Audita:
A Blockchain-based Auditing Framework for Off-chain Storage

Danilo Francati*, Giuseppe Ateniese*, Abdoulaye Faye†, Andrea Maria Milazzo†,
Angelo Massimo Perillo*, Luca Schiatti† and Giuseppe Giordano†
* *Stevens Institute of Technology, USA*
{*dfrancat, gatenies, aperillo*}*@stevens.edu*
† *Accenture Labs, France*
{*abdoulaye.faye, andrea.maria.milazzo, luca.schiatti, giuseppe.giordano*}*@accenture.com*

***Abstract*—The cloud changed the way we manage and store data. Today, cloud storage services offer clients an infrastructure that allows them a convenient source to store, replicate, and secure data online. However, with these new capabilities also come limitations, such as lack of transparency, limited decentralization, and challenges with privacy and security. And, as the need for more agile, private and secure data solutions continues to grow exponentially, rethinking the current structure of cloud storage is mission-critical for enterprises.**

By leveraging and building upon blockchain's unique attributes, including immutability, security to the data element level, distributed (no single point of failure), we have developed a solution prototype that allows data to be reliably stored while simultaneously being secured, with tamper-evident auditability, via blockchain.

The result, Audita, is a flexible solution that assures data protection and solves for challenges such as scalability and privacy. Audita works via an augmented blockchain network of participants that include storage-nodes and block-creators. In addition, it provides an automatic and fair challenge system to assure that data is distributed and reliably and provably stored.

While the prototype is built on Quorum, the solution framework can be used with any blockchain platform. The benefit is a system that is built to grow along with the data needs of enterprises, while continuing to build the network via incentives and solving for issues such as auditing, outsourcing and malicious users.

***Index Terms*—Blockchain; Distributed systems; Proof of storage**

1. Introduction

A cloud storage service provides a decentralized infrastructure that allows users to store their data online. Users pay cloud providers (such as Amazon [2] or Google [15]) to access the service and receive benefits such as data replication, reliability, and security. Users' data is entirely administered by the cloud provider, which is entrusted with selecting reliable storage servers, maintaining data intact, and delivering it promptly when requested. Also, decentralization in cloud storage is often limited, which affects data replication and results in data loss in case of natural disasters or denial of service attacks. For example, WikiLeaks has published a highly confidential internal document [29], showing that Amazon cloud storage (S3) has a limited number of data centers across the world.

On the other hand, the *blockchain* is a technology that is fully transparent and distributed across the world. The blockchain is a sequence of public and immutable structured data (called blocks) administrated by a peer-to-peer network. It resembles the digital time-stamping system introduced by S. Haber and W. S. Stornetta [16], which was improved one year later by Bayer et al. in [8]. Today, blockchain is a building block for many technologies such as Bitcoin [22] and Ethereum [10].

Thanks to its properties, the blockchain is arguably an ideal infrastructure for the next generation of decentralized storage services. However, users' data cannot be stored into blocks because the blockchain is public and immutable, and it does not scale. Encrypting or hashing relevant information and storing the results on the blockchain is also nugatory. Encryption (even if information-theoretic secure) tends to "deteriorate" over time because keys get routinely exposed or misused. Hashing data does not guarantee actual data storage. Moreover, the resultant hash provides proof of existence, which may violate privacy policies.

In this paper, we provide: 1) A detailed overview of the state-of-the-art blockchain-based storage systems. 2) We identify the fundamental properties that such systems must satisfy, and we compare them based on these properties. 3) We propose Audita, the first blockchain-based decentralized storage system that satisfies all properties. It provides most of the benefits of storing data on the blockchain without actually doing it. The functionality we achieve is that *as long as the blockchain is growing and transactions are generated, users can be reasonably confident their data is intact even if stored off-chain*. Adding a block to the blockchain triggers an audit mechanism that implicitly verifies that a random portion of all files stored off-chain is intact. The audit is automatic and does not require file owners to be online or participate.

1.1. Our technique

We introduce a general technique to implement Audita on top of any blockchain framework. In our system, the blockchain is used for accountability, and data is stored

off-chain in a new category of peers, called *storage-nodes*, next to the standard blockchain's peers, referred to as *block-creators*. Storage-nodes provide the storage capability to save and maintain users' data, while block-creators run the protocol of the underlying blockchain.

In Audita, a user with a file F, sends a request to a server D, called *dealer*. The file distribution phase is performed by D that gives, to each storage-node, a random subset of file chunks. To ensure a storage-node maintains the file subset intact (and locally stored), we leverage the blockchain and a publicly verifiable provable data possession (PDP) scheme to implement a global interactive proof system (*i.e.*, prover-verifier paradigm). A block B, created by a block-creator, will be permanently added into the chain if and only if it is accompanied by a set of ℓ proofs of possession $\{\pi_i\}_{i\in[\ell]}$ (generated by ℓ distinct storage-nodes over a portion of stored data). We refer the reader to Figure 1 for a view of the protocol workflow. In more detail, the block-creator, behaving as the verifier, challenges the storage-nodes (the provers): It contacts a random subset of storage-nodes and asks for a proof of possession π_i (step (1)). The first ℓ received proofs $\{\pi_i\}_{i\in[\ell]}$ will be used as a "ticket", that will give to the block-creator, the possibility to propose the candidate block B to the network (step (2)). If both block B and proofs $\{\pi_i\}_{i\in[\ell]}$ are valid, then the blockchain is extended (step (3)).

Thanks to its modularity, our technique allows us to implement Audita on any reward-based blockchain system. Rewards are widely used in Bitcoin and Ethereum to incentivize peers to act honestly. Audita follows the same approach. Storage-nodes and block-creators that cooperate to extend the blockchain will receive a reward in exchange for their work. The type of reward in our system can be quite arbitrary and can range from cryptocurrency-based rewards in a public blockchain to contractual agreements in a permissioned blockchain.

1.2. Organization

The paper is organized as follows. Section 2 discusses the main blockchain-based decentralized storage systems present in the literature. In Section 3, we give a detailed overview of the main properties that blockchain-based decentralized storage systems must satisfy. Section 4 introduces the notation and the required building blocks that we use throughout this paper, and section 5 defines the security assumptions we make. In Section 6 we introduce Audita with a detailed discussion about its properties and requirements. Section 7 presents our implementation and the experimental results. Finally, Section 8 concludes the paper.

2. Related work

Some recent works have focused on how to store, in a decentralized way, a publicly known file (e.g., the digital content of a library) using the blockchain. Miller et al. [20] propose Permacoin that repurposes the computational power spent in computing the Proof of Work (PoW), currently in use by many blockchain systems such as Bitcoin. In Permacoin, the PoW involves two distinct elements to incentivize miners to store a file locally. First, the PoW is computed over the chunks of the file. This makes the computation an implicit form of *proof of possession*. However, this does not give any guarantee about the decentralization of the data: Miners could outsource both the file chunks and the PoW computation to a third party and jeopardize data decentralization. For this reason, Permacoin proposes an efficient multi-use hash-based signature scheme called floating preimage signature scheme and forces miners to use their secret key during the PoW. At a high level, miners are required to sign and hash the intermediate states of the PoW recursively. In this way, storage outsourcing results in either exposing the secret key or decreasing the probability of generating a new valid block. Since chunks are hashed during the computation of the PoW, Permacoin suffers from high bandwidth consumption. Indeed, the chunks must be sent together with the PoW solution for verification.

Motivated by the need to decrease the bandwidth overhead of Permacoin, Sengupta et al. [25] propose Retricoin. At a high level, Retricoin modifies the mining protocol of Permacoin not to involve file chunks during the computation of the PoW. This permits to verify a candidate PoW solution without any file chunk, decreasing the network bandwidth. Retricoin uses the PoW as a form of commitment, *i.e.,* it selects the file chunk indexes that must be proven. Then, it leverages the *proof of retrievability* (PoR) scheme of Shacham and Waters [26] to prove the possession of that chunks. Retricoin maintains Permacoin's objective of hindering peers from outsourcing files to third parties. While the authors state that Retricoin is secure against outsourcing, we observe that the modification made to the mining protocol makes it susceptible to an attack. We refer the reader to Section 3 for more details.

Armknecht et al. [3] propose a new PoW called EWoK (Entangled proofs of WOrk and Knowledge) that aims at increasing the decentralization and replication of the blockchain data by including the blockchain itself into the PoW. When EWoK is in place, the blockchain is divided into shards and the probability to compute a valid PoW solution is proportional to the number of independent shards the pool miner stores. The mining protocol is composed of two distinct PoW phases: The first is a standard PoW whose objective is to reach consensus among the pool miners on the set of transactions to include into the block. The second one is a special PoW that is computed over the shards so that a single solution can be expected on average. In this way, pool miners are incentivized to store different shards of the blockchain to maximize the probability of solving the second PoW.

Kopp et al. design KopperCoin [18], a new storage-based mining process that replaces the computationally expensive PoW. In KopperCoin, a user can submit a special *store* transaction that notifies miners of the intent to store a file in the system. Among other things, the store transaction contains the file chunks and the public information needed to verify the proofs of retrievability computed by the miners. Miners are free to choose which chunks to store. The storage based mining of KopperCoin is implemented by combining a PoR scheme and a bitwise XOR-based metric $f(x, y) = x \oplus y$. During the mining, the current block is hashed and the nearest chunk c_j (measured using the bitwise

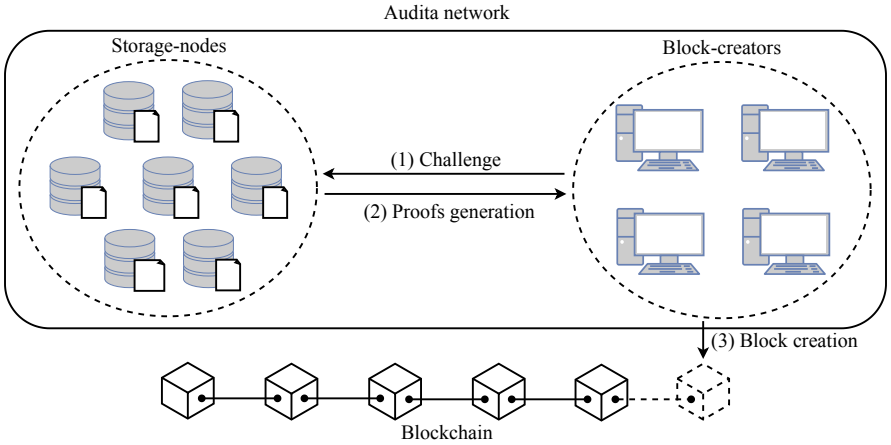

Figure 1: High level protocol workflow.

XOR-based metric) is selected, *i.e.*, $j = argmin_k\{H \oplus k|c_k$ set of chunks locally stored$\}$. Then, a miner generates a new block B (and collects the reward) if it outputs a valid proof of retrievability for the chunk c_j and $\mathsf{H}(B) \cdot 2^{|j \oplus h|} \leq threshold$ where h is the hash of the previous block and $threshold$ is the mining difficulty factor. The bitwise XOR-based measure allows miners to mine proportionally to the storage offered. Indeed, the more chunks a miner stores, the higher the probability is of finding a chunk index j close to h. We note that KopperCoin suffers from two significant drawbacks. First, it does not guarantee that a file is entirely stored. This is because miners can select which chunks to store. Hence, malicious miners could collude and choose not to store a certain portion of the file. Second, the store transaction contains the file chunks. This implies that a copy of the file is stored on the blockchain. As discussed in the introduction (Section 1), this creates several stumbling blocks (*e.g.,* in relation to scalability and file deletion), and makes KopperCoin unusable in our context.

The systems proposed in [19], [24], [28] adopt a different approach. They leverage the blockchain as a broadcast channel that allows users to publish orders (or contracts), rent storage and, distribute their files across the network. Sia [28] leverages a smart contract enabled blockchain to post arrangements between users and storage-nodes. When a new arrangement is posted, the file is sent off-chain to the corresponding storage-node. The arrangement contains several pieces of information such as: The Merkle root of the file, the expiration date, the challenge rate, and the reward for each challenge. At specific times (determined by the challenge rate), the storage-node samples a random chunk by hashing the current blockchain block and generates a proof of possession. The proof consists of the chunks and the list of hashes of the file's Merkle tree. Proofs are submitted to the blockchain network and, if valid, an automatic payment is triggered that compensate the storage-node as specified in the arrangement. A major drawback of Sia is the usage of Merkle trees. Although they allow verification of large data, they are not meant to be used as a form of proof of possession since a verification proof must contain the challenged chunk (along with a logarithmic number of hashes). This significantly increases the network bandwidth since chunks must be sent to other peers during the verification phase.

