disagree, a farmer will instead screen new transactions pessimistically and wait until enough fraud proofs have arrived to rule out all but one remaining valid ER.

In the unlikely event that all executors agree on the same invalid ER, these dishonest executors may then only convince a farmer that a proposed ER is valid for as long as it takes for the farmer to see a valid fraud proof, which is largely a function of the network propagation delay. For a given delay *D*, the maximum amount of time for a farmer to see the fraud proof is *2D* plus the proof generation time. As *D* must already be a small fraction of the expected farming rate, for any secure implementation of Nakamoto consensus, we can expect that with high probability all fraud proofs will be seen within two blocks.

Perhaps the worst act a dishonest executor can achieve is to confuse farmers as to the legality of transactions by presenting a fraudulent EOA balances diff as part of an invalid ER. This would allow them to insert illegal transactions into the next block while preventing many legal ones from being selected. As previously mentioned, farmers only apply the diff optimistically if there are no conflicting ERs for the last block. This means that a dishonest party would first need to control a super-majority of stake and then hope that a single honest executor was not still elected. Even then, any illegal transactions will still be recognized as such by honest executors and will sanitized from the ledger, while costing the attacker dearly in forfeited stake.

G. Sub-Linear Synchronization

As new farmers join the network, they must sync from genesis, to ensure they work to extend the longest valid chain. Traditionally, this would be done by verifying the PoR and state transitions for each block. Since we wish to prevent farmer's from having to maintain the state, even during initial sync, we require an alternative which still allows them to be assured the chain is valid. If they instead only sync the block headers, they would still need to expend computation and bandwidth linear in the height of the chain. This is made worse by the fact that we do not embed encodings or Merkle proofs within blocks, which means they would have to manually verify each PoR, by retrieving the piece and its Merkle proof from the DHT, and then re-derive each encoding from its inputs.

To allow farmers to be assured that they are on the longest chain, while only having to download and manually verify blocks logarithmic in the chain's height, we implement a super-light client, inspired by Flyclient [34]. To allow this, we must first extend the protocol such that we incrementally commit to each new block using a Merkle Mountain Range. When a new farmer joins the network, they query peers for the most recent block, taking that as the supposed height of the chain. They then iteratively divide the chain in half (by index), and randomly select a block in the left half, before fetching the block from the DHT, reconstructing it in full and manually verifying the PoR. If everything checks out, the farmer continues, splitting the right half of the chain once more. If they reach the head of the chain, and all blocks are valid, then, with high probability they are on the longest chain.

For a farmer to be certain they are on the longest *valid* chain, in the presence of multiple candidate chains, they employ a simple process of elimination. To eliminate chains which are the product of a long-range attack, the farmer compares the average height of solutions for each chain to the expectation, favoring those which are closest. To further eliminate chains which include an invalid state transition, perpetuated by a dishonest majority, the farmer need only ask each proposing peer for the latest valid fraud proof it has observed. If at least one peer is honest, the farmer will be able to obtain fraud proofs for all invalid chains, leaving the farmer on the longest *valid* chain.

H. Optimizations

Following [25], we note that the (potentially large) execution trace within an ER is only needed if the new proposed state root is invalid, and then only by executors (or any full node), as they are the only ones who may generate the corresponding fraud proof. Farmers however only need to be assured that the execution trace is available, so that any single honest executor may recover the full trace and generate a fraud proof if it is indeed invalid. We can then require executors to erasure code the trace, such that any node can be assured that the coding was done correctly and that the full data is available, while only having to sample a small constant fraction of the data at random. We further note that in the expected case, only farmers would need to sample the data, as executors would be able to immediately tell if the ER is invalid from its header alone.

While all honest ERs for a given block will have the same body, recall that the size of the header grows linearly with the expected number of elected executors for each block, due to the additional proofs-of-election. We can reduce this to a constant size by replacing the VRF with a non-interactive deterministic signature scheme, such as BLS [35], which would allow us to compress all honest proofs-of-election into a single public key and a single signature.

