This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

An Incentive-Compatible Mechanism for Decentralized Storage Network

Iman Vakilinia, *Senior Member, IEEE,* Weihong Wang, and Jiajun Xin,

Abstract—The dominance of a few big companies in the storage market arising various concerns including single point of failure, privacy violation, and oligopoly. To eliminate the dependency on such a centralized storage architecture, several Decentralized Storage Network (DSN) schemes such as Filecoin, Sia, and Storj have been introduced. DSNs leverage blockchain technology to create a storage platform such that the micro storage providers can also participate in the storage market. To verify the accurate data storage by the storage providers during a storage contract, DSNs apply a Proof of Storage (PoS) scheme to continuously inspect the storage service. However, continuous verification of the storage provider imposes an extra cost to the network and therefore end-users. Moreover, DSN's PoS verification is vulnerable to a service denying attack in which the storage provider submits valid PoS to the network while denying the service to the client.

Considering the benefits and existing challenges of DSNs, this paper introduces a novel incentive-compatible DSN scheme. In this scheme, the PoS is conducted only if the client submits a challenge request. We model the storage service as a non-cooperative repeated dynamic game and set the players' payoffs such that the storage provider's dominant strategy is to honestly follow the storage contract. Our proposed mechanism leverages the smart-contract and oracle network to govern the storage agreement between the client and storage provider efficiently. Furthermore, our scheme is independent of a specific blockchain platform but can be plugged into any blockchain platform with smart-contract execution capability. As a proof of concept, we have implemented our scheme using solidity language and chainlink oracle network. The performance analysis demonstrates the applicability of our scheme.

The outcome of this paper is a new incentive-compatible mechanism designed carefully for the blockchain-based DSN. The proposed mechanism utilizes different tools including game-theory, smart-contract, oracle network, and Merkle tree to improve the security and performance of storage verification in DSN.

Index Terms—Decentralized Storage Network, Blockchain, Smart Contract, Mechanism Design

1 Introduction

NOWADAYS giant companies dominate the data storage market. The centralized architecture of such storage providers arises a number of concerns. First, data centers are more vulnerable to the single point of failure causing the data breach, data outage, and facilitating censorship. Second, such companies misuse clients' personal data to earn more profit. Third, prices and rules are dictated by a few big players causing oligopoly. This is due to the lack of competitiveness and the small number of service providers [1].

The Decentralized Storage Network (DSN) has offered a storage platform where micro storage providers can also participate in the storage market. DSNs leverage blockchain technology to facilitate the management of the storage service. Blockchain applies the distributed ledger to store transaction histories, and the information is stored across a network of computers instead of on a single server. Utilizing the smart contract, various incentivization mechanisms can be developed on top of the blockchain to automatically moves digital assets following arbitrary pre-specified rules.

DSNs provide an algorithmic storage market to clients and storage providers. The client can outsource the data storage by making a payment to the network. On the other hand, the storage providers share their storage resources with the network in return for a premium. To satisfy data confidentiality, clients' data is encrypted end-to-end at the client side and storage providers do not have access to the decryption keys. This can offer enhanced security and privacy by eliminating the central entity that controls the data. Moreover, micro storage providers can rent out their excessive storage resources, improve the network throughput and reduce the maintenance cost of data centers. As a result, the storage service can be delivered cheaper with more players and options.

One of the main challenges in the decentralized storage network is to verify the correct storage of data by the storage provider. To this end, a DSN platform should be equipped with a *Proof of Storage* (PoS) scheme to monitor the honest behavior of the storage provider for storing the outsourced data intact [2]. Currently, DSNs perform PoS periodically during the storage contract to ensure the accurate storage of the data by the storage provider [3], [4], [5]. However, such a continuous verification is costly and it is vulnerable to a malicious storage provider that submits PoS to the DSN's nodes while refusing service to the client.

To improve the DSN's performance and protect the client from service denying attack, in this paper, we present and analyze a novel decentralized storage scheme in which it is not required to verify the storage service constantly but only once challenged. To this end, we design a new *Incentive-Compatible* mechanism such that players achieve their best outcomes by choosing their actions truthfully.

- *I.Vakilinia is with the School of Computing, University of North Florida, Jacksonville, FL, 32224.*
  E-mail: i.vakilinia@unf.edu
- *W.Wang and J.Xin are with The Hong Kong University of Science and Technology.*

More specifically, we design a non-cooperative repeated dynamic game such that the storage provider's strategy for honestly sharing the stored data is the only subgame-perfect equilibrium. We utilize the smart contract and oracle network to enforce the rules of our proposed storage game. Eliminating the continuous storage verification improves the performance of DSN significantly. On the other hand, our incentive design supports clients from the service denying attack as we explain further in next sections.

To achieve these goals, first we model a storage contract as a non-cooperative repeated dynamic game. Then, we set the players' payoffs such that the dominant strategy of the storage provider is to provide the storage service honestly. Finally, we implement our proposed scheme using a smart-contract and oracle network. To the best of our knowledge, this work is the first to investigate an incentive-compatible challenge-based decentralized storage mechanism utilizing the smart-contract and oracle network.

The main contribution of this work are the three parts, as described below:

- We highlight the inefficiency of current DSN platforms in their continuous storage verification schemes. We also highlight a service denying vulnerability in the current DSNs such that the storage provider submits PoS to DSN while refusing service to the client.
- We design an incentive-compatible mechanism for DSN in which the storage provider's dominant strategy is to truthfully provide the storage service while the storage client's best response strategy is to not submit a storage verification challenge to the network. Such a mechanism improves performance by relaxing the requirement of having continuous storage verification in DSN. Moreover, our design protects DSN from service denying vulnerability.
- We develop a suitable blockchain-based protocol utilizing smart-contract, oracle network, and Merkle-tree-based proof of storage to satisfy the requirements of our proposed DSN. We publicly share our implementation as a proof of concept.

The rest of the paper is organized as follows. The next section reviews major works in the field of the decentralized storage network. In section 3, we overview the decentralized storage network's components. Details of our proposed mechanisms are described in section 4. The experiment results have been discussed in section 5. Finally, we conclude our paper in section 6.

2 Related Work

2.1 Decentralized Storage Network

Filecoin [3], sia [4], storj [5], and swarm [6] are the most well-known platforms utilizing the blockchain technology to implement the DSN. Such platforms leverage the blockchain asset management capabilities to enforce incentive models for clients and storage providers.

Filecoin [3] runs on a blockchain with a native protocol token (also called "Filecoin") which miners earn by providing storage to clients. Clients spend Filecoin hiring miners to store or distribute data. Filecoin miners compete to mine blocks with sizable rewards, but Filecoin mining power is proportional to active storage, which directly provides a useful service to clients. To earn Filecoin, storage providers must prove they are storing the data properly. The Filecoin network verifies that data is stored securely through cryptographic proofs. Storage providers submit their storage proofs in new blocks to the network and validate new blocks sent from the network. Filecoin applies the Proof-of-Spacetime, where a verifier can check if a prover is storing the outsourced data for a range of time. Filecoin works as an incentive layer on top of the Interplanetary File System (IPFS). IPFS [7] is a p2p storage network. Content is accessible through peers located anywhere in the world. These nodes relay information, store it, or do both. IPFS uses content addressing rather than location based addressing to find data. A content identifier, or CID, is a label used to point to material in IPFS. It doesn't indicate where the content is stored, but it forms a kind of address based on the content itself.

Storj [5] is another well-known DSN. Storj utilizes a service called satellite to manage the decentralized storage system. Satellites are responsible for verifying storage, data repair service, receiving and distributing payments, managing storage nodes, account management and authorization system, and storing storage metadata. Storj extends the probabilistic nature of common per-file proofs-of-retrievability to range across all possible files stored by a specific node. Figueiredo *et al*. [1] have investigated the security of the Storj network and explored a DoS vulnerability within Storj's dev./test environment which was experimentally evaluated to be highly feasible. The attack results in the inability of developers to access their test data and storage providers missing out on their payments. However, the authors pointed out that Storj's production system is not vulnerable to such an attack as long as multiple satellites running on load-balanced clusters of servers.

Sia [4] is also a famous DSN. Sia runs its own blockchain. Sia's blockchain stores the file contract. This contract includes the terms of the storage agreement such as pricing and uptime commitment. Sia divides files into 30 segments and uploads each segment in different nodes. This distribution assures that no one host represents a single point of failure and reinforces overall network uptime and redundancy.