Protocol Labs proposes Filecoin [19] that leverages the blockchain to implement two decentralized markets: The storage market, where users submit orders to pay storage-nodes and rent their storage, and the retrieval market, where users submit requests to retrieve a file previously stored. Storing and retrieving files involve matching orders between users and storage-nodes. Files are sent off-chain to storage-nodes that later generate a sequence of *proofs of spacetime* to prove files were stored during the stipulated period. Storage-nodes are paid using off-chain payment channels when they provide valid proofs. File retrieval works similarly: Users submit retrieval requests to nodes by including payments that are exercised off-chain only when proofs of spacetime are valid or when files have been successfully retrieved.

Blockstore [24] also employs the blockchain as a decentralized storage market, in which users can rent space from storage-nodes. It differs from [19], [28] on how the auditing tasks are handled. Indeed, in [24], users are responsible for their data and have to audit it by themselves or collaborate with a third party. Namely, users repeatedly challenge storage-nodes and ask for proofs of possession.

Storj [30] is an open-source project that aims to build a decentralized cloud storage service. The core of the system is composed of a class of peers, called satellites. They are responsible for implementing a distributed hash table (DHT) that stores all the information needed by users to upload and download files to/from the network (e.g., storage-nodes location, storage-nodes reputation, file metadata, etc.). Additionally, they issue payments to storage-nodes, which receive an ERC20 Ethereum-based token, according to the service provided, and they are responsible for data maintenance and data replication. As in [24], auditing tasks are performed either by the user or a third party (in this case the satellites). As in Sia, Storj suffers from high network bandwidth overhead since it uses Merkle trees as proofs of possession.

3. Properties

In Section 3.1, we give an overview of the fundamental properties that a blockchain-based decentralized

	PoW-free	Decentralized auditing	Fair auditing	Resilient to outsourcing
Permacoin [20]	○	●	○	●
Retricoin [25]	○	●	○	○
EWoK [3]	○	●	○	●
KopperCoin [18]	●	●	○	○
Sia [28]	●	●	◐	○
Filecoin [19]	●	●	◐	?
BlockStore [24]	●	○	◐	○
Storj [30]	●	○	◐	○
Audita (Sec. 7)	●	●	●	●

Table 1: Comparison between Audita and the existing state-of-the-art systems. ● Property is satisfied. ○ Property is not satisfied. ◐ Property is partially satisfied. Question mark ? indicates insufficient implementation details. The highlighted systems present the drawback of either store file chunks on the blockchain or send them across the network for verification purposes.

storage system must satisfy. We make a comparison between Audita and the existing systems (described in Section 2) based on such properties. For completeness, in Section 3.2, we report some additional factors that must be considered before deploying such systems.

3.1. Fundamental properties

We refer the reader to Table 1 for the comparison summary between Audita and current state-of-the-art systems.

PoW-free. The current decentralized storage systems can be divided into two distinct categories according to how they prove the integrity of the files: PoW-based and PoW-free. Systems such as [3], [20], [25] are PoW-based: Nodes are required to spend a significant amount of their computational power to compute the PoW.[1] On the other hand, [18], [19], [24], [28], [30] do not rely on PoW at all: Nodes are only required to generate proofs of possession. A decentralized storage system should not make use of PoW for two main reasons: It results in a significant waste of electricity, and it excludes from the network nodes with limited computational power but available storage space. Audita belongs to the PoW-free category: A storage-node must only provide its storage and have access to modest network connectivity to interact with other peers. Note, however, that block-creators could still deploy PoW as their consensus mechanism if deemed suitable.

Decentralized auditing and fairness. Data auditing is the core of a decentralized storage system. Once users upload their files, they expect to receive assurance that files are stored correctly, and their integrity is preserved. The auditing process is composed of two distinct aspects that we call *decentralization* and *fairness*.

Decentralization refers to how the system audits the data. Resorting to the file owners for auditing activities (via PDP or PoR) is not acceptable in a decentralized scenario. Users can be offline or fail to trigger audit events. Auditing must be decentralized and automatically performed by the system. Blockstore [24] and Storj [30] are centralized: The user authorizes a third party to audit its data by continually challenging storage-nodes. On the other hand, [3], [18]–[20], [25], [28] are decentralized. In particular, [3], [20], [25] are PoW-based, and nodes challenge themself by computing the PoW. A similar consideration holds for KopperCoin [18] where, instead of the PoW, nodes execute a storage-based mining algorithm. Lastly, in [19], [28], storage-nodes regularly generate proofs of possession to fulfill the agreement established with the user.

Fairness refers to the ability to uniformly audit every part of the file uploaded. In other words, the auditing is fair when each file chunk has the same probability of being checked. Systems such as [3], [18], [20], [25] are *not* fair under this definition. In KopperCoin [18], storage-nodes are free to select which chunks to store and could merely choose not to preserve some of them. While, in [3], [20], [25], the probability of challenging a node is proportional to its hash power since PoW is used to prove storage integrity.

Instead, systems [19], [24], [28], [30] partially satisfy fairness because file owners are in charge of selecting storage nodes and challenge rates in advance.

Audita provides decentralized auditing that is decentralized and fair. Storage-nodes and their respective integrity challenges are uniformly sampled by hashing the output of the election phase that determines the block-creator.

Resilience to outsourcing. Outsourcing refers to the possibility for a node to store data within a cloud storage service (*e.g.*, Amazon, Google, Microsoft, etc.) to release its memory. If several nodes outsource their data, decentralization will not be guaranteed. A reliable decentralized storage service should tackle this issue by hindering peers that outsource their data. As an example, systems such as [3], [20] force peers to use their local storage by including both the data and the peer's secret key in the PoW process. On the other hand, [18], [24], [28], [30] do not propose any defense against outsourcing. Filecoin [19] will block outsourcing by leveraging a *proof of spacetime* and a *time-bounded proof of replication*, but details about these primitives are still work in progress. Lastly, Sengupta et al. [25] state that Retricoin is secure against outsourcing. However, we note that file chunks are not directly hashed during the computation of the PoW (as it is done in Permacoin). Instead, the PoW involves only the indexes of the chunks that are challenged. Thus, a malicious miner could simply compute the PoW independently, and have the proof of retrievability generated by an external server.

Audita proposes a new solution for the storage outsourcing problem by making it financially inconvenient. A block-creator will eventually ask and wait from storage-nodes ℓ proofs of possession $\{\pi_i\}_{i\in\ell}$. A reward is only assigned to the first ℓ storage-nodes that provide such proofs. This consistently puts storage-nodes in a highly competitive state. The reason is that block-creators have the incentive to include in the winning ticket the first ℓ proofs π_i received to avoid any delay in getting the reward. If a storage-node decides to outsource its data, it will need to contact an external server to retrieve π_i. The assumption we make here is that this will cause network delays which decrease the probability of getting the reward. The advantage of our technique is that it does not involve any intensive computational task from

1. Retricoin [25] leverages PoW to select the chunks indexes to challenge.

the storage-nodes' side. Note that a similar approach is used by Bowers et al. [9]. They show that the response time permits to verify that a server is storing a file in a fault-tolerant manner.

3.2. Additional considerations

In this Section, we discuss about problems that must be tackled before deploying a blockchain-based decentralized storage system.

Incentivization. In a decentralized storage service, peers offer their local storage space to build the network storage infrastructure. Hence, one critical aspect is how to incentivize peers to join the network. Peers must be persuaded to provide their storage and keep data intact. Every existing system [3], [18]–[20], [24], [25], [28], [30] uses rewards to encourage peers to participate in the storage process honestly. In more detail, in [3], [18], [20], [25], the system automatically generates rewards to peers that correctly store the file. As an example, [3], [20], [25] are conceived to work with Bitcoin, and peers receive a bitcoin coinbase transaction as their reward. A different approach is adopted by [19], [24], [28], [30]: Users pay nodes in exchange for their storage space. In Audita, at each timestamp, a block-creator asks storage-nodes for proofs of possession on their respective file chunks. A reward is assigned only to the first ℓ storage-nodes that answer correctly. The goal of Audita is to incentivize storage-nodes (1) to store file chunks faithfully *and* (2) to actively provide valid proofs of possession. Indeed, unlike other systems, Audita creates a strong incentive for storage-nodes to compete with each other and provide valid cryptographic proofs promptly.

Malicious users. The incentive that a blockchain-based storage system provides to invite nodes to participate can become a double-edged sword. Clients and nodes can collude to take advantage of the network. As mentioned in [19], a malicious user can generate a large file through a program. By sharing this program with a node, the latter can free its storage space but still claim to be storing the file. Specifically, let us assume that a malicious user requests to store a file F generated in the following way:

- Choose a PRF G with key $s \leftarrow\!\!\$\ \{0,1\}^*$.
- Create an arbitrary long file $F = \{f_1, \ldots, f_n\}$ where $f_i = G_s(i)$.
- Make a request to store the file F in the network.
- Collude by sharing the G_s with the nodes that are storing the file.

If the data owner spends less money to store his file than the rewards the storage-node gets, then their collusion will generate significant earnings. A system that could suffer from this problem is KopperCoin [18]. In KopperCoin a user, that wants to store a file for a fixed period must destroy a fixed amount of coins. The longer the period, the higher the number of coins to destroy. On the other side, storage-nodes will earn some new coins that are included in a coinbase transaction. The authors do not mention any relationship between the number of coins to destroy and the amount contained into the coinbase transaction. Hence, KopperCoin could be susceptible to the above attack if the amount paid by the user is lower than the expected rewards that a storage-node can earn. On the other hand, [3], [20], [24], [25], [28], [30] are not susceptible to this attack. Miller et al. [20], and Sengupta et al. [25] assumes that a *trusted* dealer owns the file F. Hence, the attack is out of scope since F comes from a trusted party. A similar argument holds for [3]: F is the Bitcoin blockchain which is publicly verifiable, and thus implicitly trusted. Filecoin [19] mentions that their *time-bounded proof of replication* and *proof of spacetime* prevent the attack, but their solution is still being worked out. Lastly, in [24], [28], [30], the amount paid by the user is precisely the amount that the storage-nodes will earn during the storage process. We stress that this approach applies to most of the systems, including Audita.

Recovery. A decentralized storage system must allow users to retrieve the file from the network. Unfortunately, merely deploying PDP and PoR is not sufficient. These primitives were designed to challenge cloud storage servers, but cannot help if providers misbehave or simply disappear [6]. Works such as [3], [24], [25], [28] do not provide any insight on how the file can be retrieved. Permacoin [20] assumes that a portion of altruistic peers may return the file to the user. However, without any incentives, the number of altruistic peers could slowly decrease over time, making the file unretrievable. On the other hand, [19], [30] propose two similar solutions that rely on the same idea: Clients pay storage-nodes to retrieve their files as an incentive. However, they do not describe how to fairly exchange the payment with the content of the file. Thus, in principle, the storage-node could receive the payment and disappear from the network. The solution of KopperCoin [18] involves a 2-out-of-2 multisignature. The user and the storage-node generates a transaction which includes a payment for the storage-node and two collaterals, one for each of the parties. Honest parties get their collateral back; in particular, when the user receives the file requested, then the multisignature is used to pay the storage-node and return the two collaterals to the respective parties.

Audita relies on similar techniques and leverages smart contract enabled fair exchange protocols [13] to implement a reliable file recovery mechanism.

4. Preliminaries

4.1. Notation

We use the notation $[n] = \{1, \ldots, n\}$. Capital boldface letters (such as $\mathbf{X}$) are used to denote random variables, small letters (such as x) to denote concrete values, calligraphic letters (such as $\mathcal{X}$) to denote sets and serif letters (such as A) to denote algorithms. For a string $x \in \{0,1\}^*$, we use $|x|$ to denote its length; if $\mathcal{X}$ is a set, $|\mathcal{X}|$ is the number of elements in $\mathcal{X}$. If A is an algorithm, we use $y = \mathsf{A}(x)$ to denote the run of A on input x and output y; If A is a randomized algorithm we write $\mathsf{A}(x;r)$ to denote the run of A on input x and uniform randomness r). We sometimes write $y \leftarrow\!\!\$\ \mathsf{A}(x)$ to denote the output y of the randomized algorithm A over the input x and uniformly randomness. The min-entropy of a random variable $\mathbf{X}$ is $\mathbb{H}_\infty(\mathbf{X}) = -\log \max_{x \in \mathbf{X}} \Pr[\mathbf{X} = x]$, and intuitively it measures the best chance to predict $\mathbf{X}$ (by a computationally unbounded algorithm).