We may also implement one of several stateless blockchain proposals [36]–[38], which would allow us to entirely relieve farmers, and perhaps executors, of the burden of maintaining the state, while also removing the EOA balances diff from the ER. At the present time, these schemes impose additional security assumptions and computational overhead for farmers, while increasing the complexity of the user experience. As these schemes become more developed, we expect they could be used to remove the need for farmers to maintain the EOA balances entirely, while perhaps fully resolving the farmer's dilemma.

With respect to fraud proofs, we note that although they are rarely expected, they may still be quite large, as their size is only bounded by the maximum size of the code and state for any given contract account, as a function of the total number of contract calls that may be included in a transaction, typically constrained by the block gas limit itself. To reduce the size of these inputs, we may increase the number of intermediate state commitments required for the execution trace, noting the larger ER size will be mitigated with the erasure coding scheme presented earlier. We may further reduce the size of the contract code required for the proof by storing it within an efficient accumulator, such as a Merkle tree, in a manner similar to the MAST proposal [39].

VI. Conclusion

Consensus based on proofs-of-archival-storage yields an incentive-compatible PoC protocol, without sacrificing the security guarantees or dynamic availability of Nakamoto consensus. These properties are maintained, despite our decoupling of consensus and computation, as long as at least one honest executor (or full node) remains connected to the farmer network. In other words, the protocol remains secure even if a majority of executors are dishonest, given that a majority of farmers remain honest. By removing the requirement that farmers maintain the history, compute every state transition and, by implication, maintain the state, the farmer's dilemma is largely resolved. This has the added benefit of lowering the barriers to entry for farming, such that it may be more easily done with commodity hardware, while allowing for scaling of execution up to the limits of the hardware employed by executors. In many ways, this achievement may be likened to a decentralized farming pool, in which trusted pool operators are replaced with verifiable and accountable executors.

We are still left wondering to what extent execution may be scaled, without sacrificing some meaningful level of decentralization. Luckily, many recent works have demonstrated that it is indeed possible to scale Nakamoto consensus, at least without sacrificing security or dynamic availability, in order to achieve optimal throughput [40], fast finality [41], and horizontal scaling [42][8]. If we can also achieve a fully stateless version of farming, then the *blockchain bloat* incurred by scalability will no longer be a concern, at least as it relates to decentralization. These questions shall be fully analyzed in a forthcoming work.

We close by noting that although Subspace, like Ethereum, is designed to support any arbitrary computation, we believe that is especially well-suited to decentralized applications which either produce or require large quantities of data. In contrast to mutable and ephemeral storage services such as Sia [43], Storj [44], or Filecoin [33], Subspace provides a permanent storage layer, better suited for blockchain-based computation. Unlike other permanent storage networks, such as Arweave [23], we are able to efficiently price storage based on network capacity, while making that storage available to a global execution layer. Ultimately we believe Subspace reflects a model for a sustainable, scalable and incentive-compatible *data availability layer*, which holds true to Nakamoto's vision for a more decentralized and democratic future.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[2] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[3] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[4] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*, pp. 281–310, Springer, 2015.

[5] S. D. Lerner, "Proof of unique blockchain storage," Sep 2015.

[6] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 706–719, 2015.

[7] B. Fisch, "Poreps: Proofs of space on useful data.," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 678, 2018.

[8] S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak, "Spacemint: A cryptocurrency based on proofs of space," in *International Conference on Financial Cryptography and Data Security*, pp. 480–499, Springer, 2018.

[9] B. Cohen and K. Pietrzak, "The chia network blockchain," 2019.

[10] T. Moran and I. Orlov, "Simple proofs of space-time and rational proofs of storage," in *Annual International Cryptology Conference*, pp. 381–409, Springer, 2019.

[11] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," in *2014 IEEE Symposium on Security and Privacy*, pp. 475–490, IEEE, 2014.

[12] J. Benet, D. Dalrymple, and N. Greco, "Proof of replication," *Protocol Labs, July*, vol. 27, p. 20, 2017.

