Storchid

A New Open Source Initiative For Decentralized, Incentivized Data Storage

Avery, Chloe   Sheek, Justin

December 6, 2023

Contents

Abstract

Orchid is launching Storchid (pronounced store-kid), an open source decentralized data storage project. We believe that no third party should be in the middle of transactions between a client and a storage provider, and our design reflects this. Because our design is a truly decentralized marketplace, we minimize barriers to participation both for clients and providers while providing protection from bad actors through careful design that rewards honest participation and makes attacks costly.

With Storchid, clients are able to select a cohort of providers using the Orchid Directory mechanism, erasure code their data with Twin-coding, send data to providers, and perform payments. Through incentive alignment, clients are able to go offline and be assured that their data will be there when they come back. Providers advertise their storage capabilities by locking funds in a staking contract, prove that they are storing data correctly by making bonded commitments and performing cryptographically sound self-audits, and help maintain data availability by participating in repairs.

1 Overview

Entrusting your data to a single company entails the risk that it can be lost or compromised by that organization. Individual companies may disappear or become financially unstable, products go defunct (Google, notoriously, sunsets products all the time), and services that were free when you engaged them may suddenly force you to pay, or introduce intrusive advertising or API changes (viz. Reddit). Many major companies in recent memory have been the target of data breaches, or worse, have been found to be selling your data to affiliates. When using these products, you are trusting companies to handle your data appropriately, without recourse. The alternative proposed in this paper is *decentralized, incentivized data storage* that removes single points of failure and orchestrates independent Providers to securely store your data.

While some companies offer a degree of safety by storing data in multiple geographic locations, it is still subject to loss if the organization itself is compromised. Moreover, Clients are subjected to whatever price these companies decide to charge. Fees for data retrieval or migration can often be expensive, even prohibitively so, causing lock-in[1]. By contrast, decentralized data storage empowers the Client to price shop, is configurable in a multitude of ways, and provides an opportunity for smaller storage Providers at the network edge to profit from contributing to a robust data storage system without the need for massive data centers.

While there are existing products that claim to provide decentralized incentivized data storage, we believe there is significant room for improvement in the space, both in terms of more comprehensive decentralization and more robust incentive design.

With Orchid's storage design, you distribute your data into a self-monitoring self-healing system. That means you can go offline with the assurance that your data will still be there when you come back. All of this is done in a trustless, incentive-aligned, and configurable way. Achieving this goal requires innovative protocols with rigorous incentives that accommodate Provider failure, reward Providers who participate in repairs, and make your data self-maintaining and self-healing.

1.1 Design

Our design begins by asking the Client to select a cohort of Providers using the Orchid Directory mechanism. For the sake of efficiency, Client data is first erasure coded, rather than replicated. We utilize a technique called Twin-coding for further efficiency gains during repair operations. The particular choice of erasure coding scheme, however, is configurable by the Client. The Client distributes these coded Blocks to their selected cohort of Providers. To demonstrate that they are correctly storing these Blocks, the Providers perform self-audits, which are made possible through a carefully constructed protocol, making use of Bonded Commitments, Rate Certificates, and intricate incentive alignment. All of this can be done almost entirely off-chain in a way that makes storage cheap for the Client and profitable for the Providers.

[1]Amazon does this, making it difficult to leave AWS.

1.2 The Case for Decentralized Storage

Digital assets residing on blockchain networks and decentralized Web 3.0 applications (DApps) require significantly more data storage than available on blockchains, such as Ethereum, which have very limited and costly storage capabilities [3]. Decentralized storage networks offer a solution by providing storage for additional data, particularly metadata, while upholding decentralization advantages over centralized alternatives, ensuring resilience against single point of failures, guaranteed persistence, censorship resistance, verifiability, scalability, transferability, and community engagement.

