SoK: Modelling Data Storage and Availability

Carlo Brunetta[1] and Massimiliano Sala[2]

[1] Independent Researcher, Bagnolet, France brunocarletta@gmail.com
[2] Department of Mathematics, University of Trento, Trento, Italy
massimiliano.sala@unitn.it

Abstract. Our digital society's ever-increasing demand for data storage has driven up costs and security concerns, particularly with the shift towards outsourced third-party storage providers. This transition raises critical issues regarding privacy, trust, and data availability, i.e. the assurance that stored data remains accessible and retrievable.
This paper introduces a novel model that abstracts data-storage mechanisms by identifying distinct entities, each with a specific role in managing the data flow. Our model is designed to describe storage mechanisms, ranging from centralized to fully-decentralized systems, together with some data availability guarantees.
We highlight the underlying trust assumptions, providing a guideline for understanding application requirements and systematizing knowledge on data storage and availability. To illustrate our model usefulness, we examine several real-world data-storage-and-availability solutions, classifying them within our model as well as showcasing advantages and disadvantages. We conclude by comparing these solutions, which lets us propose open questions and future research directions for data-storage-and-availability methodologies.

Keywords: Data Storage · Data Availability · Decentralization · blockchain

1 Introduction

Our society has become deeply integrated with digital services designed to understand our needs, anticipate them and better align with our individual requirements. This level of customization is the new expected norm, shaping our interactions with technology and influencing our daily lives. The vast amount of data we generate and own is the driving force behind this digital experience. This data describes everything about our online activities and can come from any device connected to the web. This sheer volume of information highlights a significant engineering challenge: how to handle this ever-growing data flow.

Data management is not merely about quantity, since data might be highly confidential, thus requiring robust privacy guarantees and clear access controls, to ensure that only authorized entities can obtain or modify such confidential data. Moreover, despite its digital representation, data must be physically stored somewhere. This fundamental requirement is sometimes neglected, but it is crucial for ensuring the availability, reliability, and integrity of data. These aspects are mere examples of the multifaceted nature of data management.

Our Contribution. This paper aims to introduce a novel abstract model for both the data-storing process and the availability-verification process. Our model identifies several roles and procedures involved in data storage, providing a clear framework for comparing storage strategies. By design, our model categorizes storage-and-availability processes, ranging from local and centralized solutions to fully-decentralized protocols, together with different levels of trust, from high-trust to trust-less domains.

Our model provides a taxonomy of the solutions' landscape, enabling a better understanding of each approach. To illustrate this, we provide examples of real-world storage-and-availability solutions and classify them into the model, so that we can discuss the pros and cons of alternative methodologies. Our ultimate goal is to provide a guideline to empower developers and engineers to make informed choices on the most appropriate data storage solution required by their applications.

Related Work. Our paper joins a collection of recent systematization-of-knowledge works in the area of data storage and availability. Unsurprisingly given the strong link between blockchain technology and cryptography ([24],[10]), they mainly focus on blockchain-based approaches. Raikwar *et al.* [22] analyse the cryptographic primitives/protocols used in the blockchain domain, where some are related to the confidential storage and availability of stored data, e.g. they explain proof of retrievability (PoR), how this primitive is used to verify the retrievability of stored data, and they describe a specific instantiation for Bitcoin called Retricoin [25]. Zahed Benisi *et al.* [32] provide a comprehensive overview on the distributed and decentralized storage solutions with a major focus on the possible applications, e.g. rollups, outsourcing data storage while maintaining verifiable traceability, as well as some open challenges. Ernstberger *et al.* [11] provide a thorough analysis on data sovereignty, i.e. the ability to have control on the data, and how this is achieved or guaranteed in decentralized solutions. Li *et al.* [16] collect known decentralized storage solutions and provide a comparison of the technical cryptographic primitives/protocols they use.

We complete these works with our abstract model which provides a more precise characterization of each solution and which can categorize any solution, even centralized ones, thus creating a guideline for evaluating data-storage systems. All of these applications are extensively described in further works such as Gudgeon *et al.* [12] and their taxonomy of layer-2 solutions, where decentralized storage with rigorous availability verification is paramount.

Paper Organization. Section 2 describes our data storage model and classifying properties , of which we provide an explanation. It also includes some assumptions, conjectures and (possibly mandatory) cryptographic primitives. Section 3 lists real examples, how they are designed and how they fit into the data storage model. We place a strong emphasis on decentralized solutions and their design choices. Section 4 summarizes our findings by providing a comparison of the previous examples, along with some open questions and future research directions.

2 Abstract Data Storage Model

This section introduces our data storage model, the entities composing it and the essential properties used to classify applications. Regarding the essential properties, we focus on *data availability* being the main requirement for any realistic application, i.e. if data is not available, no other property makes sense.

As depicted in Figure 1, our model identifies conceptually different entities representing separated data-handling phases:

- ▷ *Owner* $\mathcal{O}$: this entity owns the data and is the one requesting the data to be stored. The owner is responsible for preparing the data to store and for finding a contractual agreement with other entities. This agreement outlines how the handling will be conducted and the guarantees that must be provided, e.g. encryption requirements, distribution process to multiple storing nodes, availability guarantees, etc.
- ▷ *Handler* $\mathcal{H}$: the handler plays the role of a *file-system*, i.e. it coordinates the owner's storage requests, finds the appropriate storage nodes, keeps track of the location of stored data, probes the storage for availability, etcetera.
- ▷ *Storage Node (storer)* $\mathcal{S}$: this entity is responsible for physically storing the data and for retrieving it whenever requested by the handler.
- ▷ *Retriever* $\mathcal{R}$: the retriever is the designated entity that receives the owner's data from the handler. The retriever is allowed to request the designated data from the handler.