Swarm [6] is a distributed storage platform and content distribution service. The primary objective of Swarm is to provide a decentralized and redundant store of Ethereum's public record, in particular, to store and distribute decentralized applications' code and data as well as blockchain data. From an economic point of view, it allows participants to efficiently pool their storage and bandwidth resources in order to provide these services to all participants. The goal is a peer-to-peer serverless hosting, storage, and serving solution that is DDoS-resistant, has zero downtime, is fault-tolerant and censorship-resistant as well as self-sustaining due to a built-in incentive system.

Table 1 summarizes the differences of storage verification in DSN frameworks compared with our proposed scheme.

Our proposed scheme has the following benefits compared with the existing DSNs:

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

TABLE 1: Summary of Storage Verification Differences in Decentralized Storage Network Frameworks

Decentralized Storage Network	Proof of Storage Scheme	Storage Verification Schedule	Vulnerable to Service Denying Attack
Filecoin [3]	Proof of Spacetime	Continuous	Yes
Sia [4]	Based on Merkle Tree	Continuous	Yes
Storj [5]	Satelites perform auditing using probabilistic erasure coding methods	Continuous	Yes
This work	Using Oracle Network and based on Merkle Tree	Only when challenged by the client	No

- Current DSNs require continuous proof from the storage provider to ensure that the data is stored accurately. This process is costly and causes a waste of energy. However, in our scheme, the storage provider does not need to continuously prove the correct storage but only when the client challenges the storage provider.
- Current DSNs require adding third-party services or a new blockchain platform to intervene in the examination of honest behavior of client and storage providers. However, our proposed mechanism does not require a new service but it is pluggable on any blockchain with smart-contract execution capability (e.g., Ethereum).
- The proof of storage in the current DSNs requires storage providers to proof the storage to DSN's network nodes. This can cause a service denying attack such that a dishonest storage provider only provides proof to the network nodes while rejecting service to the client. This threat is not credible in our scheme as we discuss later on.
- Current DSNs are unable to manage the number of data requests from clients. As a result, a dynamic pricing model such as pay-as-you-go cannot be implemented. We discuss how our proposed model can manage the number of requests to support diverse storage services based on the volume of retrieval requests.

2.2 Proof of Storage

One of the main challenges for having a robust DSN is to audit that the storage provider is honestly storing data intact. To this end, a Proof of Storage (PoS) scheme is used. A similar notion of PoS is Proof of Data Possession (PDP). If the storage provider fails to provide the proof, then it will be penalized by the DSN.

PoS schemes have been studied widely in the literature for centralized setting [8], [9], [10], [11]. However, such schemes are not necessarily applicable in the DSNs as in the decentralized setting the verification must be cheap, and the proof and public parameters must be succinct.

Filecoin has introduced a new proof scheme, called Proof of Spacetime (PoSt), where a verifier can check if a prover has indeed stored the outsourced data they committed to over space (i.e., storage) and over a period of time. In PoSt, the prover generates sequential PoS and recursively composes the executions to generate a short proof. Filecoin's PoSt applies zk-SNARKs [12] to generate succinct proofs which are short and easy to verify [3].

Storj [5] introduces the satellite component as a third party to audit the storage service. However, using such a third party service weakens the decentralized architecture. Moreover, storj utilizes the reputation based system for storage providers based on the history of their service.

Sia [4] uses a Merkle tree based auditing scheme such that the host is required to demonstrate the possession of a random segment. A storage provider must present a certain number of proofs to the network within the time frames specified in the file contract to get fully paid.

Recently, several research studies have investigated new methods to improve the proof of storage for DSNs [2], [13], [14]. Du *et al*. [14] have proposed a new framework for auditing the data in the DSNs using pairing based cryptography and zero-knowledge proof. Yu *et al*. [15] have designed a data-time sampling strategy that randomly checks the integrity of multiple files at each time slot with high checking probability. Furthermore, this research proposes a fair sampling strategy by designing an arbitration algorithm with a verifiable random function.

Besides, Vector Commitments (VCs) can also serve as a PoS solution. VCs can commit to a list of values to a digest, and later provide succinct proof to prove one value is the committed value in some specific location. However, vector commitment has different limitations. RSA based VCs require a trusted setup for the hidden order group [13], [16], [17], [18]. A class group [19] can be used to generate the hidden order group instead of the trusted setup, but it is still not practical due to calculation overheads. Bilinear groups based VCs [16], [20], [21], [22] require at least a linear number of public parameters as common reference strings which limits its adoption in decentralized settings. Lattice based VCs [23] have the pros of post-quantum security, simple setup, and cons of larger communication as well as computation overhead due to the lattice itself.

A relative notion is Proof of Retrievability (PoR) [24], [25], [26], [27], [28], [29]. PoR guarantees that only if a server stores entire files without loss, it can provide a valid proof. While in PoS, the server can still provide a valid PoS proof with some part of the file lost with non-negligible probability. However, the stronger guarantees come with a price. Most PoR schemes require heavy cryptography tools, assumptions, or large overhead.

2.3 Game Theory

Several research studies have investigated the application of game-theory in decentralized networks. Rzadka *et al*. [30] have investigated the problem of maximization of data availability in a decentralized data replication system. They propose a game-theoretic mechanism that reduces the price of anarchy and at the same time provides incentives to agents to be highly available. The resulting game is modeled as an extensive game in which agents change their

replication agreements. They also show that in the unique subgame-perfect equilibrium, agents will form replication agreements with agents of similar availability.

Wang *et al*. [31] have designed an incentive-compatible consensus mechanism for the proof of stake based blockchain to encourage the cache helpers to actively provide service. They have modeled the interaction between the cache helpers and the content providers as a Chinese restaurant game.

Dong *et al*. [32] have utilized game theory to address scenarios where a client outsources a computation task to two clouds. To this end, they have designed two smart contracts (the Prisoner's contract and the Traitor's contract). The contracts guarantee that the two clouds, if they are rational, will behave honestly even though they have the opportunity to collude together and cheat. They proved that for the two clouds, both being honest and not colluding is the unique sequential equilibrium of the game.

Manshaei *et al*. [33] have studied the strategic behavior of rational processors within committees in a shard-based consensus protocol. They analyzed the Nash equilibria in an N-player static game model of the sharding protocol. Further they shown that depending on the reward sharing approach employed, processors can potentially increase their payoff by unilaterally behaving in a defective fashion resulting in a social dilemma. To solve this issue, they proposed an incentive-compatible reward sharing mechanism to promote cooperation among processors.

2.4 Motivation

Inspired by the previous schemes, in this paper we present a novel game-theoretic challenge-based storage contract mechanism for DSNs. To this end, our proposed mechanism allows the client to submit a challenge request indicating that the storage provider has not shared the outsourced data. Once the challenge request is received, smart-contract and oracle network conduct the storage verification. In contrast with previous works, our scheme is designed such that it does not require a continuous verification of storage but it only executes once the client submits a challenge request. The mechanism is designed such that the dominant strategy for the storage provider is to honestly store and share data with the client. On the other hand, the client's dominant strategy is to not submit a challenge request if the storage service has been delivered accurately. Moreover, our scheme prevents the dishonest storage provider to deliver PoS to the DSN while refusing the storage service to the client. This design significantly decreases verification overhead costs in the current DSNs. We leverage the oracle network to alleviate the DSN's execution cost. Furthermore, our proposed scheme is independent of an underlying blockchain layer and can be executed on top of every generic blockchain with smart-contract execution capabilities such as Ethereum and Bitcoin. Our scheme leverages the Merkle tree for the PoS as we describe in section 4.

3 Overview

In this section, we review the system architecture of the storage service in a decentralized storage network utilizing blockchain technology.

A DSN provides a platform for a storage provider to offer the storage service to clients. A client aims to purchase the storage service to store and access her data for a specified time period. On the other hand, a storage provider aims to sell his storage service to host the client's data in return for a premium. In a nutshell, a DSN is equipped with two main components of payment settlement and storage verification. In the payment settlement module, the DSN charges the client for the storage service and makes the payment to the storage provider. Moreover, in case the storage provider fails to provide the committed storage service, the DSN penalizes the storage provider and compensates the client accordingly. In the verification module, the DSN verifies that the storage provider is delivering the storage service accurately. To this end, the storage provider should submit proof of storage to the DSN, and DSN verifies the correctness of such proofs.

Once the client and the storage provider agree on a storage service, they enroll in a storage contract. This contract conveys the storage Service Level Agreement (SLA) which specifies the storage service including the duration of the contract, premium, quality of service, and compensation rates.