A key application for metadata lies in the domain of non-fungible tokens (NFTs), representing various assets such as art, music, movies, tickets, and real-world assets (RWAs). The crucial metadata linked to these NFT assets, such as the image file required to view the actual art, requires secure storage. The decentralized approach ensures that metadata complements the asset without relying on a trusted third party. Currently, NFT metadata storage is often entrusted to centralized providers like Amazon Web Services (AWS). However, this poses challenges as NFT buyers can't be assured that metadata will persist. For example, the art associated with an NFT would disappear if an AWS customer were to close the account that stored the metadata the NFT pointed to. Although protocols like IPFS are addressing this problem by providing a universal locator, they don't guarantee metadata persistence. Decentralized storage networks can empower NFT owners by ensuring the persistent storage of metadata, with options for alterable and unalterable states as well as for proving authenticity or enabling transfer of control over the metadata to the ultimate owner of an NFT.

The potential of decentralized storage networks extends beyond NFTs to diverse Web 3.0 applications like gaming assets, metaverse property, legal documentation for real-world assets, and artificial intelligence (AI). They unlock use cases restrained by centralized solutions, such as secure and scalable infrastructure for open and collaborative AI development or decentralized social networks free from censorship. These networks play a pivotal role in achieving an increasingly decentralized Web 3.0 tech stack by addressing DApp challenges, where reliance on centralized providers like AWS poses accessibility risks. Orchid's open-source project uniquely aims to cater to these emerging use cases, including bandwidth-intensive scenarios, with a design that minimizes costs.

2 Concepts

- Actors: Client, Provider, Blockchain
- Erasure Coding
- Audits
- Bonded Commitment
- Rate Certificate

2.1 Actors

The Client decides what data to store, how to encode it into blocks, how to choose Providers, how to map those blocks onto those Providers, and how to persist and retrieve that data through payments to those Providers.

The Provider decides which Client blocks to accept, how to store those blocks, how much to charge for that service, when to send invoices, when to claim payments, and so on.

The Blockchain neutrally evaluates the contracts, payment terms, cryptographic security, commitment reveals, etc. in order to facilitate, secure, and otherwise finalize payments.

2.2 Erasure Coding

Erasure Coding is a mechanism for encoding source data into redundant blocks for durable (read: failure-resistant) storage. These erasure blocks are what a Provider is ultimately responsible for.

2.3 Audits

Audits are a proxy for data retrievability. They come in two flavors: *interactive* and *non-interactive.*

Interactive audits are performed by the Client, in direct communication with the Provider, in response to an invoice. A successful interactive audit precedes a nanopayment.

Non-interactive audits are performed by the Provider, absent the Client, by submitting a bonded commitment to the Blockchain. Correctness of these audits permits the Blockchain to release a payment from Client to Provider.

2.4 Bonded Commitment

A Bonded Commitment is a *binding commitment* (in the sense of commitment schemes) that is *bonded* (in the sense of debt security). This is a key component of the mechanism that enables a Provider to "self-audit."

2.5 Rate Certificate

A Rate Certificate is a credible commitment created and signed by the Client, held by the Provider, which entitles the Provider to payment according to the amount and subject to the terms described in the certificate. The certificate is presented, along with supporting evidence, to the Blockchain whenever a Provider wishes to claim payment.

3 Components

3.1 Provider Selection and Staking

Clients choose Providers from the available pool using the Orchid Directory mechanism, as described in the Orchid bandwidth whitepaper [7]. The Orchid Directory is an Ethereum smart contract that grants access to a trustless and permissionless marketplace. Clients use the smart contract to discover Providers via a stake-weighted random selection algorithm. Providers advertise their availability and soundness by locking funds in a staking contract, in turn receiving a proportional amount of new Client demand. Clients can further filter Providers using criteria in the form of curated lists or more advanced selection logic.

3.2 Payment Handling

3.2.1 Nanopayments

Probabilistic nanopayments, as implemented by Orchid, rely on provably fair stochastic interactions that eliminate the vast majority of on-chain data and allow for efficient exchange of arbitrarily small values with one or many parties.

Nanopayments pay for bandwidth when data is sent between a Client and Provider or between Providers. For example, nanopayments are sent whenever a Block is first uploaded from Client to Provider, when transferring Blocks between Providers during repairs, or when downloading a Block from Provider to Client to reconstruct a file.

For a detailed discussion of nanopayments see our previous work [7].