All these entities and their differentiated roles must coordinate to allow the correct, safe and trustworthy data storage. This coordination can be classified by specific *flavoured* properties, later discussed in detail:

- ○ *Access Control*: describing who has the right to access the data and if this data must preserve confidentiality during the storage, either from potential leaks or from a malicious handler/storer. All these are effectively metadata policies that the owner requests as part of the storage agreement. The access control allows a clear distinction of the scenarios where the receiver and the owner are the same entity or not.
- ○ *Data Handling Technique*: the Handler has the task of mapping the data's retrieval request to a concrete storage location, thus creating a mapping between requests and where data is stored, basically a *file-system* instantiation. This process can be made via as a centralized, distributed or full decentralized protocol.
- ○ *Data Storage Technique*: once the data arrives at the Storage Node, this must effectively be stored on a physical medium. Similarly to data handling, this phase can be made via either centralized, distributed or decentralized solutions. Furthermore, each storage node might use different storing methodology to provide a higher level of resilience against faulty storage media.
- ○ *Proving Data Availability*: whenever the data is stored, the owner (or receiver) might request a proof that the data is available for retrieval. This proof can be used to coordinate a recovery procedure, to halt rewarding to

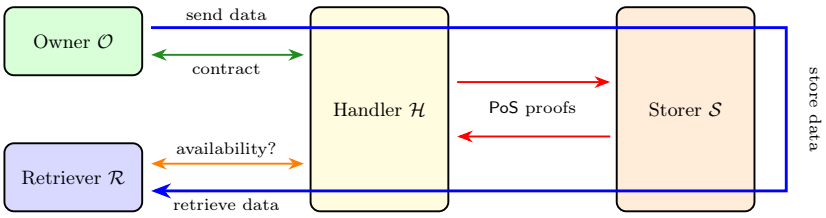

Fig. 1. Our abstract storage model with the fundamental actions between the entities as arrow. A major (blue) highlight of the data-flow is given.

the storage node, to alert the receiver for the lost of stored data or for other application-oriented responses.

2.1 Centralized, Distributed and Decentralized

Except for the owner which we assume to always be solo, all the other entities can be composed of *single or multiple* parties, hence introducing a trade-off between trust and resilience. Having single or multiple parties suggests concepts such as *centralized*, *distributed* and *decentralized* and, despite being often used interchangeably, we make a clear distinction between the terminologies:

- ▷ a *centralized* entity is defined by a single party representing the whole;
- ▷ a *distributed* entity is a *selected group* of parties where new members might join only according to strict rules (to build up a trust baseline);
- ▷ a *decentralized* entity is composed by an *open group* of parties which have limited trust between each other since everyone is (mostly) free to join.

In other words, a centralized entity is composed by a single party, while both a distributed and decentralized entity are composed by several parties interacting with each other to accomplish their goals. The difference between distributed and decentralized is found in the *underlying trust* between the parties and in the related enrolment's procedure. We define an entity to be decentralized if *anyone* can join the entity without specific prerequisites or trust requirements, similarly to how anyone can join a decentralized blockchain network by merely running a specific protocol's code. While, if the enrolment's procedure comprehends only *selected* individuals, we define such an entity to be distributed because the member selection process might not follow fair or transparent principles.

2.2 Data Availability

A fundamental requirement for any application is *data availability* (DA) which can be described as the concept that *"data must be proved available"* meaning that it should be possible to prove that any data stored is indeed available and can be retrieved by the designated receiver. Differently from other works, we use

interchangeably the concept of *availability* and *retrievability* since, in our model, if data is retrievable by the correct receiver, then data must be available. For completeness, *data immutability* is an orthogonal fundamental property which we assume to hold at all time since *"data should not be (maliciously) modifiable"* guaranteeing the retrieval of the original data.

Before moving to known mechanism used to guarantee data availability, let us collect precise goals or properties for the proving mechanism:

DA.1 *formal verification* denotes that the proving mechanism provides a computationally verifiable proof of data availability, e.g. by using well-established cryptographic primitives such as Merkle trees;

DA.2 *timed proving* indicating that the verification procedure should formally prove the data availability at the exact time of proving and not before, i.e. the proof request acts as an unpredictable challenge forcing the proving mechanism to maintain the data available;

DA.3 *selective verification* indicating the possibility to prove the availability of a selected portion of the data. This can be useful when multiple files are stored and only a subset of them must be checked or if the data is too heavy, e.g. the availability proof might be reduced to uniform randomly selected pieces, thus providing a measurable probability of the whole data being available;

DA.4 *distributed storage* meaning that storage should be distributed between different nodes that interact to improve efficiency, rather than merely replicating the storage. Adding a network of storing nodes introduces security and protocol's complexity with the advantage of (hopefully) achieving more advanced features, e.g. threshold procedures or costs offloading;

DA.5 *proving resilience* denoting that if multiple storage nodes actively cooperate to store data, they are able to prove the data's availability even if some of the nodes are unavailable;

Important aspects when considering data availability are the data *confidentiality* (depending on the trust model required), the *time-frame* considered or the *storage duration* for which the data is requested to be available (outside this window, there might not be any guarantees).

Any DA solution is defined by some general procedures, designed to commit some sort of verification material connected to the data stored, which create a random computational challenge for the storer that can be answered only if they have the stored data available.

In the literature, there are many examples of protocols providing DA guarantees, each with a specific twist required by their application [20, 26, 21, 7], e.g. provable-data-possession [2], proof-of-storage [4, 31], proof-of-retrievability [25], proof-of-ownership [13], proof-of-space-time [19, 33, 3], or similar [8]. While each solution has its own techniques, we identify some common ones:

- *erasure codes* [6] are often used to extend data into a longer version, from which some shards are generated and distributed to some storage nodes. The main reason is that by correctly tweaking the code's parameters, the data might be retrievable even with a smaller amount of shards, hence providing resilience against many attacks on/from the storage nodes.