DSNs leverage public blockchain technology to enforce storage contracts. Blockchain technology has offered an agreeable platform for parties to make payments without a single trusted third party. Blockchains are managed by a peer-to-peer network to manage a digital ledger. Recorded data on a public blockchain is publicly accessible and tamper-resistant. A smart contract is a code in the blockchain that automatically enforces a contract between two parties without any help from a single third party. Therefore, there is no need for an intermediary between contracting entities to enforce the contract. Accordingly, in a blockchain enabled DSN scheme, there is not a single party controlling any storage contract.

In a DSN storage service, the client may encrypt her data before submitting it to the storage provider to protect the confidentiality of her data. Moreover, DSNs can provide redundancy, high-availability, and fail-over by storing the data in multiple nodes in the network.

Note that the details of storage techniques that the storage provider is using to store the client's data are out of the scope of this paper. In other words, we assume the storage provider manages his storage resources including redundancy, server location, backup services, network bandwidth, etc to maximize his payoff following the SLA.

The overall scheme of a Blockchain-based DSN is depicted in Figure 1.

3.1 Design Goal

The primary goal of DSN's mechanism design is to ensure that the storage provider stores the client's data and returns it upon the client's request following the SLA. The storage service should be examined, and the client should be compensated in case of a storage failure. The client should pay the storage provider if the storage service has been delivered flawlessly. We aim to improve the current methods of PoS in DSNs by eliminating the requirement of continuous verification of the data storage on the storage provider.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

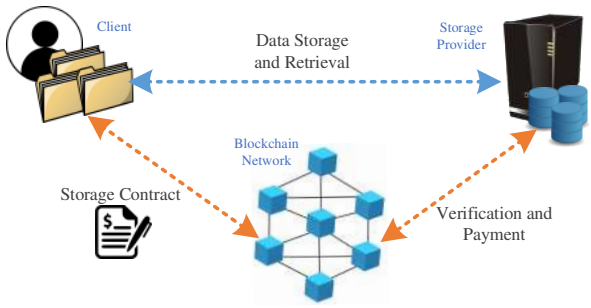

Fig. 1: The overall scheme of a Blockchain-based Decentralized Storage Network

The mechanism should be incentive-compatible such that the players can earn their best outcome by choosing their actions truthfully.

Moreover, we consider the following side features in our design goal:

- **Blockchain platform independent.** Current DSN systems work on their own customized blockchain platforms. This causes an extra overhead cost for the DSN. We aim to propose a compatible storage scheme that is pluggable to available generic blockchain platforms with smart-contract execution compatibility (e.g., Ethereum).
- **Prevent service denying attack.** Currently available DSNs are vulnerable to a service denying attack such that a dishonest storage provider denies providing the expected service to the client while successfully submitting PoS to the DSN network. In this case, the storage provider receives the service fee while the client has not received the expected service. We aim to protect the DSN network from such a fraudulent storage provider.
- **On-chain efficiency.** On-chain storage and computation are costly. Therefore, the proposed scheme should minimize the on-chain storage and computation without compromising security expectations.
- **Counting requests.** Many storage services expect to count the number of requests from the client to dynamically calculate the cost of service. So far, the available DSNs do not provide storage services based on the number of requests. We will discuss how our proposed mechanism can accomplish this task.

4 An Incentive-Compatible Mechanism for the Storage Contract

In this section, we discuss our proposed incentive-compatible mechanism for the blockchain-based decentralized storage network. First, we model and analyze the storage contract as a non-cooperative repeated dynamic game. Then, we discuss the design of the storage contract to satisfy the requirements of our proposed mechanism. To this end, we utilize the smart-contract and oracle network. Finally, we study a Merkle tree based PoS scheme for the verification of storage suitable for our proposed mechanism.

4.1 Storage Contract As a Non-Cooperative Repeated Dynamic Game

The objective of our mechanism design is to place a set of rules for the storage service in DSN to meet the requirements as mentioned earlier. A mechanism can be specified by a game $g : \mathcal{M} \rightarrow \mathcal{X}$ where $\mathcal{M}$ is the set of possible input messages and $\mathcal{X}$ is the set of possible outputs of the mechanism. In the storage system model, players are the storage provider and the storage client. We assume players are rational self-interested such that they aim to maximize their profit. A rational player chooses his strategy to increase his/her utility. The utility of a storage client player is defined based on the premium paying for the storage service, and the cost/benefit of data accessibility after outsourcing the storage to the storage provider. On the other hand, the storage provider's utility is defined based on the premium receiving for the storage service providing to the storage client, the cost of storage service maintenance, and the penalty of loosing the data or lack of service quality. In the following we elaborate such a game to satisfy our goal requirements. Specifically, we complete this storage game with strategies for the players to reach our suitable outcome for the game.

In the design of a mechanism for decentralized storage network, the following questions need to be answered:

- How the mechanism can verify the storage provider's honest behavior of sharing data?
- What is the payment channel for the service?
- How the mechanism can charge the client for the service?
- How the mechanism can penalize the storage provider for loss of data or low quality service?

A naive model is to have a Trusted Third Party (TTP) mediating between the client and storage provider. In this case, the client requests data from TTP, and TTP receives data from the storage provider. TTP can check the integrity of data by storing the hash of data and verifying it whenever receiving data from the storage provider, and then forward it to the client. Upon successful execution of the service, TTP charges the client and pays the storage provider. On the other hand, if the storage provider fails to provide data back to the client, TTP charges the storage provider and pays the client for the data loss or low quality service according to the contract. Although this model ensures the storage service expectations, it is inefficient, expensive, and not scalable due to the requirement of having a TTP as a middleman for every request and response. On the other hand, finding such a TTP is impractical, and such a design resembles a centralized architecture where the TTP acts as a central party and can be potentially bribed.

To solve this problem, DSNs rely on a blockchain platform to act as a TTP to remove a single central entity for managing the system. Moreover, DSNs minimize TTP intervention in the data recovery process such that the client retrieves data from the storage provider directly. However, DSN continuously verifies the proof of storage from the storage provider to ensure the storage provider is storing the data truthfully. DSNs utilize blockchain asset management features to deliver the fees among players. Although this

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

method notably improves the naive solution mentioned above, there are two main issues that remain:

- First, the continuous verification of storage is costly for the network.
- Second, a dishonest storage provider can successfully submit the proof of storage to DSNs while refusing the service to the client.

To solve these issues, in our model, DSN does not continuously verify the storage service but whenever the client submits a challenge request. In our model, the client directly requests data from the storage provider, and the storage provider sends back data directly to the client. However, as a dishonest storage provider might refuse to provide data back or send back incorrect data, the mechanism is equipped with a challenging option. In this case, the client can send a challenge request to the TTP, and TTP verifies the data that the storage provider returns back. Therefore, in our mechanism, the interaction between the client and the storage provider can be modeled as a non-cooperative repeated dynamic game. Once the storage contract starts, in the first stage of the game, the storage provider can choose between *Sharing* data or *Not Sharing* data strategies. Here, sharing means that the storage provider honestly follows the storage contract and shares the client's data upon the client's data request. On the other hand, not sharing, indicates that the storage provider refuses the service to return data upon the client's request.

Afterward, the client can choose *Challenging* or *Not Challenging* the storage provider. Challenging means that the client submits a challenge request to DSN indicating that the storage provider has not shared data. Not challenging indicates that the client does not submit a challenge request. Upon receiving the challenge request, DSN performs the storage verification.

Once the storage provider is challenged, then his strategy set is to *Proof* of storage or *Not Proof* of storage. Proof means that the storage provider submits the proof of storage to DSN. Not Proof means that the storage provider does not submit the proof of storage or fails to submit the accurate proof of storage to DSN. This game is depicted in Figure 2.

The goal of our mechanism design is to ensure that the *subgame-perfect-equilibrium* of this non-cooperative repeated dynamic game is {*Share, No Challenge*}. In other words, we aim to design a mechanism in which storing and sharing data is the storage provider's dominant strategy, and the client's dominant strategy is to not submit a challenge request. Therefore, the proposed mechanism should be *incentive-compatible* such that players can achieve their best outcome only by acting based on their true preferences. To achieve this goal, first, we investigate the players' payoffs in the leaf nodes, and then we set payoffs such that the *Sharing* and *No Challenge* strategies are the dominant strategies for the storage provider and the client, respectively.