Negligible functions. We denote by $\lambda \in \mathbb{N}$ the security parameter and we implicitly assume that every algorithm takes as input the security parameter. A function $\nu : \mathbb{N} \to [0, 1]$ is called *negligible* in the security parameter λ if it vanishes faster than the inverse of any polynomial in λ, *i.e.* $\nu(\lambda) \in \mathcal{O}(1/p(\lambda))$ for all positive polynomials $p(\lambda)$. We sometimes write $\mathsf{negl}(\lambda)$ (resp., $\mathsf{poly}(\lambda)$) to denote an unspecified negligible function (resp., polynomial function) in the security parameter.

4.2. Blockchain

A blockchain $\mathcal{C}$ is a sequence of public and immutable blocks administrated by the network. Each block B is composed of two parts: The state s and the data x. The state links B to its predecessor, while the data contains some arbitrary information (*e.g.*, transactions). At each timestamp t, the peers (called *block-creators*) collaborate (executing a series of algorithms and exchanging a sequence of messages) in order to agree on the next block B to append on the blockchain $\mathcal{C}$. A new block B is valid if it passes the following verification steps: 1) B is well-formed (*e.g.*, valid block structure, block size, etc.); 2) B is properly linked to the head of the blockchain $\mathsf{Head}(\mathcal{C})$ by means of a linking function link. If the block B is valid, then the network extends the chain $\mathcal{C}$ by appending B.

More formally, we define a blockchain $\mathcal{C}$ in the following way:

Definition 1 (Block)**.** A block B is a tuple (s, x) such that $s \in \{0,1\}^*$ and $x \in \{0,1\}^*$. s and x are called the block's state and the block's data, respectively.

Definition 2 (Blockchain)**.** A blockchain $\mathcal{C}$ is a sequence of blocks $B_1, \ldots, B_n$. The rightmost block is called the head of the chain, denoted with $\mathsf{Head}(\mathcal{C})$. The length of a blockchain $\mathsf{len}(\mathcal{C}) = n$ is its number of blocks. A blockchain $\mathcal{C}$ can also be empty, *i.e.*, $\mathcal{C} = \epsilon$. Let link be a function. Any blockchain $\mathcal{C}$ with head $\mathsf{Head}(\mathcal{C}) = (s, x)$ can be extended to a new longer chain $\mathcal{C}' = \mathcal{C}||B'$ by appending a new block $B' = (s', x')$ such that $s' = \mathsf{link}(\mathsf{Head}(\mathcal{C}))$.

A blockchain protocol Π over a chain $\mathcal{C}$ is a tuple of four algorithms $(\mathsf{KGen}, \mathsf{Elect}, \mathsf{CreateBlock}, \mathsf{Ver})$, executed by block-creators in order to create new blocks. The key generation algorithm KGen allows block-creators to join the network and generate their public and secret keys. We denote with $\mathcal{BC} = \{\mathsf{bc}_1, \mathsf{bc}_2, \ldots\}$ and $(\mathsf{pk}_{\mathsf{bc}_i}, \mathsf{sk}_{\mathsf{bc}_i})$, the block-creators present in the system and their public and secret keys, respectively. At each timestamp t, block-creators work to extend the blockchain by appending a new block B. The protocol is composed of two distinct phases: *Election phase* and *creation phase*. The election phase starts at the beginning of each timestamp t. Each block-creator executes Elect to reach consensus on a *leader* that will be in charge of creating and appending a new block. Once the consensus is reached, Elect outputs an identification string idstr that publicly identifies the leader bc^* (*e.g.*, idstr includes the leader's public key $\mathsf{pk}_{\mathsf{bc}^*}$). Then, the creation phase starts: bc^* broadcasts a new block B (generated by executing $\mathsf{CreateBlock}$) and the identification string idstr over the network in order to be verified. The blockchain is extended by appending B if and only if both B and idstr are valid, and B has been created by the leader bc^* identified by the identification string idstr.

More formally, a blockchain protocol consists of the following four algorithms:

$\mathsf{KGen}(1^\lambda)$**:** The randomized key generation algorithm takes as input the security parameter and outputs a public and secret key $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$.

$\mathsf{Elect}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$**:** The randomized consensus algorithm takes as input a public and secret key pair $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$ and outputs an identification string idstr.

$\mathsf{CreateBlock}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}}, \mathsf{idstr})$**:** The randomized creation algorithm takes as input the public and secret key pair $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$, and an identification string idstr, and outputs a block B.

$\mathsf{Ver}(B, \mathsf{idstr})$**:** The deterministic verification algorithm takes as input a block B and an identification string idstr, and outputs a decision bit.

We assume that leaders act in a rational manner, broadcasting new blocks as soon as are ready. This is a common behavior presents in most of the existing blockchains. Timestamps have a predefined time length that is the expected average time needed to run the protocol and produce new valid blocks. To make the system live, any delay is automatically handled by the network (*e.g.*, electing another leader). Furthermore, in blockchain systems such as Bitcoin and Ethereum, block-creators compete to mine new blocks and earn the corresponding coinbase transaction. Any voluntary delay would result in an economic loss.

We now define the correctness. Intuitively, a blockchain protocol Π is correct if an honest execution produces a valid block B.

Definition 3 (Correctness of blockchain protocol)**.** We say that a blockchain protocol Π is correct if, for every $\lambda \in \mathbb{N}$, set of block-creators $\mathcal{BC} = \{\mathsf{bc}_1, \mathsf{bc}_2, \ldots\}$ with keys $\{(\mathsf{pk}_{\mathsf{bc}_i}, \mathsf{sk}_{\mathsf{bc}_i})\}_{i \in [|\mathcal{BC}|]}$ we have:

$$\Pr\left[\begin{array}{l} \mathsf{idstr} \leftarrow\!\!\$\ \mathsf{Elect}(\mathsf{pk}_{\mathsf{bc}_i}, \mathsf{sk}_{\mathsf{bc}_i}), \\ B \leftarrow\!\!\$\ \mathsf{CreateBlock}(\mathsf{pk}_{\mathsf{bc}^*}, \mathsf{sk}_{\mathsf{bc}^*}), \\ \mathsf{Ver}(B, \mathsf{idstr}) = 1 \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)\,,$$

where $(\mathsf{pk}_{\mathsf{bc}_i}, \mathsf{sk}_{\mathsf{bc}_i}) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$, for $i \in [|\mathcal{BC}|]$ and $(\mathsf{pk}_{\mathsf{bc}^*}, \mathsf{sk}_{\mathsf{bc}^*})$ are the public and secret keys of the elected block-creator bc^* identified by idstr.

Remark 1. Network partitioning and asynchronous communication may interfere during the election and bring block-creators to have different consensus views, *i.e.*, Elect algorithm may return inconsistent identification strings idstr. This ends in having multiple leaders that create new blocks, generating blockchain forks. We implicitly assume that the system has a mechanism to handle and solve forks automatically.

Remark 2. Our definition focuses on blockchain protocols that are composed of two distinct phases: Election phase (Elect algorithm) and creation phase ($\mathsf{CreateBlock}$ algorithm). An example of systems that lie in this category are permissioned blockchains. In a permissioned setting, block-creators could jointly elect a leader (*e.g.*, by running a cooperative consensus algorithm) that will be in charge

of generating a new block B. Even the well-know permissionless blockchains Bitcoin and Ethereum fall in this category. In these systems, the block creation works as a form of *self-election*. Indeed, block-creators are required to locally solve the PoW in order to produce a valid block. The coinbase transaction of a newly mined block uniquely identifies the creator. Hence, according to our definition, the election and creation phases collapse into a single one, *i.e.*, the mined block can be seen as an identification string (*e.g.*, $\mathsf{idstr} = B$), and the creation algorithm $\mathsf{CreateBlock}$ is just the identity function.

We are interested in blockchain protocols that are unpredictable. Unpredictability refers to the inability to predict the output of the election phase, *i.e.*, the identification string idstr.

Definition 4 (Unpredictability)**.** A Blockchain Protocol Π is unpredictable if:

$$\mathbb{H}_\infty(\mathbf{X}) \geq \omega(\log(\lambda)),$$

where $\mathbf{X}$ is a random variable that describes the distribution of the identification strings idstr (output by Elect).

Remark 3. Several blockchain systems are considered unpredictable. For example, as described in Remark 2, Bitcoin's identification string idstr corresponds to the next candidate block B. Each block contains several elements such as: 32 bit nonce used to randomize the output of the PoW, ECDSA signatures, Merkle root, extra-nonce, etc. These elements have a non-trivial amount of entropy and make hard the prediction of the next block. We also emphasize that, in some cases, it is straightforward to make a blockchain system unpredictable. For example, in a blockchain that deploys a cooperative consensus algorithm (*e.g.*, Ripple [11]), block-creators can jointly agree on randomness r (*e.g.*, by using coin tossing [1] or other multi party computation techniques) with at least $\omega(\log(\lambda))$ bits of min-entropy during the consensus phase.

4.3. Provable Data Possession

A Provable Data Possession scheme (PDP) $\Pi = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{GenChal}, \mathsf{GenProof}, \mathsf{CheckProof})$ allows a user to check the integrity of a file $F = \{f_1, \ldots, f_n\}$ stored in a remote untrusted server. The user computes the file *fingerprint* (by using its public and secret key $(\mathsf{pk}, \mathsf{sk})$ generated by KGen) that consists in tagging each file chunk f_i using the tagging algorithm Tag. Then, both the file F and the tags $\{\tau_i\}_{i\in[n]}$ are outsourced to the untrusted server. At any moment, the user can issue a challenge chal (generated through the challenge generation algorithm $\mathsf{GenChal}$) to the server in order to audit its data. We assume that the challenge generation algorithm takes in input an integer d and an index space $\mathcal{I} \subseteq [n]$ such that $d \leq |\mathcal{I}|$, and returns chal that, among other things, includes a set of d distinct indexes $\mathcal{I}_{\mathsf{chal}}$ sampled at random from $\mathcal{I}$. The server uses both the file and the related tags to run $\mathsf{GenProof}$ and generate a proof of possession π. The latter is sent to the user that verifies it by running the verification algorithm $\mathsf{CheckProof}$. In this work, we are interested in publicly verifiable PDP, *i.e.*, the verification process does not involve the user's secret key sk. This permits to delegate a third party for verification.

Formally, a publicly verifiable PDP scheme for a file $F = \{f_1, \ldots, f_n\}$ is composed by the following algorithms:

$\mathsf{KeyGen}(1^\lambda)$**:** The randomized key generation algorithm takes as input the security parameter and outputs the public key pk and the secret key sk.

$\mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, f_i)$**:** The randomized tagging algorith takes as input a public key pk, a secret key sk, and a file chunk $f_i \in F$, and outputs a tag τ_i.

$\mathsf{GenChal}(d, \mathcal{I})$**:** The randomized challenge algorithm takes as an integer $d \in \mathbb{N}$ and an index space $\mathcal{I} \subseteq [n]$, and outputs a challenge chal defined over d distinct chunk indexes $\mathcal{I}_{\mathsf{chal}}$ sampled at random from $\mathcal{I}$

$\mathsf{GenProof}(\mathsf{pk}, \mathsf{chal}, \{f_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}, \{\tau_i\}_{i\in\mathcal{I}_{\mathsf{chal}}})$**:** The deterministic prove algorithm takes as input a public key pk, a challenge chal defined over a set of d chunk indexes $\mathcal{I}_{\mathsf{chal}}$, a set of file chunks $\{f_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}$, and a set of tags $\{\tau_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}$, and outputs a proof π.

$\mathsf{CheckProof}(\mathsf{pk}, \mathsf{chal}, \pi)$**:** The deterministic verification algorithm takes as input a public key pk, a challenge chal, and a proof π, and outputs a decision bit.

A PDP scheme is correct if honestly generated proofs verify correctly.