- *Merkle trees* (or similar hash-based data structure) are used to compress DA proofs based on hash-evaluation into a more compact format while providing paramount immutability guarantees. These solutions offer high throughput at the cost of limited feature-extensions capability, e.g. many solutions have private verifiability (only the receiver can verify the DA).
- *Zero-Knowledge* (ZK), as in succinct non-interactive/transparent argument of knowledge (SNARK/STARK) [17], provide a more computationally-expensive proving protocol at the benefit of allowing higher degree of expressibility. For example, differently from the hash-based solutions, many ZK solutions are designed to permit public DA verifiability.
- *Commit-and-open* solutions, especially via Kate *et al.* [15] polynomial commitment scheme, where the idea is to encode data into a secret polynomial, which is publicly committed and provided to the storage node. To generate a DA proof, the receiver requests the evaluation of the polynomial on some challenge points, which allows the verification of the polynomial's knowledge (this knowledge implies the availability of stored data).

Typically, these solutions are considered in a DA *layer* which is the network of interacting entities that creates, handles and verifies the DA proof requests. In the vast majority of known approaches, this network is based on a (public) ledger to allow for a sequential and traceable transcript of the stored data's history. This ledger also provides an easier computational control of the verification process by means of smart contracts (or similar programs) that automatically verify DA cryptographic proofs.

2.3 Rewarding Mechanism

Both storing and proving data availability have a non-negligible computational cost, which must be considered in applications, thus requiring the introduction of a *rewarding mechanism*, i.e. how the effort by any entity should be rewarded (and by whom).

Intuitively, the owner $\mathcal{O}$ and retriever $\mathcal{R}$ should pay for the storage and availability service, but they should be compensated by the handler if their data is lost (or turns out to be not available in the agreed availability-window). This suggests $\mathcal{O}$ and $\mathcal{R}$ pay the handler $\mathcal{H}$, the storer $\mathcal{S}$, or both, depending on the entities' independence and the (agreed) work compensations. The total compensations should consider: the effective costs for the data storage, all the DA proofs computed by the storage nodes, the handler's proof verification and also some management overhead.

As later showcased by the examples in Section 3, the burden of these costs can be effectively addressed in centralized or distributed scenario, where payments can be done at the final verification of the correct execution of the service and, if something goes wrong, the entities are supposed to follow the agreed contract (which is often legally binding). For example, if $\mathcal{S}$ loses the data, $\mathcal{R}$ and $\mathcal{O}$ will request a refund from $\mathcal{H}$ which must compensate them with an amount demanded to $\mathcal{S}$ for breaching the agreement between $\mathcal{S}$ and $\mathcal{H}$.

The decentralized scenario follows the same payments and claims requirements but, depending on the technology used to obtain the decentralization, the costs might be required to be provided upfront by $\mathcal{O}$. This problem demands a stricter automatic mechanism to ensure the correct protocol's execution. The same might apply to $\mathcal{H}$ or $\mathcal{S}$ whenever involved in the DA verification process, e.g. these entities might lock a deposit that will be returned if the verification is done correctly (together with the appropriate compensation for the work). Otherwise, the deposit is used as collateral, i.e. to refund $\mathcal{O}$ for the data loss.

The rewarding mechanism must therefore provide a cleverly designed incentive that avoids favouring malicious behaviour, otherwise malicious entities would never be punished. Furthermore, it is convenient to publicly maintain an immutable trace of the rewards, at the advantage of a traceable reputation mechanism, and to favour algorithmically-defined procedures that automatically partitions the rewards, e.g. via smart contracts.

3 Classifying Known Solutions

In this section we classify several known solutions in our model. We also provide explanations on the used methodologies/technologies, together with: some considerations on costs, the control over data and the required trust level. The examples are sorted according to the degree of decentralization.

3.1 Autonomous Storage

The simplest solution is to have a unique party acting in all the entities roles, i.e. the party self-hosts and manages its own data storage and availability.

Such a solution is clearly fully centralized, while confidentiality and resilience highly depend on the choices made. For example, the storage can be setup as a RAID system to increase redundancy and resilience against failing hard-drives. Availability verification does not require any specific procedure, since the owner has direct (or close enough) access to the storage system. All the costs, both hardware and management, are on the owner. There is no effective trust required, except possibly for the hardware itself, which might contain a backdoor.

3.2 Self-Handled Storage

The second-simplest solution considers an owner that outsources its storage to either a single or multiple storage nodes, in some sort of *storage renting* where the owner takes the additional role of the data handler.

We classify self-handled storage to be a centralized handler with distributed storage. DA mechanisms can be implemented since they are specific protocols. In this scenario, the data owner *autonomously* decides to reward verified DA proofs and all the rewarding rights/guarantees are executed as per contract. To have a more algorithmic rewarding mechanism, the DA verification might be executed over any public ledger and with a smart contract. On the other hand, this latter

idea would defy the purpose of *keeping-it-simple*, since it would turn the whole system into a sort of decentralized storage solution, discussed later.

3.3 Cloud Storage

Whenever the data owner outsources the data handling to an either centralized or distributed entity, the scenario depicts the typical cloud storage service where users pay a cloud for storage space without any responsibility on how to prepare data for storage. Usually in such solutions, the storage nodes are under the full control of the handler, which is a unique bigger entity that coordinates the storage as it better fits its infrastructure. This leads to more efficiency, as for example a storage contract with fewer retrievability guarantees creates an opportunity for the cloud server to save in storage-hardware costs, with the saving going for investment in some other parts of its system (e.g., cyber security).