As can be seen in Figure 2, there are six possible outcomes for our storage contract game. The terminal nodes demonstrate the players' payoffs. Let P_C and P_S represent the client's strategy and storage provider's strategy, respectively. We can represent the client's and storage provider's utility functions in our proposed dynamic game of storage contract as follows:

$$U_C(P_C, P_S) \rightarrow \{C_1, C_2, C_3, C_4, C_5, C_6\} \tag{1}$$

$$U_S(P_C, P_S) \rightarrow \{S_1, S_2, S_3, S_4, S_5, S_6\} \tag{2}$$

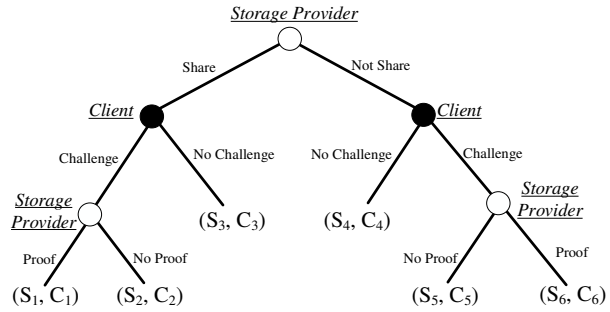

Fig. 2: Dynamic Game of the storage contract

At the first stage of the game, the storage provider's action set is {*Share*, *No Share*}. Note that although the client can be the first player to submit a *Challenge* request even before a data request, we will see that such an action is an *non-credible threat*, and therefore for simplicity and without loss of generality, we will consider the storage provider as the first player in the game. The goal of our mechanism is to ensure that the *Sharing* action is the best strategy for the storage provider. However, Sharing is costly because of the cost of storage, data retrieval, backup, and network bandwidth required for successful sharing.

In the second stage, the client action set is {*Challenge*, *No Challenge*}. By choosing the *Challenge* strategy, the client claims that the storage provider has not shared data. If the client chooses to challenge, then the storage provider has two options as {*Proof*, *Not Proof*}. If the storage provider chooses *Proof*, then it should provide the proof of storage, otherwise, if the storage provider fails to proof, then the storage provider will be penalized according to the contract.

We assume the data has a value for the client. In other words, the client receives compensation in return for the data loss caused by the storage provider. Note that, failure of proof is the worst outcome for the storage provider as the storage provider will be penalized the compensation amount, therefore, we should have:

$$S_{i\in\{1,3,4,6\}} >> S_{j\in\{2,5\}} \tag{3}$$

Following the *backward induction*, the storage provider would choose proof action unless it lost data or cannot provide the service. This is because the cost of *No Proof* is the cost of compensation for the client, and we have:

$$S_1 >> S_2 \quad , \quad S_6 >> S_5 \tag{4}$$

As the mechanism's goal is to ensure that the storage provider chooses the *Sharing* strategy, the cost of *Proof* should be higher than the cost of *Sharing*. Therefore, we should have:

$$S_3 >> S_1 \quad , \quad S_4 >> S_6 \tag{5}$$

On the other hand, as the mechanism's goal is to ensure that the client chooses *No challenge* when the storage

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

provider honestly shared data, then we should have the following:

$$C_3 > C_1 \tag{6}$$

To this end, the mechanism should make the *Challenging* request costly for the client. Let $\mathcal{X}$ represent this cost.

On the other side, as we want the storage provider chooses *Sharing*, the client should choose *Challenge* when the data has not been shared by the storage provider. To achieve this goal, the mechanism should motivate the client for choosing the challenge once the data has not been shared. However, the challenge request is costly as we discussed earlier. To cover the cost of the challenge, our mechanism is designed such that the *Proof* strategy enforces a copy of data to be sent out to the client to improve the payoff of the challenge strategy in case of not sharing. On the other hand, once the data has not been shared, there is a possibility that the storage provider cannot provide the storage service (e.g., due to the data loss). Let $\mathcal{P}$ represent the probability that the storage provider cannot present the storage service. Let $\mathcal{V}$ represent the value of accessing data for the client. Then, the client's expected utility for choosing the challenge can be modeled as:

$$C_c = \mathcal{P}.(C_5) + (1 - \mathcal{P}).(\mathcal{V}) - \mathcal{X} \tag{7}$$

The mechanism should set $C_c > 0$ by making sure that $\mathcal{P}.(C_5) + (1 - \mathcal{P}).(\mathcal{V}) > \mathcal{X}$. On the other hand, we have $C_4 < 0$ (the client's payoff is negative when the storage service is not delivered). Therefore, we have $C_c > C_4$. Using the backward induction, it can be seen that the subgame-perfect-equilibrium of this game is {*Share, No Challenge*} strategy profile.

Example

In this section, we provide an example to clarify the proposed mechanism. For simplicity, we consider the utility of players as a number without the declaration of a specific currency.

Consider that the compensation cost is indicated as "$1,000$" in the storage contract. In other words, if the storage provider is unable to retrieve the client's data, then the storage provider should pay the client "$1,000$". Let the cost of losing data for the client be "500", and the client's benefit of reading data is "5". The cost of requesting the challenge is "1", and the cost of proof of storage for the storage provider is "3". Finally, let the benefit of not sharing data with the client is "2", and the benefit of sharing data be "1" for the storage provider (note that this is the payoff that the storage provider earns by charging the client for providing the correct storage service). Therefore, the dynamic game tree of this game can be depicted as Figure 3.

Using the backward induction, it can be seen that the {Share, No Challenge} is the subgame nash equilibrium of the game.

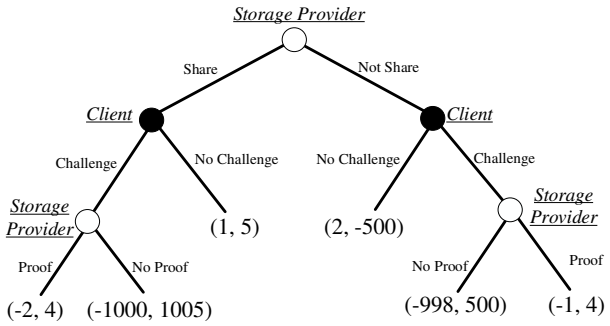

Fig. 3: Dynamic Game of the example storage contract

4.2 Scheme details

Identifying the storage contract requirements and the players' payoffs, now we discuss the design architecture to satisfy the design goal.

We utilize the smart-contract to act as a TTP to manage the agreement between the client and the storage provider. Smart-contract is powered by blockchain technology. The blockchain is managed by a peer-to-peer network to manage a digital ledger. A smart-contract is a code in the blockchain that automatically enforces a contract between two parties without any help from a third party. Therefore, there is no need for an intermediary between contracting entities to enforce the contract. A public blockchain network capable of executing the smart-contract is used as a platform for developing DSN.

In our proposed model, first the client and storage provider reach a storage agreement. This agreement includes the following information: Length of contract, Merkle root of data, premium, delivery time, and compensation. For example, a storage provider makes an agreement with a client to store her 1 TB data for the length of 1 year for the premium of $20, if the storage provider fails to return data, then the storage provider will be penalized by $40, and the time window for delivering data after client's request is 20 minutes.

This agreement will be specified in the smart-contract and deployed on the blockchain. Note that, the Merkle root of data is stored on-chain which will be used for verification of data. Storing the whole data on the blockchain is too costly, therefore Merkle tree is used to minimize the cost of the storage verification process as we discuss later on. Moreover, the smart-contract includes the client's premium as well as the storage provider's collateral asset. Upon the successful storage service, the premium will be automatically transferred to the storage provider. On the other hand, if the storage provider fails to provide the storage service, the client will be automatically compensated through the collateral asset of the storage provider following the contract details.

Note that the blockchain platform is an isolated network, and it cannot pull in or push data out to any external system. This problem is known as the oracle problem [34], [35]. To solve this problem, the oracle network has been presented. Oracle network provides a trusted source for accessing off-chain data to the blockchain. Moreover, it can perform arbitrary programs more efficiently compared with the smart-contract, due to the fact that fewer resources are needed to execute the code. Leveraging the oracle network, the challenge request in our scheme works as follows:

The client submits a challenge request by calling the challenge function of the smart-contract with the specific data segment

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

number to be challenged. Once the challenge request has been received by the smart-contract, the oracle network will submit a challenge to the storage provider. Upon receiving the challenge request from the oracle network, the storage provider should send the challenged data along with the Merkle path to the oracle. In the next step, the oracle network first calculates the Merkle root and compares it with the Merkle root in the storage contract stored on-chain. If they match, then the oracle sends a copy of the data to the client. Otherwise, the oracle sends a fail signal to the smart-contract, and the smart-contract will transfer the compensation fund from the storage provider account to the client account following the pre-specified agreement.

The interaction between different components of our scheme is depicted in Figure 4.