Definition 5 (Correcteness of PDP)**.** A PDP scheme $\Pi = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{GenChal}, \mathsf{GenProof}, \mathsf{CheckProof})$ is correct if $\forall \mathcal{I} \subseteq [n]$, $\forall F = \{f_1, \ldots, f_n\}$, $\forall d \in \mathbb{N}$ such that $f_i \in \{0,1\}^*$ and $d \leq |\mathcal{I}|$, we have:

$$\Pr\left[\begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(1^\lambda),\\ \mathsf{chal} \leftarrow_\$ \mathsf{GenChal}(d, \mathcal{I})\\ \tau_i \leftarrow_\$ \mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, f_i), \text{ for } i \in \mathcal{I}_{\mathsf{chal}},\\ \pi = \mathsf{GenProof}(\mathsf{pk}, \mathsf{chal}, \mathcal{F}, \mathcal{T}),\\ \mathsf{CheckProof}(\mathsf{pk}, \mathsf{chal}, \pi) = 1\end{array}\right] = 1,$$

where $\mathcal{I}_{\mathsf{chal}}$ are the d chunk indexes determined by chal, $\mathcal{F} = \{f_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}$, and $\mathcal{T} = \{\tau_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}$.

As for security, it must be infeasible to generate a valid proof of possession π without knowing the file chunks specified into the challenge.

Definition 6 (Security of PDP)**.** A PDP scheme $\Pi = (\mathsf{KGen}, \mathsf{Tag}, \mathsf{GenChal}, \mathsf{GenProof}, \mathsf{CheckProof})$ is secure if, for every file $F = \{f_1, \ldots, f_n\}$, every index space $\mathcal{I} \subseteq [n]$, every $d \leq n$, and every PPT adversary A, the probability that A wins the game $\mathbf{G}^{\mathsf{pdp}}_{\Pi,F,\mathsf{A}}(\lambda, d, \mathcal{I})$ is negligibly close to the probability that the challenger can extract the chunks $\{f_i\}_{i\in\mathcal{I}_{\mathsf{chal}}}$ by means of a knowledge extractor E. The game $\mathbf{G}^{\mathsf{pdp}}_{\Pi,F,\mathsf{A}}(\lambda, d, \mathcal{I})$ is defined in the following way:

1) The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow_\$ \mathsf{KGen}(1^\lambda)$ and sends pk to the adversary A.
2) A issues oracle queries to the oracle Tag. On input $f_i \in F$ the oracle returns the tag $\tau_i \leftarrow_\$ \mathsf{Tag}(\mathsf{pk}, \mathsf{sk}, f_i)$.
3) The challenger sample a randomness $r \leftarrow_\$ \{0,1\}^*$ and send it to A. The adversary is required to provide a valid proof of possession π with respect to the challenge $\mathsf{chal} = \mathsf{GenChal}(d, \mathcal{I}; r)$ defined over the d indexes $\mathcal{I}_{\mathsf{chal}} \subseteq \mathcal{I}$.
4) The adversary outputs a proof π.

5) If $\mathsf{CheckProof}(\mathsf{pk}, \mathsf{chal}, \pi) = 1$ where $\mathsf{chal} = \mathsf{GenChal}(d, \mathcal{I}; r)$, output 1; otherwise output 0.

Remark 4. The PDP definitions used in this work differs from the ones proposed by Ateniese et al. [5] in how the challenges are generated. We assume the existence of a randomized algorithm GenChal that produces challenges on a portion of the file. This main difference affects our security definition. Indeed, similarly to [7, Section 4], the challenger does not directly choose chal, but it outputs the randomness r for the challenge algorithm GenChal that will randomly sample d chunks from the index space $\mathcal{I}$. Note that every secure PDP scheme under the definition of [5], is also secure under our Definition 6. This because in [5], the challenge distribution is arbitrary.

Remark 5. PDP and PoR [26] are related primitives. The former guarantees that the challenged chunks are correctly stored (or known) by the server, while the latter additionally guarantees that the entire file can be retrieved from a set of proofs.

It's possible to show that retrievability can be added to PDP by applying an erasure code to the file F before uploading it [4]. Thus, we implemented Audita with PDP and left the use of erasure codes optional.

4.4. Signature Schemes

A signature scheme is made of the following efficient algorithms.

$\mathsf{KGen}(1^\lambda)$**:** The randomized key generation algorithm takes the security parameter and outputs a public and a secret key $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Sign}(\mathsf{sk}, m)$**:** The randomized signing algorithm takes as input the secret key sk and a message $m \in \mathcal{M}$, and produces a signature σ.

$\mathsf{Ver}(\mathsf{pk}, m, \sigma)$**:** The deterministic verification algorithm takes as input the public key pk, a message m, and a signature σ, and it returns a decision bit.

A signature scheme should satisfy two properties: 1) honestly generated signatures verify correctly and, 2) it is infeasible to compute valid signatures for new fresh messages without knowing the respective secret key sk.

Definition 7 (Correctness of signatures)**.** A signature scheme $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ with message space $\mathcal{M}$ is correct if $\forall m \in \mathcal{M}, \forall (\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$, the following holds:

$$\Pr[\mathsf{Ver}(\mathsf{pk}, m, \mathsf{Sign}(\mathsf{sk}, m)) = 1] = 1.$$

Definition 8 (Unforgeability of signatures)**.** A signature scheme $\Pi = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Ver})$ is existentially unforgeable under chosen-message attacks (EUF-CMA) if for all PPT adversaries A:

$$\Pr\left[\mathbf{G}^{\mathsf{euf}}_{\Pi,\mathsf{A}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)\,,$$

where $\mathbf{G}^{\mathsf{euf}}_{\Pi,\mathsf{A}}(\lambda)$ is the following experiment:

1) $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}(1^\lambda)$.
2) $(m, \sigma) \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{Sign}(\mathsf{sk},\cdot)}(1^\lambda, \mathsf{pk})$
3) If m has not been queried to oracle $\mathsf{Sign}(\mathsf{sk}, \cdot)$, and $\mathsf{Ver}(\mathsf{pk}, m, \sigma) = 1$, output 1; otherwise output 0.

5. Security Assumptions

Audita allows users to store their files in a decentralized way. For the sake of clarity, we introduce the system assuming the presence of a single user with a single file. Erasure code may be pre-applied in order to add redundancy and guarantees a user to retrieve the file even if a part of the chunks are lost or corrupted. We assume the file size is too large (*e.g.*, Petabytes) to be stored by an individual node. For this reason, the file is divided into smaller portions that are distributed to each node. The distribution is performed by a third party D, called the *dealer*. In Section 6.1 we discuss how to handle multiple files and decentralize the distribution. Additionally, we make the following assumptions:

Rational nodes. We assume that the majority of nodes (block-creators and storage-nodes) are rational and act in an economically rational manner. Rewards are widely used in several blockchain systems to incentivize honest behavior. In both Bitcoin and Ethereum, peers earn coins for the service provided to the network. Incentivization can also be obtained in other ways. As an example, in a permissioned scenario, the network is composed of authorized nodes. It is reasonable to assume that, in this case, nodes have a contractual agreement with an authority that must be fulfilled.

Secrecy of private keys. Each node possesses a public and secret key $(\mathsf{pk}, \mathsf{sk})$. We assume sk is kept secret, and thus not outsourced to an external party. In permissionless systems such as Bitcoin or Ethereum, the node's secret key is used to sign transactions and spend their rewards. Hence, a node that reveals its secret key sk would end in exposing its wallet. On the other hand, in a permissioned setting, nodes must be authorized and sk allows them to authenticate in the system. Revealing sk would allow an attacker to maliciously act in its name and breaks any existing contractual agreement.

Network latency. We assume that communicating through the network requires a significant amount of time. If a node desires to execute a computational task as fast as it can, it will be most likely faster if it minimizes the network communication by computing the task locally. Outsourcing the computation to an external party will add a significant delay due to the multiple network hops necessary to exchange inputs and outputs for the task.

6. Audita

Audita defines a new technique to add storage capabilities to every blockchain system (Definition 4.2). The network is composed of two types of nodes: Block-creators and storage-nodes. Block-creators execute the standard protocol of the underlying blockchain while storage-nodes are entitled to store a portion of the file. Audita requires cooperation between these two categories in order to create a new block. A block-creator that wants to extends the blockchain with a new block B must challenge a subset of k storage-nodes and retrieve at least ℓ proofs of possession $\{\pi_i\}_{i\in[\ell]}$.

Formally, Audita is composed of eight algorithms $(\mathsf{BCKGen}, \mathsf{SNKGen}, \mathsf{Setup}, \mathsf{GetChunks}, \mathsf{Elect}, \mathsf{Prove}, \mathsf{CreateBlock}, \mathsf{Ver})$ and consists of four distint phases: *Join*, *distribution*, *election*, and *block creation*.

Join. As in every blockchain system, nodes are free to join the network. They only need to generate a valid key pair. Since there are two categories of nodes, Audita have two distinct key generation algorithms: BCKGen generates the block-creators' keys while SNKGen generates the storage-nodes' ones. We denote with $\mathcal{SN} = \{\mathsf{sn}_1, \mathsf{sn}_2, \ldots\}$ and $\{(\mathsf{pk}_{\mathsf{sn}_i}, \mathsf{sk}_{\mathsf{sn}_i})\}_{i\in[|\mathcal{SN}|]}$, the storage-nodes present in the system and their public and secret keys, respectively.

Distribution. The distribution phase starts with a user that wants to store a file $F = \{f_1, \ldots, f_n\}$. It executes Setup that outputs an encoding $\hat{F} = \{\hat{f}_1, \ldots, \hat{f}_n\}$ of F and the file public key $\mathsf{pk}_{\hat{F}}$ (*i.e.*, file identifier). Among other things, the encoding $\hat{F}$ contains a set of tags $\{\tau_i\}_{i\in[n]}$, computed by a publicly verifiable PDP scheme. The tags will allow storage-nodes to prove the file is correctly stored. The user publishes $\mathsf{pk}_{\hat{F}}$ to announce its intention to store the file (see Section 6.1 for more details). Then, it sends $\hat{F}$ to the dealer D whose job is to distribute to each storage-node sn a subset $\hat{F}_{\mathsf{sn}} \subseteq \hat{F}$ (computed by GetChunks) of m chunks. An example of distribution is depicted in Figure 2.

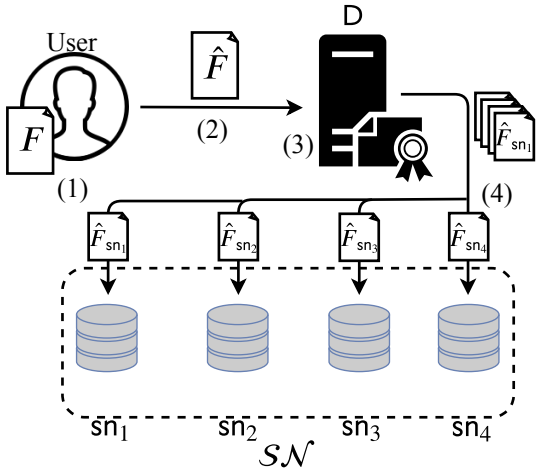

Figure 2: An example of file distribution with four storage-nodes $\mathcal{SN} = \{\mathsf{sn}_1, \mathsf{sn}_2, \mathsf{sn}_3, \mathsf{sn}_4\}$. (1) The user, holding a file F, executes Setup and computes the file public key $\mathsf{pk}_{\hat{F}}$ and the encoding $\hat{F}$. (2) The user sends the encoded file $\hat{F}$ to the dealer D. (3) D computes the file subsets $\{\hat{F}_{\mathsf{sn}_1}, \hat{F}_{\mathsf{sn}_2}, \hat{F}_{\mathsf{sn}_3}, \hat{F}_{\mathsf{sn}_4}\}$ by running GetChunks. (4) D sends to each storage-node sn_i the respective file subset $\hat{F}_{\mathsf{sn}_i}$ to store.

Election. Once the distribution phase is completed, the election phase starts. At each timestamp, block-creators try to reach consensus on a leader bc^*. This is achieved by executing the election algorithm Elect that outputs an identification string idstr that identifies bc^*. Furthermore, k distinct storage-nodes $\mathcal{SN}^*$ are elected. Each elected storage-node is invited to prove the possession of its file portion by using the PDP scheme.

Creation. The leader bc^* sends to each elected storage-node $\mathsf{sn}_i^* \in \mathcal{SN}^*$ the identification string idstr to prove that it is the leader for the current timestamp. Then, each storage-node $\mathsf{sn}_i^* \in \mathcal{SN}^*$ sends back a proof of possession π_i (computed by executing Prove) with respect to a challenge chal_i^* defined over d chunk indexes $\mathcal{I}_{\mathsf{chal}_i^*}$. The challenge chal_i^* is randomly generated by hashing the identification string idstr and the storage-node's public key $\mathsf{pk}_{\mathsf{sn}_i^*}$.