Data availability proofs are usually not provided for cloud storage service, since any data loss is often handled according to some contractual agreements (e.g., monetary compensation via an insurance). Yet, this cost-efficient strategy relies on completely trusting the cloud's promise to keep the data available.

Similarly to self-handled storage, we classify any common cloud storage to be a centralized handler with distributed storage or distributed handler and storage. DA guarantees are (usually) not provided because a legally binding contract is often used to handle the scenario of a data loss. Internally, most probably the cloud storage services have mechanisms to distribute the stored data to minimize space costs and to create redundancy to verify that all the data is available (DA.4, DA.5). Unfortunately, this information is not provided to the data owner.

3.4 Decentralized Storage

The final class considered are decentralized storage solutions, commonly referred as *data availability layers*. The conceptual difference from the other examples is that anyone can join the decentralized network by providing computational power for data handling or storage space. However, data availability verification is paramount, since there is no underlying trust among potentially anonymous parties. These observations lead to consider a public ledger that algorithmically coordinates at least the reward protocol (if not the entire system), which must compute and distribute the economic incentives to honest-behaving entities.

The management costs are obviously higher than those of other solutions, since the coordinating protocols must build trust from scratch, which is known to be a very expensive process. Fortunately, the security guarantees are stronger, formally proven and verifiable (at least in principle). Additionally, the rewards themselves, typically awarded via cryptocurrencies or specific crypto-tokens, can increase their intrinsic value with the increase of demand for the decentralized storage service. The value growth pushes more participation in the storage service, providing higher stability and resilience for the underlying decentralized network.

Differently from other solutions, a decentralized storage based on a public ledger can more easily be used for *roll-ups scaling* mechanism. With a roll-up we mean a protocol aiming for a higher transaction throughput by performing an aggregation of several transactions into a single transaction. We consider only roll-ups that also provide an aggregation proof that is effectively reported and verified (possibly on the main ledger). Any such (aggregated) transaction plus its proof needs significantly less space than all initial transactions combined. This space saving opens the door to the scaling down of transaction throughput on the main blockchain. Without entering into the discussion of computational costs for such scaling solutions, which is abundantly discussed in the research community (see e.g., [12, 14, 23]), in any case the initial transactions must be stored somewhere (i.e., in a DA) and must remain available for verifiers, who can determine later the correctness of the aggregated transaction and its proof.

From the extensive solution space of DA layers [1, 26, 32, 7, 16], for our analysis we select Storj [29], FileCoin [21], Avail [5], Sia [28], Arweave [30] and Celestia [9],checking their techniques to achieve DA guarantees.

Storj. Storj is a cloud-storage service that allows users to store their data in a distributed cloud system. Storing devices are awarded for the space (and time) provided and used. Storj's network is composed by three types of nodes: clients of the service, Storj's nodes, which in practice compose the DA layer, and some special network's coordinators, which keep note on both the storage network's distribution and the rewards accrued by the network's nodes.

From a technical point of view [29], the owner $\mathcal{O}$ encrypts via a symmetric encryption scheme, e.g. $\mathsf{AES256-CTR}$, their data m and obtains an encrypted ciphertext c which is split into specific-size shards $\{\mathsf{c}_i\}_{i\in I}$. For each shard, the storage node $\mathcal{S}$ provides a root node and a specific tree level, so that the owner instantiates a proof-of-retrievability (PoR) by computing, intuitively, a salted Merkle tree with all shards of $\mathcal{O}$'s interest. To allow resilience against failing storage nodes, Storj suggests the shard to either be replicated to multiple nodes or to use erasure-encoding schemes. To select $\mathcal{S}$, Storj is based on a distributed hash table called Kademlia [18], which creates and maintains a distributed message routing among nodes, together with some security guarantees. This primitive allows the association of a unique shard's identifier, e.g. a digest of the shard, with a unique node $\mathcal{S}$ that is responsible for the storage, up to coordinated modifications to the distributed hash table's key-value entries.

The DA verification is done via a challenge-response protocol, where $\mathcal{O}$ provides to $\mathcal{S}$ a random salt used during the PoR's initialization. If the file is available, $\mathcal{S}$ can provide a Merkle proof thus guaranteeing the retrievability of its shard. The verification process is not automatic and it is intrinsically limited by the number of salt values chosen during initialization.

Regarding the handler's role, Storj considers a software solution that should facilitate networks communications together with the contractual agreements with the storer for their service (which are coordinated on a public ledger). Their idea is to detach such responsibilities to a software, which can either be hosted locally or outsourced to a trusted party.

We classify Storj as a centralized handler with decentralized storage able to provide non-timed formal DA proofs of the whole encrypted data (DA.1). The storage is distributed between the nodes and can be made more resilient (DA.4, DA.5). Access control is completely left to the owner's control and, without major modification to the protocol, it is not trivial to integrate a protocol that allows new access permissions after-storing, e.g. via proxy re-encryption.

Arweave. Arweave [30] is a decentralized data storing protocol that aims to provide the infrastructure for the *permaweb*, i.e. a decentralized version of the web where content cannot be easily removed, censored or modified (because distributed over multiple supposedly-independent nodes). The protocol defines an independent blockchain based over a custom proof-of-work mechanism. Any transaction contained in the block defines either some exchange of cryptocurrency between wallets (e.g. payments) or a *data transaction*, i.e. a transaction indicating that some data is requested to be stored by the network (once the transaction is validated, the network distributes the storing of its data).