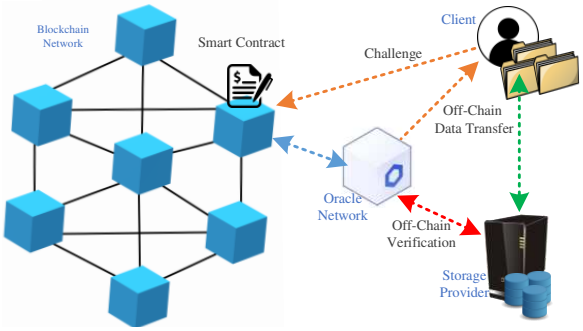

Fig. 4: Interaction of different components in the proposed challenge based DSN storage contract

As can be seen, the data transfer is done off-chain. The only on-chain operation is the challenge request. Note that in this scheme, the storage provider should send the challenged data to the oracle network, and the oracle network forwards the data back to the client. There are two main reasons for this design.

- First, it prevents the service denying attack which we explained earlier. This is due to the fact that if the storage provider refuses service to the client, the client receives a copy of data with the challenge request if the storage provider can pass the proof. Therefore, the storage provider cannot deny service to the client while proofing the storage to DSN.
- Second, when the storage provider has not shared data, the mechanism should provide incentives for the client to submit the challenge. By forwarding the data, we add the value to the client's payoff for choosing the challenge strategy to achieve our desired subgame perfect equilibrium as we discussed in the previous section.

Challenge Level

To improve the efficiency of the scheme, we consider different levels for the challenge request. Let us motivate this feature by an example. Assume the client outsourced a very large dataset of 100 TB to a storage provider. If there is no challenge level option for the system, the storage provider should forward the whole data back to the client upon the challenge request submitted by the client. This can be a resource consuming task. To solve this problem, we consider that the client and storage provider agree to split data into a specific number of segments. In the challenging phase, the client can submit challenges for a set of segments. The price of challenge requests is an increasing function of the size and number of the data segments being challenged. For example, assume the outsourced data of size 100 TB is divided into 100,000 segments of 1 GB size. The client can submit challenges for any number of segments, however, submitting a challenge for larger data is more costly for the client to deter a malicious client to cause a denial of service attack on the storage provider. We explain how our proof of storage scheme can handle this feature in the next section.

4.2.1 Proof of Storage

Proof of Storage (PoS) scheme allows a verifier to check if a storage provider is storing the client's data at the time of the challenge. We follow the definitions from [9] with minor modifications.

Definition 1. *Given security parameter λ, the PoS scheme is a tuple of four probabilistic polynomial-time (PPT) algorithms* (Setup, Challenge, Prove, Verify)*:*

- $(d, h) \leftarrow \text{Setup}(1^\lambda, \mathcal{D}, sz)$. *This algorithm takes as input the security parameter λ, outsourced data $\mathcal{D}$, and segment size sz. The algorithm outputs a digest of data d which is used to verify the proof and the maximum data segments number h. We denote the data segment with number i as $\mathcal{D}_i$ and its complementary data segment as $\hat{\mathcal{D}}_i$ such that $\mathcal{D}_i \cup \hat{\mathcal{D}}_i = D$.*
- $c \leftarrow \text{Challenge}(h)$. *This algorithm takes as input the maximum data segments number h and outputs a challenge number c.*
- $\pi \leftarrow \text{Prove}(\mathcal{D}_c, c)$. *This algorithm takes as input the corresponding data segment $\mathcal{D}_c$ and the challenge number c. The algorithm outputs a proof π to prove the storage of data segments corresponding to the challenged number c.*
- $0/1 \leftarrow \text{Verify}(d, h, c, \pi)$. *This algorithm takes as input the digest d, the Merkle tree height h, the challenge number c, and the proof π. It outputs 1 if the proof π is a valid proof, and it outputs 0 otherwise.*

The first property we require from a PoS scheme is completeness, i.e., a proof output by the scheme algorithms, for a valid statement is verified correctly with all but negligible probability. It can be formulated as below.

Definition 2. *Completeness. A PoS scheme is complete if*

$$\Pr\left[\begin{array}{c}(d,h) \leftarrow \text{Setup}(1^\lambda, \mathcal{D}, sz)\\ c \leftarrow \text{Challenge}(h)\\ \pi \leftarrow \text{Prove}(\mathcal{D}_c, c) :\\ 1 \leftarrow \text{Verify}(d, h, c, \pi)\end{array}\right] > 1 - \text{negl}(\lambda).$$

The second property is challenge soundness which captures that a malicious challenger cannot generate a challenge c' which cannot be answered even if the prover holds the whole file. It can be formulated as below.

Definition 3. *Challenge Soundness. A PoS scheme is challenge sound if*

$$\Pr\left[\begin{array}{c}(d,h) \leftarrow \text{Setup}(1^\lambda, \mathcal{D}, sz)\\ c' \leftarrow \text{Challenge}(h)\\ \pi \leftarrow \text{Prove}(\mathcal{D}, c') :\\ 0 \leftarrow \text{Verify}(d, h, c, \pi)\end{array}\right] < \text{negl}(\lambda).$$

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

The third property is proof soundness which captures that a malicious prover cannot generate a valid proof if it does not hold the challenged file segment even with all the other file segments. We denote the complementary $\hat{\mathcal{D}}_c$ It can be formulated as below.

Definition 4. *Proof Soundness. A PoS scheme is proof sound if*

$$\Pr\left[\begin{array}{c}(d,h)\leftarrow \text{Setup}(1^\lambda,\mathcal{D},sz)\\ c\leftarrow \text{Challenge}(h)\\ \pi\leftarrow \text{Prove}(\hat{\mathcal{D}}_c,c):\\ 1\leftarrow \text{Verify}(d,h,c,\pi)\end{array}\right]<\text{negl}(\lambda).$$

Cryptographic building blocks

Cryptographic secure hash function. We use a cryptographic secure hash function $H \leftarrow \text{Hash}(x)$ that is collision resistant and pre-image hard.

Digital signature. We use the standard EU-CMA secure digital signature function [36] that contains three functions: 1) $(pk, sk) \leftarrow$ KeyGen:(1^λ); 2) $\sigma \leftarrow$ Sign(sk, m); 3) $1/0 \leftarrow$ Verify(pk, m, σ).

Our construction

- Setup$(1^\lambda, \mathcal{D}, sz)$: this algorithm divides data $\mathcal{D}$ into a set of segments $\mathcal{S} = \{s_1, \ldots, s_h\}$ where each segment has the size of sz. This algorithm further builds a textbook Merkle tree based on Hash function Hash. The hash of each segment Hash(s_i) builds the leaves of the Merkle tree. Output:(d, h).
- Challenge(h): this algorithm picks a uniform random number c in $[1, h]$ as the challenge number. Output:(c).
- Prove$(\mathcal{D}_c, c)$: this algorithm outputs the sibling nodes of the challenged node's Merkle path and the preimage s_c (corresponding data segment) to calculate the challenged hash result as the proof π. Output:(π).
- Verify(d, h, c, π): this algorithm checks if the node generated by the segments has a valid Merkle path towards the Merkle root d.

Theorem 1. *Let* Hash *be a collision resistant and pre-image hard hash function. The construction presented above is complete, challenge sound and proof sound.*

Proof. Completeness comes directly from the protocol. Challenge soundness comes from the authenticated maximum challenge number from the setup function. Proof soundness comes from that Hash is a collision resistant and pre-image hard hash function. It is clear to check if there is an Adversary that break the proof soundness, we can use this Adversary to break the collision resistant property of Hash, which contradicts with the assumption. □

For example, assume a Merkle tree as depicted in Figure 5. Here, the challenged number is 2. The authenticated maximum challenge number is 8. Nodes 9 14 are numbered to illustrate the example. The proof π includes the raw data of segment s_2 to generate leaf node 2, and the sibling nodes of the Merkle path for node 2, which includes node numbers 1, 10, and 14. In order to verify the proof π, the verifier first calculates the leaf node based on the data of segment s_2 received from the prover. Then, it calculates node 9 using nodes 1 and 2, calculates node 13 using nodes 9 and 10, and calculates root node using nodes 13 and 14. Finally, it checks if the calculated root is the same as the Merkle root stored on-chain.

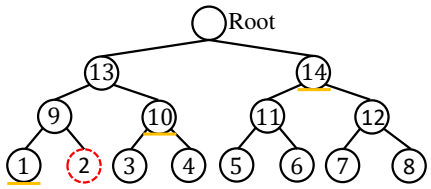

Fig. 5: Merkle tree example.