The leader bc^* broadcasts the identification string idstr, a new block B (generated by CreateBlock), and the first received ℓ proofs $\{\pi_i\}_{i\in[\ell]}$. Each node in the network executes the verification algorithm Ver that checks the following: i) idstr is a valid identification string that identifies bc^*, ii) B is valid block created by bc^*, iii) $\{\pi_i\}_{i\in[\ell]}$ are valid proofs of possession generated by ℓ distinct elected storage-nodes. If the verification succeeds, then the network extends the blockchain $\mathcal{C}$ by appending the block B. Figure 3 shows a creation phase execution. We stress that the ℓ valid proofs of possession are not included into the new block B. If the majority of block-creators is honest, then the creation of a new block implies that $\ell \cdot d$ chunks have been proven correctly.

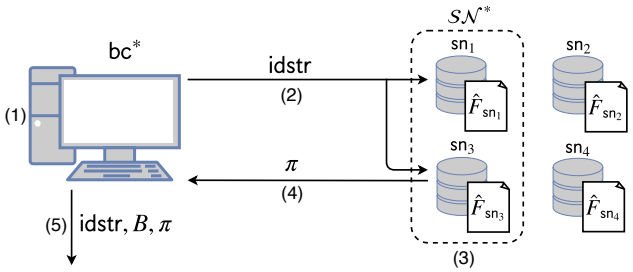

Figure 3: Creation phase execution with four storage-nodes $\mathcal{SN} = \{\mathsf{sn}_1, \mathsf{sn}_2, \mathsf{sn}_3, \mathsf{sn}_4\}$, $k = 2$, and $\ell = 1$. (1) The network elects a leader bc^* and a set of k storage-nodes $\mathcal{SN}^* = \{\mathsf{sn}_1, \mathsf{sn}_3\}$ by running Elect. (2) The leader bc^* sends the identification string idstr to each storage-node $\mathsf{sn}_i^* \in \mathcal{SN}^*$. (3) Each $\mathsf{sn}_i^* \in \mathcal{SN}^*$ executes Prove and computes the proof π. (4) sn_3^* (the fastest storage-node) sends π to the leader bc^*. (5) bc^* broadcasts idstr, a new block B, and π for verification.

Below, we provide the formal instantiation of Audita.

Construction 1. Let BC, PDP, SS be a blockchain protocol, a public verifiable PDP scheme, and a signature scheme, respectively. Let $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^*$ be three distinct hash functions.

We build Audita with parameters (n, m, k, d, ℓ) such that $d \leq m \leq n$ and $\ell \leq k \leq |\mathcal{SN}|$ in the following way:

$\mathsf{BCKGen}(1^\lambda)$**:** The block-creator key generation algorithm, on input the security parameter, returns $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{BC}}(1^\lambda)$.

$\mathsf{SNKGen}(1^\lambda)$**:** The storage-node key generation algorithm, on input the security parameter, returns $(\mathsf{pk}_{\mathsf{sn}}, \mathsf{sk}_{\mathsf{sn}}) \leftarrow\!\!\$\ \mathsf{KGen}_{\mathsf{SS}}(1^\lambda)$.

$\mathsf{Setup}(1^\lambda, F)$**:** The setup algorithm, on input the security parameter and a file $F = \{f_1, \ldots, f_n\}$, computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KeyGen}_{\mathsf{PDP}}(1^\lambda)$ and $\tau_i = \mathsf{Tag}_{\mathsf{PDP}}(\mathsf{pk}, \mathsf{sk}, f_i)$, for $i \in [n]$. Finally, it returns $\mathsf{pk}_{\hat{F}} = \mathsf{pk}$ and $\hat{F} = \{\hat{f}_i = (f_i, \tau_i)\}_{i\in[n]}$.

$\mathsf{GetChunks}(\mathsf{pk}_{\hat{F}}, \hat{F}, \mathsf{pk}_{\mathsf{sn}})$**:** The chunk distribution algorithm takes as input the file public key $\mathsf{pk}_{\hat{F}}$, the encoded file $\hat{F} = \{\hat{f}_i = (f_i, \tau_i)\}_{i\in[n]}$, and a storage-node public key $\mathsf{pk}_{\mathsf{sn}}$. Then, it randomly samples without replacement m file chunk indexes in the

following way:[2]

1) Set $\mathcal{X}_{\mathsf{sn}} = \{\emptyset\}$, $\mathcal{V} = \{1, \ldots, n\}$, and $j = 0$.
2) Until $|\mathcal{X}_{\mathsf{sn}}| < m$ then:
   a) $j = j + 1$.
   b) $v_j = \mathsf{H}_1(\mathsf{pk}_{\mathsf{sn}}||j)$.
   c) If $v_j \notin \mathcal{X}_{\mathsf{sn}}$, then add v_j to $\mathcal{X}_{\mathsf{sn}}$

Finally, it returns $\hat{F}_{\mathsf{sn}} = \{\hat{f}_i = (f_{v_i}, \tau_{v_i})\}_{v_i \in \mathcal{X}_{\mathsf{sn}}}$.

$\mathsf{Elect}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$**:** The election algorithm, on input a block-creator public key and private key pair $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$, computes $\mathsf{idstr} \leftarrow\!\!\$\ \mathsf{Elect}_{\mathsf{BC}}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$. Then, it randomly samples without replacement k storage-nodes $\mathcal{SN}^*$ in the following way:[2]

(I) Set $\mathcal{SN}^* = \{\emptyset\}$, $\mathcal{V} = \mathcal{SN}$, and $j = 0$.
(II) Until $|\mathcal{SN}^*| < k$ then:
   a) $j = j + 1$.
   b) $v_j = \mathsf{H}_2(\mathsf{idstr}||j)$.
   c) If $\mathsf{sn}_{v_j} \notin \mathcal{SN}^*$, then add sn_{v_j} to $\mathcal{SN}^*$

Finally, it returns $(\mathsf{idstr}, \mathcal{SN}^*)$.

$\mathsf{Prove}(\mathsf{pk}_{\hat{F}}, \mathsf{pk}_{\mathsf{sn}}, \mathsf{sk}_{\mathsf{sn}}, \mathsf{idstr}, \hat{F}_{\mathsf{sn}})$**:** The prove algorithm takes as input the file public key $\mathsf{pk}_{\hat{F}}$, a storage-node public and secret key $(\mathsf{pk}_{\mathsf{sn}}, \mathsf{sk}_{\mathsf{sn}})$, the identification string idstr, and a subset of file chunks $\hat{F}_{\mathsf{sn}} = \{\hat{f}_i = (f_i, \tau_i)\}_{i \in [m]}$. It generates a challenge by executing $\mathsf{chal}^* = \mathsf{GenChal}_{\mathsf{PDP}}(d, \mathcal{X}_{\mathsf{sn}}; \mathsf{H}_3(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}}||\mathsf{idstr}))$ where $\mathcal{X}_{\mathsf{sn}}$ is the set of chunk indexes stored by sn (as described in GetChunks). Let $\mathcal{I}_{\mathsf{chal}^*} \subseteq \mathcal{X}_{\mathsf{sn}}$ be the set of d chunk indexes determined by chal^*. The algorithm computes $\sigma = \mathsf{Sign}_{\mathsf{SS}}(\mathsf{sk}_{\mathsf{sn}}, \pi')$ where $\pi' = \mathsf{GenProof}_{\mathsf{PDP}}(\mathsf{pk}_{\hat{F}}, \mathsf{chal}^*, \{f_j\}_{j \in \mathcal{I}_{\mathsf{chal}^*}}, \{\tau_j\}_{j \in \mathcal{I}_{\mathsf{chal}^*}})$. It returns a proof of possession $\pi = (\pi', \sigma)$.

$\mathsf{CreateBlock}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}}, \mathsf{idstr})$**:** The creation algorithm, on input a block-creator public and secret key pair $(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}})$ and an identification string idstr, runs $B \leftarrow\!\!\$\ \mathsf{CreateBlock}_{\mathsf{BC}}(\mathsf{pk}_{\mathsf{bc}}, \mathsf{sk}_{\mathsf{bc}}, \mathsf{idstr})$ and returns the new block B.

$\mathsf{Ver}(\mathsf{pk}_{\hat{F}}, \{\mathsf{pk}_{\mathsf{sn}_i}\}_{i \in [\ell]}, \{\pi_i\}_{i \in [\ell]}, \mathsf{idstr}, B)$**:** The algorithm takes as input the file public key $\mathsf{pk}_{\hat{F}}$, a set of storage-node public key $\{\mathsf{pk}_{\mathsf{sn}_i}\}_{i \in [\ell]}$, a set of proof of possession $\{\pi_i = (\pi'_i, \sigma_i)\}_{i \in [\ell]}$, an identification string idstr, and a block B. The algorithm proceeds in the following way:

- Compute $\mathcal{SN}^*$ as described in Elect (Item (I) and Item (II)).
- If there exists a $\mathsf{pk}_{\mathsf{sn}_i}$ that belongs to a storage-node sn_i such that $\mathsf{sn}_i \notin \mathcal{SN}^*$, then return 0.
- Otherwise, for every $i \in [\ell]$ proceed in the following way:
  - Compute the challenge $\mathsf{chal}^*_i = \mathsf{GenChal}_{\mathsf{PDP}}(d, \mathcal{X}_{\mathsf{sn}_i}; \mathsf{H}_3(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_i}||\mathsf{idstr}))$ where $\mathcal{X}_{\mathsf{sn}_i}$ is the set of chunk indexes stored by sn_i (as described in GetChunks, Item 1 and Item 2).
  - Compute $b^1_i = \mathsf{CheckProof}_{\mathsf{PDP}}(\mathsf{pk}_{\hat{F}}, \mathsf{chal}^*_i, \pi'_i)$ and $b^2_i = \mathsf{Ver}_{\mathsf{SS}}(\mathsf{pk}_{\mathsf{sn}_i}, \pi'_i, \sigma_i)$

Finally, if $1 = \mathsf{Ver}_{\mathsf{BC}}(\mathsf{idstr}, B)$ and $b^1_i = b^2_i = 1$ for every $i \in [\ell]$, the algorithm returns 1; otherwise, it returns 0.

2. Other randomized algorithms can be used to sample without replacement, *e.g.*, reservoir sampling [27].

Audita inherits from the PDP scheme the same security guarantee, *i.e.*, every time the chain is extended then a set of ℓ storage-nodes have provided ℓ valid proofs of possession. At high level, Theorem 1 means that the ℓ storage-nodes, that have produced the valid proof of possessions, know the corresponding challenged chunks (either because they store the chunks or they know how to recompute them). Below we establish the result, and the proof appears in appendix A.

Theorem 1. *Let t a timestamp in which the blockchain has been extended. Let $F = \{f_1, \ldots, f_n\}$ and $\mathcal{SN}^*$ an arbitrary file and the elected storage-nodes at timestamp t, respectively. If* PDP *is secure (Def. 6), then Audita guarantees (in the random oracle model) that there exists a set of ℓ storage-nodes $\{\mathsf{sn}_i\}_{i \in [\ell]} \subseteq \mathcal{SN}^*$ such that each sn^*_i has generated a proof of possession π_i with probability negligibly close to the probability that the user can extract the challenged d file chunks $\{f_j\}_{j \in \mathcal{I}_{\mathsf{chal}^*_i}}$ by means of a knowledge extractor* E*, where $\mathcal{I}_{\mathsf{chal}^*_i}$ are the indexes contained in the challenge chal^*_i of storage-nodes sn^*_i.*

6.1. Discussion

Publishing the parameters. Audita relies on two public parameters that the entire network must know: 1) An ordered list $\mathcal{SN}$ that contains the storage-nodes present in the system and, 2) the public key $\mathsf{pk}_{\hat{F}}$ of the file. The ordered list $\mathcal{SN}$ is essential in order to check that the proofs of possession $\{\pi_i\}_{i \in [\ell]}$ come only from elected storage-nodes $\mathsf{sn}^*_i \in \mathcal{SN}^*$. On the other hand, $\mathsf{pk}_{\hat{F}}$ is needed to validate the proofs $\{\pi_i\}_{i \in [\ell]}$. Audita introduces two types of transactions and leverages the blockchain to publish $\mathcal{SN}$ and $\mathsf{pk}_{\hat{F}}$. The first, called *join* transaction $\mathsf{T}_{\mathsf{join}}$, allows storage-nodes to join the network and publish their public keys. All the transactions recorded on the blockchain compose the ordered list $\mathcal{SN}$. The second is called *store* transaction $\mathsf{T}_{\mathsf{store}}$ and allows the user to publish the file public key $\mathsf{pk}_{\hat{F}}$.