Arweave opts for an ad-hoc protocol to seek maximal usage efficiency while maintaining storage and computational costs relatively low, at the cost of a more complex tokenomics design (which is well motivated in their documentation [30]). The core DA verification is executed periodically (approximatively every two minutes) via a Succinct Proof of Access (SPoA) that acts similarly to Merkle-tree/hash-based PoS primitive by providing the proof for the whole Merkle-path for some challenged leaf. Differently from other solutions, SPoA is mainly designed to be used in more complex protocols able to prove the availability of replicated data on random offsets and time, by means of a verifiable delay function (VDF). We classify Arweave as a decentralized handler and storage system, designed over an ad-hoc blockchain, which provides timed DA proofs verified by the network and used to mine the next block (DA.1, DA.2, DA.3). Each node decides what data to replicate based on its country's legislation and some tokenomics incentives, needed for the replication of less distributed data (DA.4, DA.5).

FileCoin. FileCoin [21] is a decentralized storage framework that introduces an economically incentivized mechanism on top of the InterPlanetary File System (IPFS) [27], i.e. a decentralized peer-to-peer storage and retrieval network mostly used for decentralized web services. Differently from Storj, FileCoin provides the rewarding infrastructure on top of the already existing IPFS storage layer.

Technically, such an infrastructure is based on a public ledger that stores the agreements between owners and storage nodes. These agreements are created via a *bidding* procedure. More specifically, the owner creates an order specifying the storage and availability requirements, while the storing nodes bid to win the deal which is agreed between the parties via its mutual signature. All the storage costs are therefore independently agreed upon between owner and storage nodes. The protocol enables the handlers to be rewarded for both maintaining the different data-structure required and providing security and fairness guarantees in case

of disputation, e.g. if after some time, the stored data is not available or cannot be retrieved.

The IPFS provides *addressable storage-location*, that is, any file in the peer-to-peer system can be uniquely identified by the entire network. Such an identification is specified in the deal, thus clarifying the logical position where the data must be stored. Notably, the file retrieval can be done off-chain by exchanging data chunks for micropayments that can be reported to the ledger for the correct reward (which is managed by the handlers' decentralized network).

Regarding the DA, FileCoin provides a *proof-of-spacetime* primitive that creates PoS proofs for a chosen period, i.e. a proof that the data is retrievable for a specifically chosen window of time. The main idea is to exploit more computational time for the proof generation the proof, while reducing the communication costs (this cost reduction is especially notable when compared with the periodical execution of an entire PoS verification round). Intuitively, the primitive sequentially generates challenges from three inputs: an initial challenge, a counter and the current interaction proof. The computed challenge is used to compute the PoS Merkle proof, in the same spirit as Storj's one, which is the input for a zero-knowledge SNARK that compacts the proof. Providing *"time guarantees"* is obtained by requiring the proving interaction to be executed t times which would require noticeable computational cost (which translates to wall-clock time).

We classify FileCoin as a decentralized handler with decentralized external storage (IPFS) that can provide timed DA proofs verified by the network (via smart-contracts) of the whole or partial data according to IPFS's file management (DA.1, DA.2, DA.3). Even if the storage is distributed over the IPFS's network, there are no (concrete) threshold storing mechanism, rather, the resilience is obtained by having a large replicating network. As per FileCoin, access control is completely left to the owner's control.

Sia. Sia [28] is a framework based on a dedicated blockchain where storage contracts are agreed upon of which design reassembles the Bitcoin blockchain with major differences on the transaction format and goals. In particular, Sia's transaction does not consider a scripting language and mainly focuses on providing the use of a multi-signature scheme where contracts, DA proof and contract updates are the only possible encoded messages. The DA verification are defined using a hash-based PoS and the framework allows for periodic release of rewards without interaction from the data owner, i.e. the contract defines the signing rules for the automatic rewarding of the storage node.

We classify Sia solution as a decentralized handler with a decentralized peer-to-peer storage that provides DA's guarantees on the whole or partial data (DA.1, DA.2, DA.3). As per protocol definition, the storage is not natively distributed, i.e. multiple contracts must be created and the data must be appropriately handled to achieve a somewhat threshold reconstruction mechanism.

Avail. Avail [5] introduces a framework for unifying blockchain networks. For our work, we focus on the underlying DA layer proposed with major interest in how data is prepared and their DA verification mechanism. Avail's DA layer is

based on a decentralized blockchain where storage nodes and handlers, called *validators*, maintain trace of all the stored data and agreements between owners and storage nodes made via smart contracts, similarly in spirit to FileCoin but with a different approach because Avail is not based over IPFS.

The major technical difference is that Avail's ledger stores, together with the entities agreements, the effective DA verification protocol's transcription, i.e. the whole proving is publicly executed and verified on the ledger by strictly following the rules provided in a smart contract. The key point is that the owner is not required to be online to provide a challenge for the verification by the cryptographic primitive. Whenever data must be stored, data is first encrypted (as other solutions) and represented as a matrix of which data chunks for each row is considered as a secret polynomial to be used in the Kate *et al.* [15] polynomial commitment scheme, i.e. the polynomial is homomorphically evaluated on a publicly random secret value, with the related result published as the commitment of the secret polynomial. All the commitments from the rows[3] are posted on the Avail's DA ledger and, if verified, are effectively published.

To prove availability, a smart contract managing the agreement's rewards can be programmed to require a proof over a challenge computed by the current (hash of the) status of the ledger which is (perhaps optimistically) assumed to be unpredictable. Each storer computes the DA proof, composed of the secret polynomial's evaluation on the provided challenge, and the KZG's proof, to allow handlers to verify proofs and correctly execute the smart contract's rewarding mechanism. By cleverly shaping the representation matrix, the protocol allows selective proving, e.g. different files are committed into different polynomials because of their placement in the matrix representation, thus permitting the verification of specific data by requiring the (proved) evaluation of selected polynomials. Such a property enables selective verification that reduces the bandwidth demanded to the network and an increased scalability.