The process of storing the Merkle root of data on-chain is as follows. First, the client and storage provider generate their public/private key pairs. Let (pk_c, sk_c) and (pk_{sp}, sk_{sp}) to denote their public/private key pairs respectively. Client runs Setup, signs the Merkle root and Merkle Tree height, and passes $(d, h, \text{Sign}(sk_c, d||h))$ to the storage provider. The storage provider runs Setup separately, and if it gets the same result, he then signs (d, h), and sends $(d, h, \text{Sign}(sk_c, d||h), \text{Sign}(sk_{sp}, d||h))$ to the smart-contract. The smart-contract verifies signatures and stores d on-chain for future proof of storage verification.

Discussion

Based on our definition of PoS, several different tentative solutions can be applied. Various cryptographic accumulators [18], [37] and vector commitments [13], [16], [17], [18] are valid solutions. However, we choose the Merkel tree to construct the PoS based on mainly two reasons.

Firstly, in the decentralized settings, we desire no trusted setup and less public parameters which limits the use of RSA and bilinear pairing based solutions.

Secondly, the cost of the proof and verification has been considered in the incentive layer. The major concern of our scheme is the digest size because the digest needs to be stored on-chain. On this point, the Merkle tree is a good solution as we only need to store the Merkle root on-chain which is a hash value.

4.2.2 Counting the number of requests

The number of requests for retrieving data during the storage contract is an important factor in pricing the storage service. This is due to the fact that the number of retrieval requests directly impacts the storage provider's workload. For example, consider that a client demands a service with a maximum of 5 retrieval times in a year. On the other hand, another client demands 1000 retrieval times in a year. Therefore, the storage service should be able to dynamically charge clients based on the number of reading requests. Currently, DSNs do not support the number of requests in the storage service.

A naive approach is to apply request counting. In this case, the client first signs a request for data and sends it to the storage provider. The storage provider then verifies the signature and sends the data back to the client. However, as a malicious storage provider might refuse to send data, the client should send back a signed acknowledgment message upon successful delivery of data. On the other hand, a malicious client refuses to send back the signed acknowledgment message. To solve this issue, one simple approach is to split the data into smaller pieces and

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

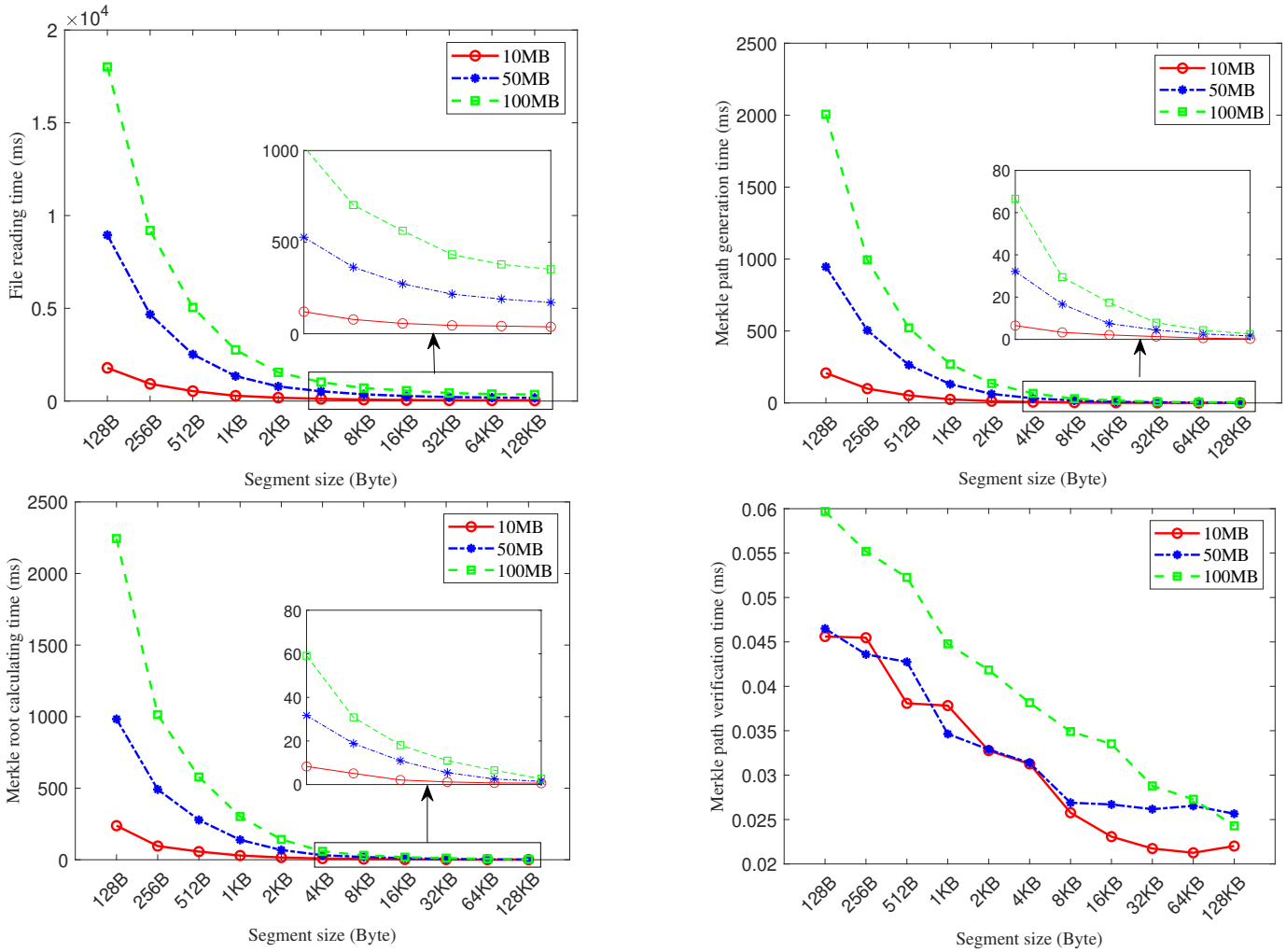

Fig. 6: Computation cost of File reading, Merkle path generation, merkle root calculation, and Merkle path verification for files of 10MB, 50MB, and 100MB with varying segment size

send the next portion of data upon receiving the previous message acknowledgment. Although such an approach is appropriate in the network layer with TCP protocol, in the application layer this approach is too costly as every acknowledgment message should be signed and verified. Moreover, there is no guarantee that the client sends the last message acknowledgment.

To solve the naive approach, in our scheme we follow our proposed non-cooperative repeated dynamic game-theoretic approach for challenging the storage provider. In our model, the client only sends a signed request message. The storage provider has two options, whether send or not to send the data. Then, it is the client's choice to challenge or not challenge. Following our dynamic game tree, the best outcome will be reached through sharing and not challenging strategies.

For cashing out the number of requests, the storage provider only needs to submit the last signed request message. Note that the request message includes a counter indicating the number of requests so far that have been submitted by the client.

5 Implementation

In this section, we first describe the details of our implementation, then discuss the performance evaluation of our proposed mechanism. We use the Solidity (version 0.8.7) programming language to implement our smart contract. In the part of the blockchain solution, we select Kovan, an Ethereum test network for smart contract development. We use Remix IDE to develop, deploy and administer the smart contract. In our implementation, we use Chainlink's oracle [38], which is currently the most famous oracle network and has the majority share of the oracle market [39]. We have shared our implementation in a GitHub repository[1].

Considering gas fees and expensive on-chain calculation costs, we only record the basic information including the premium, duration of the contract, compensation rates, and the Merkle root value of the file on-chain. The storing of the original outsourced data and the verification process are done by the storage provider and the external adapter off-chain, respectively. The contract has two main functions as we described below.