Financial model. In order to block denial of service attacks, Audita charges the user to pay storage fees proportional to the time the file is stored. Suppose the user intends to store the file for t consecutive timestamps, then it must include $t\alpha$ coins in the store transaction $\mathsf{T}_{\mathsf{store}}$. These coins will be gradually delivered to the fastest storage-nodes of the next t timestamps, *i.e.*, for each timestamp, the leader includes into its proposed block ℓ transactions $\{\mathsf{T}_{\mathsf{sn}^*_i}\}_{i \in [\ell]}$, each of which transfers $\frac{\alpha}{\ell}$ coins from $\mathsf{T}_{\mathsf{store}}$ to one of the fastest ℓ storage-nodes. We stress that the payment can also be delivered in other ways. For example, in a permissioned setting, storage-nodes can have some kind of off-chain contracts.

This financial model makes Audita resilient against outsourcing. To extended the blockchain, leaders are required to broadcast ℓ proofs of possession along with the new block B. For this reason, a rational leader, that intends to broadcast a new block as fast it can (see Section 4.2), will wait only for the first ℓ incoming proofs. This incentivizes storage-nodes to keep their data locally stored. Indeed, outsourcing a significant portion of data to a third party will end in an economic loss. This because the proof generation will require communication through

the network making the storage-node significantly slower than the others (see the security assumptions in Section 5). Moreover, note that storage-nodes can not predict in advance which will be the challenged chunk indexes since they are computed by hashing the identification string idstr that has a non-trivial amount of entropy (Def. 4). We stress that Audita is outsourcing free only if $\frac{\ell}{k}$ is small enough. The higher the value (*e.g.*, $\frac{\ell}{k} \approx 1$), the lower the storage-node competition: A large portion of storage-nodes may decide to outsource the data since a high number of proofs are required during the creation phase. On the other hand, the lower the value (*e.g.*, $\frac{\ell}{k} \approx 0$), the higher the competition: The leader will collect only a limited number of proofs. Any delay significantly decreases the probability of getting the reward. Lastly, by requiring users to pay storage-nodes, Audita makes worthless any collusion strategy. Indeed, even if the user and a set of storage-nodes collude, their expected reward will be negative. This discourages any collusion and mitigates attacks such that the one described in Section 3.2.

To deploy this financial model, the block must be created after receiving the proofs of possession, *i.e.*, CreateBlock is executed after the election algorithm Elect. Unfortunately, this is not the case of the existing systems Bitcoin and Ethereum. As described in Remark 2, in these systems, the creation of the block is a form of self-selection (*i.e.*, Elect and CreateBlock collapse into a single algorithm). This does not permit the leaders to include the transaction $\{\mathsf{T}_{\mathsf{sn}_i^*}\}_{i\in[\ell]}$ into the new block since it would require to recompute the PoW. To overcome this problem, in this kind of systems, storage-nodes are paid in the next timestamp. Before proposing the block to the network, the leader must add into the transaction pool $\{\mathsf{T}_{\mathsf{sn}_i^*}\}_{i\in[\ell]}$ that pay the fastest ℓ storage-nodes $\{\mathsf{sn}_i^*\}$. After that, it broadcasts the block B along with the proofs $\{\pi_i\}_{i\in[\ell]}$, generated by $\{\mathsf{sn}_i^*\}_{i\in[\ell]}$. The network will accept the block B by additionally checking that the transaction pool contains $\{\mathsf{T}_{\mathsf{sn}_i^*}\}_{i\in[\ell]}$.

Recovering the file. As discussed in 3.2, a malicious storage-node sn can choose to do not return the stored file portion $\hat{F}_{\mathsf{sn}}$. This problem can be solved by adopting the solution proposed by KopperCoin [18]. The user and the storage-node sn create a 2-out-of-2 multisignature transaction T that includes three amounts: a payment α, a two collaterals β (the client one) and γ (the storage-node one). The collaterals β and γ are a form of warranty to encourage the parties to act honestly. Once the user receives $\hat{F}_{\mathsf{sn}}$ from the storage-node sn, they unlock T by signing a new transaction T′ that returns the collaterals to the respective parties and α is payed to sn. We stress that in smart contract enabled blockchains, other solutions such as fair exchange protocols [13] can be used.

Multiple files. Audita can be easily extended in order to store c files $\{\hat{F}_i\}_{i\in[c]}$ (with public keys $\{\mathsf{pk}_{\hat{F}_i}\}_{i\in[c]}$) of c different users. For each file $\hat{F}_i$, the dealer distributes to each storage-node sn a portion $\hat{F}^i_{\mathsf{sn}}$ of $\hat{F}_i$ composed of m chunks. As before, the portion $\hat{F}^i_{\mathsf{sn}}$ for a storage-node sn is computed by running GetChunks with input $(\mathsf{pk}_{\hat{F}_i}, \hat{F}_i, \mathsf{pk}_{\mathsf{sn}})$. During the creation phase, each elected storage-node $\mathsf{sn}_i^* \in \mathcal{SN}^*$ is now required to provide to the leader c different proofs of possession $\{\pi_{i,j}\}_{j\in[c]}$, one for each of the stored file portion $\hat{F}^j_{\mathsf{sn}_i^*}$. Lastly, a new block is accepted by the network only if it is accompanied by $\ell \cdot c$ PDP proofs $\{\pi_{i,j}\}_{i\in[\ell],j\in[c]}$ computed by ℓ elected storage-nodes.

Note that when multiple files are stored, the memory and computational power required to join the network increase proportional to the number of files stored. Indeed, each storage-node needs to store $c \cdot m$ chunks and must compute c independent proofs. To decrease these requirements, the parameters m, d, ℓ can be tuned in the following way:

- m (the number of chunks to store for each file) can be reduced (*e.g.*, $m' = \frac{m}{c}$) in order to decrease the amount of free space required to join the network.
- d (the number challenged chunks determined by chal) can be reduced to speed up the proof generation.
- When d decreases, the system checks the integrity of a smaller portion of the files. This can be counterbalanced by using the parallelism that Audita provides: By increasing ℓ (the number of proofs of possession required to propose a new block) is it possible to check the integrity of bigger portions of the files leveraging the proofs of possession of different storage-nodes. In Section 7.1 we will show the performance of Audita with different values ℓ.

Decentralized dealer. The dealer is just responsible for distributing the files across the network, allowing the user to go offline. Hence, it can be easily decentralized by replicating the dealer on multiple servers. Alternatively, it is possible to leverage a smart contract based blockchain to implement the dealer with a smart contract.

New storage-nodes. A storage-node could join the system after the file distribution has been completed. The (decentralized) dealer will serve these new storage-nodes by sending them the chunks to store. We stress that the dealer does not need to keep the file locally stored, indeed, it can simply retrieve the chunks from the storage-nodes already in the system. Lastly, we emphasize that a storage-node could refuse to send the requested chunks back since a new node in the system decreases its chance to be elected. To punish and disincentivize this behavior, the dealer marks the malicious node as faulty and excludes it from the system.

7. Implementation and Evaluation

We implemented a prototype to validate and demonstrate the technical feasibility of Audita. We implemented the system as a decentralized application via smart contract logic [10]. Smart contracts allow us to build a functioning system without modifying the consensus mechanism and the block structure of a forked blockchain. The choice to implement a decentralized application is justified by two main reasons: a pragmatic and a technical one. The pragmatic one is related to the opportunity to reduce the complexity of the implementation while proving the functioning of the protocol. The technical one is to be able to demonstrate that Audita can be built on a smart contract Enabled blockchain without touching the core protocol of the platform. Naturally, as already described, Audita can be implemented by modifying the protocol of an existing blockchain.

The client architecture (Figure 4) is structured with two main building blocks: a smart contract enabled

blockchain and a server component. The blockchain platform selected for our implementation is the Ethereum-based distributed ledger protocol Quorum [21]. The version of Quorum used in our implementation is the 2.1.1, the geth version used is 1.73, the consensus mechanism configured in our system is RAFT [23] and the smart contracts are written in Solidity 0.4.19. The server component is implemented in Node.Js 8.12.0 and has 5 main building blocks:

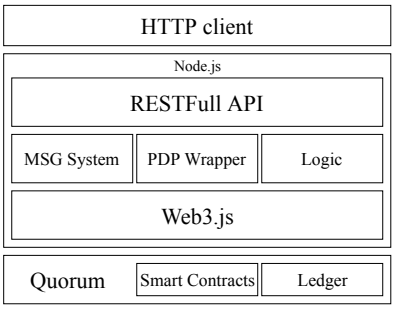

Figure 4: High level architecture.

- A RESTFull API Layer for HTTPS client interactions;
- A PDP Wrapper connected to the PDP subroutine;
- A messaging system for off-chain communication;
- Web3.js (1.0.0-beta.36 with a custom patch to overcome some limitation on WebSocket Handling);
- Participants logic libraries to implement specific logic for each of the different roles, *i.e.*, dealer, block-creator, and storage-node.

The PDP subroutine implements the publicly verifiable PDP scheme of Ateniese et al. [5], and it is based on an existing implementation called libpdp [14]. We modified the libpdp library to implement the variant of the primitive that offers public verifiability. The server component is the core logic of the different roles, and it is responsible for computing/generate proofs of possession using the library. Additionally, it allows off-chain communication to send/receive proofs as well as the chunks to store. In our implementation, the dealer (resp. the leader bc^* of a fixed timestamp) executes a dedicated smart contract to compute the chunk indexes a storage-node must store (resp. to compute the storage-nodes $\mathcal{SN}^*$ elected in a particular timestamp).[3] This makes the blockchain a public record enabling public auditing and transparency about the file distribution and the storage-node elections.

For the sake of efficiency, in our implementation, the election phase is simulated by an external party called *oracle*. The oracle beats the time for the network by communicating the start of new timestamps. A new timestamp starts when the oracle executes the election phase, implemented by a smart contract, that on input a random seed s, selects the leader bc^*. The leader will then use the same seed s to compute the elected storage-nodes $\mathcal{SN}^*$ and the challenge chal.[4] We stress that the oracle has been introduced only to reduce the complexity of the implementation.

3. The smart contract only computes the chunk indexes. The real chunks are sent off-chain.

4. According to our Definition 2, the identification string idstr of our implementation is the tuple $(\mathsf{pk}_{\mathsf{bc}^*}, s)$.

The hash function used in our implementation is SHA-3.

7.1. Evaluation

Experimental setup. We deployed 7 Quorum nodes as docker containers using Quorum Maker [17]. The host machine is an Amazon t3.2xlarge EC2 machine (8 vCPUs and 32GB of RAM), running Ubuntu Xenial 16.04 amd64. Therefore, the ledger is only replicated 7 times. The participants of the Audita network do not have their own copy of the ledger. The Audita network has been deployed with a set of 10 Amazon m5.12xlarge EC2 machines (48 cCPUs and 192 GB of RAM), running Ubuntu Xenial 16.04 amd64. Each virtual machine runs 100 storage-nodes, 2 block-creators and 2 dealers as dockers containers. One of the 10 virtual machines runs the oracle, as a docker container, that elects the leader at each timestamp. Overall, the Audita network is composed of: 1000 storage-nodes, 20 block-creators, 10 dealers, 1 oracle.

To optimize the duration of the file-sharing process on our experiments, we used the docker bind mount process [12] instead of the originally implemented HTTP protocol. Each storage-node container is bound to a folder on the host machine where the dealer copies the right chunks. This allows a dealer to successfully sends the appropriate chunks to the right storage-nodes (hosted on the same machine) without going through the HTTP protocol.

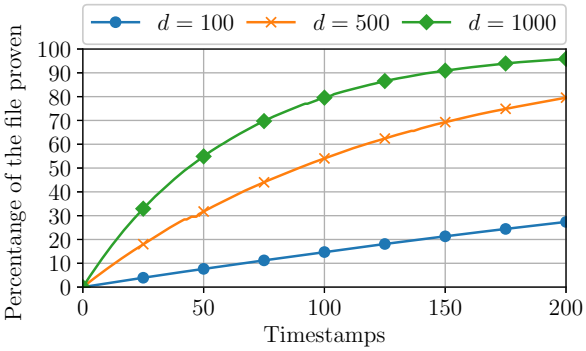

Figure 5: Percentage of the file proven over time.