Another technical advantage of using a publicly verifiable scheme, such as KZG, is the possibility to create *light clients*, i.e. handlers in our model that preserve a partial status of the underlying ledger. Each light client can store for a limited time a bounded amount of partial DA proofs. All clients are interconnected into a peer-to-peer network that acts as a *local-cache* of the current DA network. Any of their proofs can be publicly verified, which guarantees that the light clients are not maliciously faking the DA proofs.

The light client's network would offload additional workload from the network allowing any party willing to check the availability of some data to query the local-cache instead. Having lower latency allows a quicker access to the availability proofs and allows a (possible) quicker access to the data/proof.

The natural problems arising are all about correctly handling *cache-lifetime*, e.g. the availability proofs provided by the cache might be different from the real availability which might have consequences, especially if the ledger's handlers

[3] The white-paper [5] specifies the usage of an erasure code to extend either the matrix columns or the computed commitments. The paper does not provide a precise formulation thus we limit our description.

decide to provide a DA proof that will survive in the peer-to-peer cache while effectively making the data unavailable.

Avail considers tokenomics for their entities, where requesting to storage is paid by the owner and the DA proofs are used to finalize the payments via a smart contract, where forcing the availability proofs to be stored on the ledger provides accountability. At the same time, honest validators (handlers) are rewarded by the ledger's block-generation fees plus additional estimated expenses to run smart contracts. Notable, only the light clients are not paid for their effort to lowers the systems' demands however, such a rewarding mechanism might be hard to provide by the very nature of the peer-to-peer network and an unintuitive challenge to solve: it is unclear how to provably count the access to valid cached proofs.

We classify Avail's DA layer solution as a decentralized handler with decentralized storage that provides timed DA proofs verified by the network (via smart-contracts) of the whole or partial data (DA.1, DA.2, DA.3). The effective storage is distributed and allows concrete threshold reconstruction mechanism using erasure codes (DA.4). The cached-proofs, stored by the light-clients peer-to-peer network, provide redundancy of the main ledger, systematically offloading efforts on it, which can be better spent into the ledger maintenance (DA.5). As in other solutions, the owner fully controls the access on data.

Celestia. Celestia [9] is a development framework for decentralized application of which data storage and availability layer is based on Al-Bassam [7] and Al-Bassam *et al.* [8] works. Similarly to other solutions, the DA layer is an independent blockchain with a self-sustaining tokenomics that rewards honest behaviour of the network and storage nodes. Even if the DA guarantees are hash-based, the underlying proving methodology considers a two-dimensional matrix where the data is expanded into a Reed-Solomon Encoded Merkle Tree, i.e. the data is first extended via a Reed-Solomon code and later a Merkle-tree root is evaluated for each column and row for later combining all these values into a single root value. The matrix structure, together with the increased number of Merkle-roots considered, enables the verification of only selected columns/rows and, similarly to Avail, lets light clients reduce the distribution's costs of the DA proofs from the main network.

From the white-paper underlying the protocol [7, 8], we classify Celestia's DA layer solution as a decentralized handler and storage that guarantees DA verification of the whole or partial data (DA.1, DA.3). The storage is distributed and the protocol is designed to use erasure codes techniques (DA.4). This way, Celestia obtains the decentralization of the DA proving costs among nodes (DA.5).

4 Discussion and Future Directions

All known solutions introduce some tweaks to better fit the application they are designed for. We report in Table 1 the classification of such solutions into our model and the properties highlighted in Section 2. Our model does lead to a systematic classification highlighting differences between these solutions, e.g.:

Table 1. Summary for the examples of Section 3, according to our model (Section 2). The symbol ✓ indicates presence/compliance, ∼ if optionally implementable, ✗ lack of feature, **?** if unknown from the literature, and a dash "–" if not relevant. We denote the owner with $\mathcal{O}$, the handler with $\mathcal{H}$, the storer with $\mathcal{S}$ and the retriever with $\mathcal{R}$.

	Roles				Access				Storage				Handling					Data Availability								
Protocol	**Owner**	**Handler**	**Storer**	**Retriever**	**Full $\mathcal{O}$ Control**	**Confidentiality**	**Designated $\mathcal{R}$**	**Proxy re-encryption**	**$\mathcal{S}$ Centralized**	**$\mathcal{S}$ Distributed**	**$\mathcal{S}$ Decentralized**	DA Proving	**$\mathcal{H}$ Centralized**	**$\mathcal{H}$ Distributed**	**$\mathcal{H}$ Decentralized**	**Contractual Market**	**Redundant Network**	DA.1	DA.2	DA.3	DA.4	DA.5	**Automatic DA**	**Smart Contract DA**	**Cryptography**
Autonomous	$\mathcal{O}$	$\mathcal{O}$	$\mathcal{O}$	$\mathcal{O}$	✓	–	–	–	✓	–	–	✗	✓	–	–	✗	✗	–	–	–	–	–	–	–	–
Self-Handled (single storer)	$\mathcal{O}$	$\mathcal{O}$	$\mathcal{S}$	$\mathcal{O}$	✓	–	–	–	✓	–	–	∼	✓	–	–	✓	✗	∼	∼	∼	–	–	–	–	As $\mathcal{O}$ desire
Self-Handled (multi storer)	$\mathcal{O}$	$\mathcal{O}$	$\mathcal{S}$	$\mathcal{O}$	✓	–	–	–	–	✓	–	∼	✓	–	–	✓	✗	∼	∼	∼	–	–	–	–	As $\mathcal{O}$ desire
Cloud Storage (typical)	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{R}$	∼	✗	✓	✗	–	✓	–	✗	✓	✓[4]	–	✓	✓	–	–	–	✓	✓	–	–	–	–
Storj	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	✓	?	–	✓	✓	✓	✗	✗	✓	✓	∼	✓	Hash-based PoS
FileCoin	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	–	–	✓	✓	✓	✓	✓	✓	–	–	∼	∼	PoS and SNARK
Avail	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	–	–	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	KZG
Arweave	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	–	–	✓	?	✓	✓	✓	✓	✓	✓	✓	?	Hash SPoA, VDF
Sia	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	–	–	✓	✓	✗	✓	✓	✓	✗	✗	✓	✓	Hash-based PoS
Celestia [7, 8]	$\mathcal{O}$	$\mathcal{H}$	$\mathcal{S}$	$\mathcal{R}$	✓	✓	✓	∼	–	–	✓	✓	–	–	✓	?	✓	✓	✓	?	✓	✓	?	?[5]	Merkle Matrix