1. https://github.com/podiumdesu/ICM-DSN

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

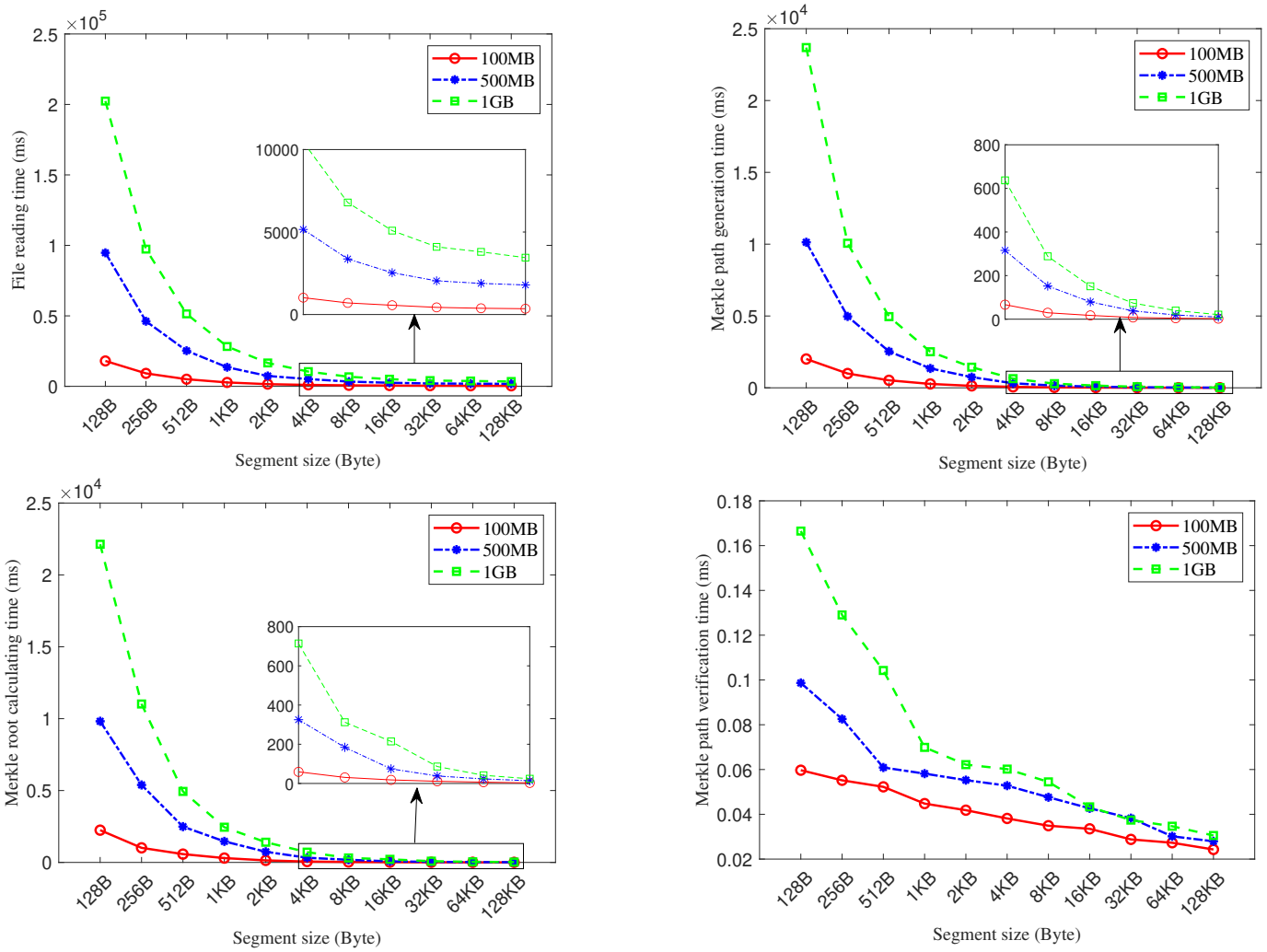

Fig. 7: Computation cost of File reading, Merkle path generation, merkle root calculation, and Merkle path verification for files of 100MB, 500MB, and 1GB with varying segment size

Recording a storage task. After the commitment between the client and the storage provider, the contract will record the basic information signed by both sides on-chain according to the designed data structure.

Resolving a challenge request. Once the client submits an on-chain request challenging a storage provider for one specific file segment, we need to build a connection with the off-chain data. We are going to introduce how to construct an oracle request.

A basic oracle request model includes four parts: Chainlink client, oracle contract, off-chain oracle node, and external adapters.

Chainlink client is a parent contract that enables smart contracts to construct and make a request "sendChainlinkRequestTo" to a known Chainlink oracle. Our smart contract inherits from Chainlink client. A complete request needs an oracle address, the job ID, and the callback function. Through job ID, the oracle knows which tasks to perform; after finishing the tasks, the oracle sends the response to the callback function.

Oracle contract will handle on-chain requests and emit an event containing information about the request. The off-chain oracle node will monitor the event. Once the oracle contract receives the result of the job, it will return the result to the Chainlink Client using the callback function. In our implementation, we use the oracle supporting the "EthBool" adapter, which makes an HTTP GET request, takes the given values, and formats them for the Ethereum blockchain in a boolean value.

Off-chain oracle node runs alongside oracle contracts. It listens to events emitted by the on-chain oracle contract and performs a job with the data emitted. Here, this node will make a GET request to our external adapter and get the result of whether the storage provider passes the challenge or not, then submits the boolean result in a transaction back to the oracle contract.

External adapter. By making it an API, we can customize the off-chain computation the way we want. Here, the external adapter sends a request to the storage provider with the parameters of the file ID and the specific data segment number chosen by the client. The storage provider will send the challenged piece of data along with the calculated Merkle path back to our external adapter. The external adapter processes the response using the Merkle path and

the hash value of that piece of data to calculate the Merkle root and compares it to the on-chain stored value. After that, it submits the challenge result to the oracle. The external adapter is written in JavaScript programming language and runs as an HTTP Server in Node.js (version 16.13.1).

Besides, on the side of the storage provider, we simulate it as an HTTP server in Node.js (version 16.13.1). It provides external APIs for calculating Merkle root, giving access to the original file, and generating the Merkle path. The storage provider divides the file with the segment size chosen by the client. The slices' hash values are the Merkle tree's leaf nodes. Therefore, we can build the Merkle tree, calculate the Merkle root, and receive the Merkle path.

The cost of storage contract deployment, recording a storage task, and submitting a challenge request is depicted in Table 2. Note that once the storage contract is deployed, different storage tasks can be recorded on top of it. Moreover, to decrease the cost of on-chain deployment, a pooling approach can be applied [40] without affecting the integrity of the scheme.

TABLE 2: Gas cost for the storage contract

Operation	Gas units
Contract on-chain Deployment	2,491,606
Recording a Storage Task	202,001
Challenge Request	192,101

5.1 Performance analysis

In the subsequent experiments, we have used a macOS (version 12.0.1) laptop with an Apple M1 Pro CPU and 32 GB of memory for the performance analysis of our scheme. We conducted our evaluation over files with various sizes of 10MB, 50MB, 100MB, 500MB, and 1GB. The computing time is calculated with varying segment sizes for the files. We included various settings for our evaluation. Specifically, we look at comparisons between four dimensions of file reading time, Merkle root calculating time, Merkle path generation time, and Merkle path verification time. Figures 6 demonstrate the computation cost of file reading time, Merkle root calculation, Merkle path generation, and Merkle path verification for files with sizes of 10MB, 50MB, and 100MB. Figures 7 demonstrates the same settings for files with sizes of 100MB, 500MB, and 1GB.

As can be seen, with the increase in segment size, the computation cost for file reading time, Merkle root calculation, and Merkle path generation is decreasing at a decreasing rate. Merkle path verification is fast, and its time is also decreasing with the increase of the segment size.

When the file sizes of the same segment size increase, the time of file reading, Merkle root calculating and Merkle path generation has the same growth. In contrast, the time of Merkle path verification time is almost the same at a very small value.

5.2 Discussion

In this subsection, we discuss the results of our experiments. Our evaluation demonstrates the applicability of our proposed mechanism. Our proposed scheme is implemented in two layers which are the incentive layer and the PoS layer. In the incentive layer, players are motivated to play the storage game truthfully. To this end, we have used the smart contract to enforce the rules of the game to make it incentive-compatible as we discussed earlier.

On the other hand, in the PoS layer, the system verifies the storage service once the client submits a challenge request. This verification module is implemented using the oracle network in the smart contract. It is worth mentioning that the security of our scheme relies on the security of the smart contract and the oracle network. The oracle networks are designed to provide a secure service to the smart contract. The oracle network provides tamper-proof inputs, outputs, and computations to support advanced smart contracts on any blockchain. However, discussing the security features of the oracle network is out of the scope of this paper. We have used Chainlink oracle network [38] which is currently the most adopted oracle network [39]. The requirement of having the oracle network in our design can be considered as a limitation of our scheme. This is mainly due to the fact that the security of our scheme relies on the security of the oracle network.