Experimental results. We performed an experimental test and a simulation to evaluate the storage guarantees that Audita provides with respect to a single file. The results are identical even when the system stores multiple files. The execution time of the protocol heavily depends on the PDP scheme used and the underlying blockchain (block creation time, number of storage-nodes, etc.). We started with a performance test with a minimal instantiation of the system (small file and limited number of storage-nodes). We measured the relation between the number of timestamps and the percentage of the file that is being processed.

In more detail, we executed our test by considering the following parameters:

- 1GB file composed by $n = 65\,536$ chunks (chunk size 16KB).
- Number of chunks to store by storage-node $m = 12\,500$ ($\approx 19\%$ of the entire file).

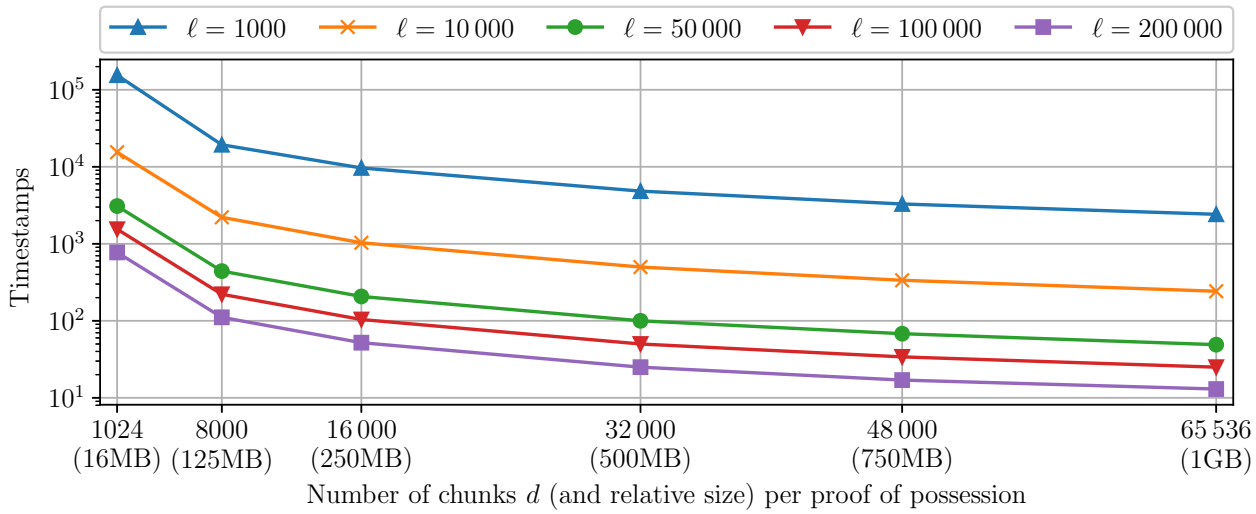

Figure 6: Number of timestamps needed to prove 90% of the 1 Petabyte file (composed of $n = 68\,719\,476\,736$ chunks). The Y-axis (timestamps number) follows a logarithmic scale.

- Number of proofs per timestamp $\ell = 1$.
- Number of elected storage-nodes per timestamp $k = 10$.

Based on these parameters, we ran three experiments with different values of d (number of chunks proven by each proof of possession): 100, 500, and 1000 (approximately 0.15%, 0.8%, and 1.6% of the file).

Figure 5 shows the results. As we can see, the parameter d has a significant impact on the percentage of the file chunks proven to be stored. For high d, the percentage of distinct file chunks proven grows logarithmically. For $d = 1000$ (*i.e.*, each proof of possession is computed on 1.6% of the total number of chunks), approximately 150 timestamps are sufficient to guarantee that 90% of the file is stored correctly. If the timestamps are 10 minutes long (*e.g.*, Bitcoin), $d = 1000$ guarantees that the 90% of the file is correctly stored only in 1 day of protocol execution. Additionally, our results allow us to determine the type of erasure code to use according to the user's preferences. For example, a $(0.9 \cdot n)$-out-of-n erasure code guarantees the retrievability of the file in 1 day while a $(n/2)$-out-of-n erasure code reduces the wait time to only 8 hours.[5]

Based on the results above, we ran a simulation to evaluate the performance of a large instantiation of Audita (large file and several storage-nodes). The simulation aims to show the impact of the parameter ℓ (number of proofs accepted at each timestamp).

In more detail, we deployed 1 Petabyte file composed of approximately 68 billion chunks (parameter n) distributed among 1 million storage-nodes, each of which entitled to store $m = 655\,360$ (10GB) chunks. This time, at each timestamp, $k = 400\,000$ storage-nodes are challenged on 1024 (16MB), 8000 (125MB), 16 000 (250MB), 32 000 (500MB), 48 000 (750MB), and 65 536 (1GB) chunks (parameter d). We ran the simulation with different values of ℓ (*i.e.*, 1000, 10 000, 50 000, 100 000, 200 000) and we report the number of timestamps needed to prove 90% of the file.

Figure 6 shows the results. By increasing ℓ, we can reduce the number of timestamps needed to prove a fixed percentage of the file (in this case, 90%). For example, for $d = 8000$ (125MB) and $\ell = 1000$, 90% is reached after 19 314 timestamps. Instead, for higher values such as $\ell = 10\,000$ and $\ell = 50\,000$, the number of timestamps drops to 2208 and 442, respectively. Moreover, note that the timestamps and the parameter ℓ are linearly correlated. As an example, between $\ell = 1000$ and the next order of magnitude $\ell = 10\,000$, the numbers of timestamps (19 314 and 2208, respectively) differ approximately by the same order.

As already discussed in Sec. 6.1, this shows that d and ℓ can be tuned to distribute the proof generation overhead among multiple storage-nodes while maintaining (or increasing) the system performance. Furthermore, the tuning can be adaptively performed by the Audita network (using a similar approach used by Bitcoin to change the PoW difficulty adaptively) according to the network status, *e.g.*, the number of storage-nodes, block creation, and proofs generation time, etc.).

Success probability of a malicious storage-node. A malicious storage-node may erase t chunks and still being able to compute a valid proof with a certain probability. Naturally, this probability depends on the number of stored chunks m, the number of deleted chunks t, and the number of chunks challenged d. Ateniese et al. [5] shows that a malicious storage-node fails to compute a valid proof with a probability p that is:

$$1 - \left(\frac{m-t}{m}\right)^d \leq p \leq 1 - \left(\frac{m-d+1-t}{m-d+1}\right)^d.$$

If we set t to be a percentage of m, a malicious storage-node fails (with high probability) if it is challenged on a constant number of chunks d. In particular, if $t = 1\%$ of m, then challenging $d = 460$ and $d = 300$ chunks permits

5. For $d = 1000$, $n/2$ chunks are proven in approximately 50 timestamps (*e.g.*, 8 hours if each timestamp is 10 minutes long).

to achieve p of at least 99% and 95%. We refer the reader to [5] for more details.

Communication complexity. The communication complexity that Audita adds to the underlying blockchain protocol depends on the PDP scheme used (the output of the Prove algorithm consists in a signature σ of size λ, and a PDP proof π'). The publicly verifiable scheme in [5] produces proofs of size $O(\log(d)+|f|+v+|N|)$ where $|f|$ is the chunk bit length, $|N|$ is the size of the RSA modulo, and v is the output length of a PRF. It is reasonable to assume $|f| \gg \log(d)$, $|f| \gg |N|$, $|f| \gg v$, hence the proof size (and thus the additional communication complexity) is mainly determined by $|f|$. However, the results shown in Figure 5 and Figure 6 depend on the chunk size ($|f| = 16$KB in our experiments). Therefore, a small $|f|$ would make the communication efficient but decrease the storage guarantees of Audita. We observe that it is possible to keep the same guarantees by tuning the parameters d and $|f|$. To be concrete, if we reduce the chunk size from 16KB to 1KB (16 times smaller), it is enough to set d to $16 \cdot d$ to achieve the same performance while lowering the communication complexity of the protocol by a factor of 16.[6]

8. Conclusions

In this work, we presented Audita, a blockchain-based decentralized storage system that redefines the current structure of the widely used cloud storage services. Audita can be built on top of several blockchain systems and uses an augmented network of participants that include storage-nodes and block-creators.

We identified the properties that a decentralized storage system must satisfy, and we provided a detailed comparison between the current state-of-the-art systems and Audita. We formally defined Audita and we evaluated its security guarantees. In addition, we demonstrated the technical feasibility of Audita by implementing a prototype based on the distributed ledger Quorum, and we evaluated its performance.

References

[1] B. Alon and E. Omri, "Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious," in *Theory of Cryptography Conference*. Springer, 2016, pp. 307–335.

[2] Amazon, "Amazon drive," 2011, last visited November 21, 2019. [Online]. Available: https://www.amazon.com/gp/drive/about

[3] F. Armknecht, J.-M. Bohli, G. O. Karame, and W. Li, "Sharding pow-based blockchains via proofs of knowledge." *IACR Cryptology ePrint Archive*, vol. 2017, p. 1067, 2017.

[4] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, "Remote data checking using provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 12, 2011.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, "Provable data possession at untrusted stores," in *Proceedings of the 14th ACM conference on Computer and communications security*. Acm, 2007, pp. 598–609.

[6] G. Ateniese, M. T. Goodrich, V. Lekakis, C. Papamanthou, E. Paraskevas, and R. Tamassia, "Accountable storage," in *International Conference on Applied Cryptography and Network Security*. Springer, 2017, pp. 623–644.

6. Note that the proof size scales logarithmically in d.

[7] G. Ateniese, S. Kamara, and J. Katz, "Proofs of storage from homomorphic identification protocols," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2009, pp. 319–333.

[8] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," in *Sequences Ii*. Springer, 1993, pp. 329–334.

[9] K. D. Bowers, M. Van Dijk, A. Juels, A. Oprea, and R. L. Rivest, "How to tell if your cloud files are vulnerable to drive crashes," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 501–514.

[10] V. Buterin, "Ethereum white paper," https://github.com/ethereum/wiki/wiki/White-Paper/f18902f4e7fb21dc92b37e8a0963eec4b3f4793a," 2017.

[11] B. Chase and E. MacBrough, "Analysis of the xrp ledger consensus protocol," *arXiv preprint arXiv:1802.07242*, 2018.

[12] Docker, "Bind mounts," 2013, last visited November 21, 2019. [Online]. Available: https://docs.docker.com/storage/bind-mounts/

[13] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 967–984.

[14] M. Gondree, "libpdp, a library for proofs of data possession," 2013, last visited November 21, 2019. [Online]. Available: https://github.com/gondree/libpdp

[15] Google, "Google drive," 2012, last visited November 21, 2019. [Online]. Available: https://www.google.com/drive/

[16] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," in *Conference on the Theory and Application of Cryptography*. Springer, 1990, pp. 437–455.

[17] S. Inc., "Quorum maker," 2017, last visited November 21, 2019. [Online]. Available: https://github.com/synechron-finlabs/quorum-maker

[18] H. Kopp, C. Bösch, and F. Kargl, "Koppercoin - A distributed file storage with financial incentives," in *Information Security Practice and Experience - 12th International Conference, ISPEC 2016, Zhangjiajie, China, November 16-18, 2016, Proceedings*, 2016, pp. 79–93. [Online]. Available: https://doi.org/10.1007/978-3-319-49151-6_6

[19] P. Labs, "Filecoin: A Decentralized Storage Network," 2017, [Whitepaper; 14-August-2017]. [Online]. Available: https://filecoin.io/filecoin.pdf

[20] 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, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 475–490. [Online]. Available: https://doi.org/10.1109/SP.2014.37

[21] J. Morgan, "Quorum," 2016, last visited November 21, 2019. [Online]. Available: https://github.com/jpmorganchase/quorum

[22] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[23] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX Annual Technical Conference*, 2014.

[24] S. Ruj, M. S. Rahman, A. Basu, and S. Kiyomoto, "Blockstore: A secure decentralized storage framework on blockchain," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2018, pp. 1096–1103.

[25] B. Sengupta, S. Bag, S. Ruj, and K. Sakurai, "Retricoin: Bitcoin based on compact proofs of retrievability," in *Proceedings of the 17th International Conference on Distributed Computing and Networking, Singapore, January 4-7, 2016*, 2016, pp. 14:1–14:10. [Online]. Available: http://doi.acm.org/10.1145/2833312.2833317

[26] H. Shacham and B. Waters, "Compact proofs of retrievability," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2008, pp. 90–107.

[27] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, vol. 11, no. 1, pp. 37–57, 1985.