- Differently from other protocols, Storj seems to prefer a (software) centralized solution. One of their future goal is to provide a (software) decentralized solution to align with similar different solutions.
- Differently from Avail and Storj, FileCoin does not develop specific redundancy/threshold mechanism to protect against the loss of data shares (because this should be handled by the underlying IPFS layer).
- FileCoin is designed for direct coordination between $\mathcal{O}$'s storage demand and store nodes, while Storj and Avail focusses more on an offer-framework where the deals are agreed upon a market which is available on the public ledger.
- Avail is the only protocol to define a peer-to-peer network composed of light clients that crate an effective cache of the DA layer. This idea provides clear intra-network optimizations that improve with the increase of the peers network. However, the white-paper [5] does not offer enough details to under-

[4] Cloud storage providers might distribute their workload or provide multiple access point for the storage service. We identify both as possible without separating the table's entry.

[5] From the research articles, it is not specified if the DA verification can be spontaneously requested by the network, if the proofs can effectively be prepared before hand and/or if the layer provides smart contracts to create a contract.

stand the limitations, how natural cache-memory problems are solved and which security assumptions are proved.[6]

4.1 Further Comments

While investigating the literature, we noted that no PoS/DA solution (with detailed technical white-paper) is designed around the idea of utilizing the periodical DA proofs as a possible mechanism to facilitate a possible data-recovery procedure. For specific scenarios where the data is not too big, imagine the handler verifies and stores t DA proofs on the ledger which are exactly the amount of proof requested by a contractual agreement. The consecutive protocol's execution would suggest that the storage node provides to the retriever the data. However, a malicious retriever might act as the data is corrupted thus forcing a rewarding resolution based on the smart contract's code.

Following our previous discussion, what we suggest is to have a reconstruction algorithm that takes the verified DA proofs and outputs the data. For example, Avail uses KZG polynomial evaluations as proofs which, by algebraic properties, would allow the reconstruction of the whole polynomial if the correct amount of evaluations is known, i.e. one more than the degree of the polynomial. Therefore, a timed mechanism that publicly releases proofs and, after a pre-defined number of periods, automatically permits recovery of committed data.

4.2 Conclusions

The domain of data storage and availability is a rapidly evolving environment where new ideas and techniques often mixes with specialized features oriented to real-world applications. Our model leads to a precise classification of these solutions (and more traditional ones), providing both help for comparing storage systems and a guideline for developers searching the best-fitting framework (that achieves their requirement without introducing additional complexity).

Acknowledgments. The authors would like to thank Ripple's University Blockchain Research Initiative, Amit Chaudhary and the MindCrypt team.

References

1. 0G Labs, 0G: Towards Data Availability 2.0, (2024). `https://0g.ai`

[6] If both the DA proving periodicity and the cache have a lifetime of Δ, then a handler $\mathcal{H}$ may give partial proofs of some data m to the light client's network *exactly* at the same time as the last DA proof is published. If $\mathcal{H}$ is malicious, it might delete m, sell its space for a timespan $< \Delta$, which would let $\mathcal{H}$ query from the cache-network the partial proofs and reconstruct m thus fraudulently fulfilling its DA obligations. $\mathcal{H}$ would gain almost twice the reward at the cost of the light client's network, which are not rewarded.