As can be seen from the evaluation results, the costs of deployment of the smart contract, recording a storage task, and submitting a challenge request are small making our proposed scheme practical. On the other hand, the computation cost for PoS on the storage provider and storage verifier is low and negligible. Note that one of the main benefits of our proposed scheme is that the network does not continuously verify the storage service but only once the storage client submits a challenge request. This can significantly improve the performance as the network and the storage provider do not need to consume resources for PoS. As the dominant strategies of players are sharing data and not submitting a challenge request for the storage provider and the storage client respectively, it is expected that the system does not need to execute PoS during the contract unless the storage provider fails to provide the storage service.

Note that for every update request (changing data), the Merkle root stored on-chain should be updated as well. In this case, the smart-contact should verify the client's and storage provider's signatures to ensure that both parties are agreed with the update. Such an update is equivalent to recording a new storage task in our current implementation. Therefore, our proposed scheme works best for storing the archived data, and it is inefficient for storing data with frequent change requests. Improving the scheme to handle update requests more efficiently can be considered as a future work.

6 Conclusion

In this paper, we have introduced a novel game-theoretic mechanism for the decentralized storage network allowing the client to challenge the storage provider. This allows us to eliminate the requirement of having continuously verifying the storage provider which in turn improves the performance of DSNs. Moreover, the client is protected from service denying attack where a dishonest storage provider submits proof of storage to the network while refusing service to the client. Our proposed model is pluggable into

any blockchain platform with smart contract execution capability. We leverage the smart contract and oracle network to govern the rules of the storage contract. We have implemented our scheme using Solidity language and Chainlink oracle network. The performance result demonstrates the applicability of our scheme.

REFERENCES

[1] S. de Figueiredo, A. Madhusudan, V. Reniers, S. Nikova, and B. Preneel, "Exploring the storj network: a security analysis," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pp. 257–264, 2021.

[2] G. Ateniese, L. Chen, M. Etemad, and Q. Tang, "Proof of storage-time: Efficiently checking continuous data availability," in *Proceedings of the 25th network and distributed system security symposium (NDSS)*, pp. 1–15, Internet Society, 2020.

[3] "Filecoin: A decentralized storage network," tech. rep., Protocol Labs, 2017.

[4] D. Vorick and L. Champine, "Sia: Simple decentralized storage," tech. rep., 2014.

[5] "Storj: A decentralized cloud storage network framework," tech. rep., Storj Labs, Inc., 2018.

[6] V. Tron, A. Fischer, D. N. A, Z. Felföldi, and N. Johnson, "swap, swear and swindle: incentive system for swarm," tech. rep., Ethersphere, 2016. Ethersphere Orange Papers 1.

[7] IPFS, "Interplanetary file system."

[8] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of space," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II* (R. Gennaro and M. Robshaw, eds.), vol. 9216 of *Lecture Notes in Computer Science*, pp. 585–605, Springer, 2015.

[9] 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*, pp. 598–609, 2007.

[10] G. Ateniese, R. Di Pietro, L. V. Mancini, and G. Tsudik, "Scalable and efficient provable data possession," in *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pp. 1–10, 2008.

[11] C. C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 4, pp. 1–29, 2015.

[12] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Annual cryptology conference*, pp. 90–108, Springer, 2013.

[13] M. Campanelli, D. Fiore, N. Greco, D. Kolonelos, and L. Nizzardo, "Incrementally aggregatable vector commitments and applications to verifiable decentralized storage," in *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 3–35, Springer, 2020.

[14] Y. Du, H. Duan, A. Zhou, C. Wang, M. H. Au, and Q. Wang, "Enabling secure and efficient decentralized storage auditing with blockchain," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[15] H. Yu, Q. Hu, Z. Yang, and H. Liu, "Efficient continuous big data integrity checking for decentralized storage," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1658–1673, 2021.

[16] D. Catalano and D. Fiore, "Vector commitments and their applications," in *International Workshop on Public Key Cryptography*, pp. 55–72, Springer, 2013.

[17] R. W. Lai and G. Malavolta, "Subvector commitments with application to succinct arguments," in *Annual International Cryptology Conference*, pp. 530–560, Springer, 2019.

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

[19] J. Buchmann and H. C. Williams, "A key-exchange system based on imaginary quadratic fields," *Journal of Cryptology*, vol. 1, no. 2, pp. 107–118, 1988.

[20] B. Libert and M. Yung, "Concise mercurial vector commitments and independent zero-knowledge sets with short proofs," in *Theory of Cryptography Conference*, pp. 499–517, Springer, 2010.

[21] S. Gorbunov, L. Reyzin, H. Wee, and Z. Zhang, "Pointproofs: Aggregating proofs for multiple vector commitments," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2007–2023, 2020.

[22] S. Srinivasan, A. Chepurnoy, C. Papamanthou, A. Tomescu, and Y. Zhang, "Hyperproofs: Aggregating and maintaining proofs in vector commitments," *Cryptology ePrint Archive*, 2021.

[23] C. Papamanthou, E. Shi, R. Tamassia, and K. Yi, "Streaming authenticated data structures," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 353–370, Springer, 2013.

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

[25] A. Juels and B. S. Kaliski Jr, "Pors: Proofs of retrievability for large files," in *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 584–597, Acm, 2007.

[26] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of retrievability: Theory and implementation," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 43–54, 2009.

[27] D. Cash, A. Küpçü, and D. Wichs, "Dynamic proofs of retrievability via oblivious ram," *Journal of Cryptology*, vol. 30, no. 1, pp. 22–57, 2017.

[28] Y. Dodis, S. Vadhan, and D. Wichs, "Proofs of retrievability via hardness amplification," in *Theory of Cryptography Conference*, pp. 109–127, Springer, 2009.

[29] B. Fisch, J. Bonneau, N. Greco, and J. Benet, "Scaling proof-of-replication for filecoin mining," tech. rep., Technical report, Stanford University, 2018. https://web. stanford. edu . . . , 2018.

[30] K. Rzadca, A. Datta, G. Kreitz, and S. Buchegger, "Game-theoretic mechanisms to increase data availability in decentralized storage systems," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 10, no. 3, pp. 1–32, 2015.

[31] W. Wang, D. Niyato, P. Wang, and A. Leshem, "Decentralized caching for content delivery based on blockchain: A game theoretic perspective," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2018.

[32] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, and A. van Moorsel, "Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 211–227, 2017.

[33] M. H. Manshaei, M. Jadliwala, A. Maiti, and M. Fooladgar, "A game-theoretic analysis of shard-based permissionless blockchains," *IEEE Access*, vol. 6, pp. 78100–78112, 2018.

[34] Chainlink, "What is the blockchain oracle problem?," tech. rep., 2020.

[35] A. Egberts, "The oracle problem-an analysis of how blockchain oracles undermine the advantages of decentralized ledger systems," *Available at SSRN 3382343*, 2017.

[36] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[37] E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopoulos, "Zero-knowledge accumulators and set algebra," in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II* (J. H. Cheon and T. Takagi, eds.), vol. 10032 of *Lecture Notes in Computer Science*, pp. 67–100, 2016.

[38] "Chainlink 2.0: Next steps in the evolution of decentralized storage networks," tech. rep., Chainlink Labs, 2021.

[39] M. Kaleem and W. Shi, "Demystifying pythia: A survey of chainlink oracles usage on ethereum," in *International Conference on Financial Cryptography and Data Security*, pp. 115–123, Springer, 2021.

[40] I. Vakilinia, S. Vakilinia, S. Badsha, E. Arslan, and S. Sengupta, "Pooling approach for task allocation in the blockchain based decentralized storage network," in *2019 15th International Conference on Network and Service Management (CNSM)*, pp. 1–6, IEEE, 2019.

This article has been accepted for publication in IEEE Transactions on Network Science and Engineering. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2023.3245326

Iman Vakilinia (Senior Member, IEEE) received the PhD degree in computer science and engineering from University of Nevada Reno, in 2019. He is currently an Assistant Professor with the School of Computing in the University of North Florida. His research interests include Cybersecurity, and Game-Theory.

Weihong Wang received a Master's degree in Computer Science and Engineering from the Hong Kong University of Science and Technology in 2022, and a bachelor's degree in Cyber Security and Engineering from the Huazhong University of Science and Technology in 2020. She will be pursuing a Ph.D. degree in computer science at KU Leuven. Her research interests include blockchain systems and privacy.

Jiajun Xin received his Master's in computer science and engineering from the University of Nevada Reno, in 2017 and his Bachelor's from the Dalian University of Technology, in 2015. He is currently a Ph.D. student at the Hong Kong University of Science and Technology. His research interests include Cryptography and blockchain.