[13] M. Van Dijk, A. Juels, A. Oprea, R. L. Rivest, E. Stefanov, and N. Triandopoulos, "Hourglass schemes: how to prove that cloud files are encrypted," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 265–280, 2012.

[14] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx.," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 366, 2015.

[15] H. K. Alper, "Network time with a consensus on clock," *Cryptology ePrint Archive, Report 2019/1348*, 2019.

[16] X. Wang, G. Kamath, V. Bagaria, S. Kannan, S. Oh, D. Tse, and P. Viswanath, "Proof-of-stake longest chain protocols revisited," *arXiv preprint arXiv:1910.02218*, 2019.

[17] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*, pp. 357–388, Springer, 2017.

[18] V. Bagaria, A. Dembo, S. Kannan, S. Oh, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Proof-of-stake longest chain protocols: Security vs predictability," *arXiv preprint arXiv:1910.02218*, 2019.

[19] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 654–663, 1997.

[20] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[21] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*, pp. 53–65, Springer, 2002.

[22] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, pp. 68–72, Berkeley, CA, USA, 2003.

[23] S. Williams, V. Diordiiev, L. Berman, and I. Uemlianin, "Arweave: A protocol for economically sustainable information permanence," *arweave. org, Tech. Rep*, 2019.

[24] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[25] M. Al-Bassam, A. Sonnino, and V. Buterin, "Fraud proofs: Maximising light client security and scaling blockchains with dishonest majorities," *arXiv preprint arXiv:1809.09044*, vol. 160, 2018.

[26] V. Buterin, "Slasher: A punitive proof-of-stake algorithm," Jan 2014.

[27] A. Hentschel, D. Shirley, and L. Lafrance, "Flow: Separating consensus and compute," *arXiv preprint arXiv:1909.05821v1*, 2019.

[28] A. Hentschel, Y. Hassanzadeh-Nazarabadi, R. Seraj, D. Shirley, and L. Lafrance, "Flow: Separating consensus and compute–block formation and execution," *arXiv preprint arXiv:2002.07403v1*, 2020.

[8] We note that these schemes also allow for more frequent reward distribution to farmers, removing the last possible benefit of pooled farming.

[29] A. Hentschel, D. Shirley, L. Lafrance, and M. Zamski, "Flow: Separating consensus and compute–execution verification," *arXiv preprint arXiv:1909.05832*, 2019.

[30] J. Teutsch and C. Reitwießner, "Truebit: a scalable verification solution for blockchains," *White Papers*, 2018.

[31] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[32] M. Al-Bassam, "Lazyledger: A distributed data availability ledger with client-side smart contracts," *arXiv preprint arXiv:1905.09274*, 2019.

[33] J. Benet and N. Greco, "Filecoin: A decentralized storage network," *Protoc. Labs*, pp. 1–36, 2018.

[34] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, "Flyclient: Super-light clients for cryptocurrencies," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 928–946, IEEE, 2020.

[35] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of cryptology*, vol. 17, no. 4, pp. 297–319, 2004.

[36] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," in *Annual International Cryptology Conference*, pp. 561–586, Springer, 2019.

[37] A. Tomescu, I. Abraham, V. Buterin, J. Drake, D. Feist, and D. Khovratovich, "Aggregatable subvector commitments for stateless cryptocurrencies," in *International Conference on Security and Cryptography for Networks*, pp. 45–64, Springer, 2020.

[38] S. Agrawal and S. Raghuraman, "Kvac: Key-value commitments for blockchains and beyond," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 839–869, Springer, 2020.

[39] J. Rubin, M. Naik, and N. Subramanian, "Merkelized abstract syntax trees," 2014.

[40] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 585–602, 2019.

[41] S. Li and D. Tse, "Taiji: Longest chain availability with bft fast confirmation," 2020.

[42] R. Rana, S. Kannan, D. Tse, and P. Viswanath, "Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation," *arXiv preprint arXiv:2005.09610*, 2020.

[43] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *Nebulous Inc*, 2014.

[44] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin, "Storj a peer-to-peer cloud storage network," 2014.