2. Ateniese, G., Burns, R.C., Curtmola, R., Herring, J., Kissner, L., Peterson, Z.N.J., Song, D.: Provable data possession at untrusted stores. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) ACM CCS 2007, pp. 598–609. ACM Press (2007). https://doi.org/10.1145/1315245.1315318
3. Ateniese, G., Chen, L., Etemad, M., Tang, Q.: Proof of Storage-Time: Efficiently Checking Continuous Data Availability. In: NDSS 2022. The Internet Society (2020). https://doi.org/10.14722/ndss.2020.24427
4. Ateniese, G., Kamara, S., Katz, J.: Proofs of Storage from Homomorphic Identification Protocols. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, pp. 319–333. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_19
5. Avail Team, Avail: A Unifying Blockchain Network, version 2.1. (2024). https://www.availproject.org
6. Balaji, S., Krishnan, M.N., Vajha, M., Ramkumar, V., Sasidharan, B., Kumar, P.V.: Erasure coding for distributed storage: An overview. Science China Information Sciences **61**, 1–45 (2018)
7. Al-Bassam, M.: LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts, (2019). arXiv: 1905.09274 [cs.CR]. https://arxiv.org/abs/1905.09274.
8. Al-Bassam, M., Sonnino, A., Buterin, V.: Fraud and Data Availability Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities, (2019). arXiv: 1809.09044 [cs.CR]. https://arxiv.org/abs/1809.09044.
9. Celestia Labs, Celestia, (2025). https://celestia.org
10. Cimatti, A., Sclavis, F.D., Galano, G., Giammusso, S., Iezzi, M., Muci, A., Nardelli, M., Pedicini, M. Journal of Mathematical Cryptology **19**(1), 20240045 (2025). https://doi.org/doi:10.1515/jmc-2024-0045
11. Ernstberger, J., Lauinger, J., Elsheimy, F., Zhou, L., Steinhorst, S., Canetti, R., Miller, A., Gervais, A., Song, D.: SoK: Data Sovereignty. In: 2023 IEEE European Symposium on Security and Privacy, pp. 122–143. IEEE Computer Society Press (2023). https://doi.org/10.1109/EuroSP57164.2023.00017
12. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Layer-Two Blockchain Protocols. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, pp. 201–226. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_12
13. Halevi, S., Harnik, D., Pinkas, B., Shulman-Peleg, A.: Proofs of ownership in remote storage systems. In: Chen, Y., Danezis, G., Shmatikov, V. (eds.) ACM CCS 2011, pp. 491–500. ACM Press (2011). https://doi.org/10.1145/2046707.2046765
14. Huang, C., Song, R., Gao, S., Guo, Y., Xiao, B.: Data Availability and Decentralization: New Techniques for zk-Rollups in Layer 2 Blockchain Networks, (2024). arXiv: 2403.10828 [cs.CR]. https://arxiv.org/abs/2403.10828.
15. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-Size Commitments to Polynomials and Their Applications. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, pp. 177–194. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_11
16. Li, C., Xu, M., Zhang, J., Guo, H., Cheng, X.: SoK: Decentralized storage network. High-Confidence Computing **4**(3), 100239 (2024). https://doi.org/10.1016/j.hcc.2024.100239
17. Liang, J., Hu, D., Wu, P., Yang, Y., Shen, Q., Wu, Z.: SoK: Understanding zk-SNARKs: The Gap Between Research and Practice, Cryptology ePrint Archive, Paper 2025/172 (2025). https://eprint.iacr.org/2025/172.

18. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Revised Papers from the First International Workshop on Peer-to-Peer Systems. IPTPS '01. Springer-Verlag, Berlin, Heidelberg (2002)
19. Moran, T., Orlov, I.: Proofs of Space-Time and Rational Proofs of Storage, Cryptology ePrint Archive, Report 2016/035 (2016). https://eprint.iacr.org/2016/035.
20. Movement Labs, Movement Network: High-Throughput Fast Finality Move-based Chains Secured by Ethereum, version 0.2.7. (2025). https://www.movementnetwork.xyz/whitepaper/movement-whitepaper_en.pdf
21. Protocol Labs, Filecoin: A Decentralized Storage Network, (2017). https://filecoin.io/filecoin.pdf
22. Raikwar, M., Gligoroski, D., Kralevska, K.: SoK of Used Cryptography in Blockchain. IEEE Access **7**, 148550–148575 (2019). https://doi.org/10.1109/ACCESS.2019.2946983
23. Saif, M.B., Migliorini, S., Spoto, F.: A Survey on Data Availability in Layer 2 Blockchain Rollups: Open Challenges and Future Improvements. Future Internet **16**(9) (2024). https://doi.org/10.3390/fi16090315. https://www.mdpi.com/1999-5903/16/9/315
24. Scafuro, A.: Blockchains and Cryptography. In: Advanced Cryptographic Protocols, pp. 100–130. De Cifris Press (2024). https://doi.org/10.69091/koine/vol-4-P05
25. Sengupta, B., Bag, S., Ruj, S., Sakurai, K.: Retricoin: Bitcoin based on compact proofs of retrievability. In: Proceedings of the 17th International Conference on Distributed Computing and Networking. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2833312.2833317
26. Skidanov, A., Polosukhin, I., Wang, B.: Nightshade: Near Protocol Sharding Design 2.0, (2024). https://near.org/papers/nightshade
27. Trautwein, D., Raman, A., Tyson, G., Castro, I., Scott, W., Schubotz, M., Gipp, B., Psaras, Y.: Design and evaluation of IPFS: a storage layer for the decentralized web. In: Proceedings of the ACM SIGCOMM 2022 Conference. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3544216.3544232
28. Vorick, D., Champine, L.: Sia: Simple Decentralized Storage, (2014). https://sia.tech/sia.pdf
29. Wilkinson, S., Boshevski, T., Brandoff, J., Prestwich, J., Hall, G., Gerbes, P., Hutchins, P., Pollard, C., Buterin, V.: Storj: A Peer-to-Peer Cloud Storage Network, (2016). https://storj.io/storjv2.pdf
30. Williams, S., Kedia, A., Berman, L., Campos-Groth, S.: Arweave: The Permanent Information Storage Protocol, (2023). https://arweave.org
31. Xu, J., Yang, A., Zhou, J., Wong, D.S.: Lightweight and Privacy-Preserving Delegatable Proofs of Storage, Cryptology ePrint Archive, Report 2014/395 (2014). https://eprint.iacr.org/2014/395.
32. Zahed Benisi, N., Aminian, M., Javadi, B.: Blockchain-based decentralized storage networks: A survey. Journal of Network and Computer Applications **162**, 102656 (2020). https://doi.org/10.1016/j.jnca.2020.102656
33. Zhang, C., Li, X., Au, M.H.: ePoSt: Practical and Client-Friendly Proof of Storage-Time. IEEE Transactions on Information Forensics and Security **18**, 1052–1063 (2023). https://doi.org/10.1109/TIFS.2022.3233780