[28] D. Vorick and L. Champine, "Sia: Simple decentralized storage," *White paper available at https://sia.tech/sia.pdf*, 2014.

[29] Wikileaks.org, "Amazon atlas," October 2018, last visited November 21, 2019. [Online]. Available: https://wikileaks.org/amazon-atlas/

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

Appendix A. Proof of theorem 1

Let t, $\mathcal{SN}^*$, $F = \{f_1, \ldots, f_n\}$ be a timestamp in which the blockchain is extended, the set of k elected storage-nodes for that timestamp, and an arbitrary file, respectively. By contradiction, assume that for every set of ℓ storage-nodes $\{\mathsf{sn}_i^*\}_{i\in[\ell]} \subseteq \mathcal{SN}^*$ there exists at least one storage-node $\widetilde{\mathsf{sn}_i} \in \{\mathsf{sn}_i^*\}_{i\in[\ell]}$ that generates a valid proof of possession π with a probability non-negligibly close δ to the probability that the user can extract $\{f_j\}_{j\in\mathcal{I}_{\mathsf{chal}_i^*}}$ by means of a knowledge extractor E. Then, we build an attacker A for $\mathbf{G}^{\mathsf{pdp}}_{\mathsf{PDP},F,\mathsf{A}}$. A has three random oracles $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^*$ under its control, and it acts as both the user and dealer for Audita. We additionally assume that A can see the messages exchanged between the block-creators and storage-nodes.[7] Through its entire execution, A answers to the queries for the oracles $\mathsf{H}_1, \mathsf{H}_2$ in the following way:

H_1**:** Upon input $x = (\widehat{\mathsf{pk}_{\mathsf{sn}}})$, if $(\widehat{\mathsf{pk}_{\mathsf{sn}}}, y) \in \mathcal{L}_1$ then return y. Otherwise, select a random $y' \leftarrow\!\!\$\ \{0,1\}^*$, add the tuple $(\widehat{\mathsf{pk}_{\mathsf{sn}}}, y')$ to $\mathcal{L}_1$, and return y'.

H_2**:** Upon input $x = (\widehat{\mathsf{idstr}})$, if $(\widehat{\mathsf{idstr}}, y) \in \mathcal{L}_2$ then return y. Otherwise, select a random $y' \leftarrow\!\!\$\ \{0,1\}^*$, add the tuple $(\widehat{\mathsf{idstr}}, y')$ to $\mathcal{L}_2$, and return y'.

We build A in the following way:

1) Sample a random storage-node $\mathsf{sn}_j \in \mathcal{SN}$ and compute the set of chunks indexes $\mathcal{X}_{\mathsf{sn}_j}$ that sn_j is entitled to store (as described in $\mathsf{GetChunks}$, Construction 1).
2) Start the game $\mathbf{G}^{\mathsf{pdp}}_{\mathsf{PDP},F,\mathsf{A}}(1^\lambda, d, \mathcal{X}_{\mathsf{sn}_j})$.
3) Receive pk^* from the challenger.
4) For each $f_i \in F$, send f_i to the oracle Tag, and receive the tag τ_i.
5) Set $\mathsf{pk}^* = \mathsf{pk}_{\hat{F}}$ and $\hat{F} = \{\hat{f}_i = (f_i, \tau_i)\}_{i\in[n]}$.
6) Eventually, receive the randomness r^* from the challenger. At this point, A programs the random oracle H_3 in the following way:

   H_3**:** Without loss of generality, assume that there are $q = \mathsf{poly}(\lambda)$ queries $x_i = (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\_)$ (*i.e.*, queries with prefix $\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}}$). We denote with $\mathcal{Q} = \{x_1, \ldots, x_q\}$ such queries. A flips a bit $b \leftarrow\!\!\$\ \{0,1\}$: If $b = 0$, it selects a random index $l \in [q]$, sets $\mathsf{H}_3(x_l) = r^*$, and answers randomly to all other queries.[8] Otherwise (i.e, $b = 1$), it answers with $\mathsf{H}_3(x_i) = y_i$ where $y_i \leftarrow\!\!\$\ \{0,1\}^*$ to each query x_i.
7) The adversary starts the Audita protocol by publishing $\mathsf{pk}_{\hat{F}}$ and sending to each storage-node $\mathsf{sn} \in \mathcal{SN}$, with public key $\mathsf{pk}_{\mathsf{sn}}$, the file portion $\hat{F}_{\mathsf{sn}} = \mathsf{GetChunks}(\mathsf{pk}_{\hat{F}}, \hat{F}, \mathsf{pk}_{\mathsf{sn}})$.
8) Wait until round t. Eventually, a leader bc^* and a set of k storage-nodes $\mathcal{SN}^*$ will be elected. The leader sends its identification string idstr to each $\mathsf{sn}_i^* \in \mathcal{SN}^*$. A finishes to program the random oracle H_3 in the following way: If $b = 1$, it sets $\mathsf{H}_3(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) = r^*$; Otherwise, it sets $\mathsf{H}_3(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) = y$ where $y \leftarrow\!\!\$\ \{0,1\}^*$.
9) Eventually, each storage-node $\mathsf{sn}_i^* \in \mathcal{SN}^*$ will output a proof of possession $\pi_i = (\pi_i', \sigma_i)$. A checks the following: if $[(b = 0 \wedge x_l \neq (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr})) \vee (b = 1 \wedge (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q})]$; If yes, A aborts. Otherwise, it samples a random proof of possession $(\widehat{\pi}', \widehat{\sigma}) = \widehat{\pi}$ generated by a storage-node $\widehat{\mathsf{sn}} \in \mathcal{SN}^*$ and sends $\widehat{\pi}'$ to the challenger.

We start by analyzing the probability of abortion. Let E_{abort}, E_1, E_2 be the event that A aborts, $(b = 0 \wedge x_l \neq (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}))$, and $(b = 1 \wedge (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q})$, respectively. We can write $\Pr[\neg E_{\mathsf{abort}}] = 1 - \Pr[E_{\mathsf{abort}}]$ and $\Pr[E_{\mathsf{abort}}] = \Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_2]$. Assuming that $\Pr[(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q}] = p_1$ we have:

$$
\begin{aligned}
&\Pr[E_1] = \Pr[b = 0] \cdot \Pr[x_l \neq (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr})] = \\
&= \frac{1}{2}\Big(\Pr[x_l \neq (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) | (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q}] \\
&\quad \cdot \Pr[(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q}] \\
&\quad + \Pr[x_l \neq (\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) | \mathsf{idstr} \notin \mathcal{Q}] \\
&\quad \cdot \Pr[\mathsf{idstr} \notin \mathcal{Q}] \Big) \\
&= \frac{1}{2}[(1 - 1/q) \cdot p_1 + (1 - p_1)] = \frac{1}{2}(1 - p_1/q)
\end{aligned}
$$

and $\Pr[E_2] = \Pr[b = 1] \cdot \Pr[(\mathsf{pk}_{\hat{F}}||\mathsf{pk}_{\mathsf{sn}_j}||\mathsf{idstr}) \in \mathcal{Q}] = \frac{p_1}{2}$. This allows to conclude that the probability of abortion is:

$$
\begin{aligned}
\Pr[E_{\mathsf{abort}}] &\leq \Pr[E_1] + \Pr[E_2] = \frac{1}{2}(1 - p_1/q) + \frac{p_1}{2} \\
&= \frac{1}{2}(1 - \frac{p_1}{q} + p_1) = \frac{1}{2}(1 + p_1(1 - 1/q)) \\
&\leq \frac{1}{2}(1 + 1 \cdot (1 - 1/q)) = 1 - \frac{1}{2q}
\end{aligned}
$$

Thus, the probability of that A does not abort is $\Pr[\neg E_{\mathsf{abort}}] \geq \frac{1}{2q}$. Moreover, by contradiction we know that for every set of of ℓ storage-nodes $\{\mathsf{sn}_i^*\}_{i\in[\ell]} \subseteq \mathcal{SN}^*$ there exists at least one storage-node $\widetilde{\mathsf{sn}_i} \in \{\mathsf{sn}_i^*\}_{i\in[\ell]}$ that generates a valid proof of possession π with a probability non-negligibly close δ to the probability that the user can extract the challenged chunks by means of a knowledge extractor E. Hence, A wins the PDP game if and only if $\mathsf{sn}_j = \widetilde{\mathsf{sn}_i} = \widehat{\mathsf{sn}}$ and $\mathsf{Ver}_{\mathsf{PDP}}(\mathsf{pk}^*, \mathsf{chal}, \widehat{\pi}') = 1$ where $\mathsf{chal} = \mathsf{GenChal}_{\mathsf{PDP}}(d, \mathcal{X}_{\mathsf{sn}_j}; r^*)$. We calculate $\Pr[\mathsf{sn}_j = \widetilde{\mathsf{sn}_i} = \widehat{\mathsf{sn}}] = \Pr[\widehat{\mathsf{sn}} = \widetilde{\mathsf{sn}_i}] \cdot \Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j]$ in the following

7. For example, it can join the network without interfering the protocol.

8. A builds a list $\mathcal{L}_3$ to answer consistently to the H_3 queries.

way:

$$\begin{aligned}
&\Pr[\widehat{\mathsf{sn}} = \widetilde{\mathsf{sn}_i}] \cdot \Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j] \geq \frac{1}{k} \Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j] \\
&= \frac{1}{k} \Bigg(\Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j | \mathsf{sn}_j \in \mathcal{SN}^*] \cdot \Pr[\mathsf{sn}_j \in \mathcal{SN}^*] \\
&\qquad + \Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j | \mathsf{sn}_j \notin \mathcal{SN}^*] \cdot \Pr[\mathsf{sn}_j \notin \mathcal{SN}^*] \Bigg) \\
&= \frac{1}{k} \Bigg(\Pr[\widehat{\mathsf{sn}} = \mathsf{sn}_j | \mathsf{sn}_j \in \mathcal{SN}^*] \cdot (1 - \Pr[\mathsf{sn}_j \notin \mathcal{SN}^*] \Bigg) \\
&= \frac{1}{k} \Bigg(\frac{1}{|\mathcal{SN}^*|} \cdot \Bigg(1 - \frac{|\mathcal{SN}| - 1}{|\mathcal{SN}|} \cdot \ldots \cdot \frac{|\mathcal{SN}| - k}{|\mathcal{SN}| - k + 1} \Bigg) \Bigg) \\
&= \frac{1}{k^2} \Bigg(1 - \frac{|\mathcal{SN}| - k}{|\mathcal{SN}|} \Bigg) = \frac{1}{k^2} \cdot \frac{k}{|\mathcal{SN}|} = \frac{1}{k \cdot |\mathcal{SN}|},
\end{aligned}$$

where we used the fact that the set of k elected storage-nodes $\mathcal{SN}^*$ is randomly computed by hashing the unpredictable identification string idstr.

Conditioned on $\neg E_{\mathsf{abort}} \wedge (\mathsf{sn}_j = \widetilde{\mathsf{sn}_i} = \widehat{\mathsf{sn}})$ and since at timestamp t the blockchain is extended, we have $\mathsf{Ver}_{\mathsf{PDP}}(\mathsf{pk}^*, \mathsf{chal}_j^*, \pi'_j) = 1$ where the challenge chal_j^* is computed in the following way:

$$\begin{aligned}
\mathsf{chal}_j^* &= \mathsf{GenChal}_{\mathsf{PDP}}(d, ; \mathsf{H}_3(\mathsf{pk}_{\hat{F}} || \mathsf{pk}_{\mathsf{sn}_j} || \mathsf{idstr})) \\
&= \mathsf{GenChal}_{\mathsf{PDP}}(d, ; \mathsf{H}_3(\mathsf{pk}^* || \mathsf{pk}_{\mathsf{sn}_j} || \mathsf{idstr})) \\
&= \mathsf{GenChal}_{\mathsf{PDP}}(d, \mathcal{X}_{\mathsf{sn}_j}; r^*).
\end{aligned}$$

Hence, $\pi'_j = \widehat{\pi'}$ is a valid proof of possession for the game $\mathbf{G}^{\mathsf{pdp}}_{\mathsf{PDP},F,\mathsf{A}}(1^\lambda, d, \mathcal{X}_{\mathsf{sn}_j})$ with probability greater than $\delta \cdot \frac{1}{2q} \cdot \frac{1}{k \cdot |\mathcal{SN}|}$ where $|\mathcal{SN}|$ and k are positive constants.[9] This concludes the proof.

9. The total number of storage-nodes $|\mathcal{SN}|$ and k are independent from the security parameter λ.