3.3 The Auditor Problem, or, Who Audits the Auditors?

Audits are a proxy for data retrievability. A successful audit serves as justification for a Provider to receive payment from the Client. Many decentralized storage projects rely on an actor, which we will generically call an "auditor", to perform this operation. However, reliance on an auditor adds a level of complexity to the incentive design that these projects largely ignore. For example, a Provider may conspire with or secretly pose as an auditor to receive favorable treatment or sow distrust among its competitors. This remains a problem even in the presence of a centralized trusted authority. In fact, a UK government report suggests that "11% to 15% of all reviews on e-commerce platforms for 3 common product categories (consumer electronics, home and kitchen, sports and outdoors) are likely fake" [5].

3.3.1 Eliminating the auditor

We eliminate the auditor by constructing a trustless on-chain mechanism for Providers to self-audit. The Providers are incentivized to perform these audits in order to collect payment, guaranteeing *correctness* through a **bonded commitment** to maintain Client data. That is, Providers periodically sequester on-chain some amount of capital, a bond, which they may only later recover by submitting proof of correctness. With these incentives and assurances, the Client can safely go offline, and the Provider can continue to receive payment in accordance with their Rate Certificate.

3.3.2 Why do self-audits work?

Given that blockchain transactions are expensive, it would be impractical to frequently submit a bonded commitment for each Client. Efficient commitment schemes such as KZG[2] commitments, a type of polynomial commitment, allow Providers to post one commitment for multiple pieces of data. Taken by itself, this approach has a couple of major problems. The first is that, unless the contents of the commitments are investigated, there remains an incentive for dishonest Providers to supply false reports. If those false reports are not discovered quickly, then they undermine the durability of Client data. Therefore, to each commitment, the Provider attaches a strong disincentive to making a false report: the bond. Furthermore, we require Provider commitments to be **irreversible**.

Definition 3.1. A commitment to a blockchain is **irreversible** if: 1) Under the security assumptions of the blockchain, the commitment itself cannot be reverted or modified. 2) Under the appropriate cryptographic assumptions, there is no method to construct an alternative history that leads to the same commitment.

Irreversibility ensures that the Provider can never recoup their bond whenever it accompanies a false report. For a Provider, this means the dominant strategy is to supply either correct or incomplete reports.

This leads to the second problem: incomplete reports. A Client may not be able to identify whether their block is referenced in a Provider's commitment. For example, there are infinitely many polynomials that share any given Kate commitment – knowledge of the commitment itself is insufficient to know its contents. If incomplete reports are not immediately identifiable, this undermines the durability of Client data. Therefore, we require Provider commitments to be **legible**.

Definition 3.2. A commitment is **legible** if its metadata is preemptively and efficiently identifiable.

Legibility ensures that an incomplete self-audit is immediately identifiable, thus preventing silent failures.

If we design a protocol in which commitments are both irreversible and legible, then, with appropriate incentives in place, it becomes irrational for a Provider to submit false or incomplete reports. That is, the dominant strategy for a Provider is to submit correct reports. These design constraints mirror those of *Proto-Danksharding* [2]. Since the design constraints are similar, the solutions are similar.

Legibility is achieved through the application of Proto-Danksharding. The bonded commitment uses paired KZG commitments that bind a set of block identifiers to a set of corresponding leaf identifiers. Because the Proto-Danksharding blob data contains the full KZG polynomials, that means anyone who knows the identity of a block can check whether a leaf of that block was committed to.

The cohort plays the role of verifier. Each member is responsible for retaining block identifiers and verifying, in part, the bonded commitments of the other members. They are incentivized to do so because we estimate prompt execution of the replacement protocol, selling their repair data, is likely the most profitable aspect of being a provider.

3.3.3 Proto-Danksharding

Proto-Danksharding is an essential step forward in Ethereum's efforts to scale globally. First, it enables native Ethereum Virtual Machine (EVM) support for KZG commitments. Second, it introduces a new ephemeral data type to blockchain transactions, called a blob, which is available to verifiers but opaque to the EVM. The former enables efficient checkpoints to "local state", i.e. state that is maintained off-chain or in a sidechain. The latter distinguishes an "execution framework" view from a "verification framework" view.

[2]ETH Research, Vitalik, and Dankrad present lengthy discussions regarding the merits and implementation of Kate/KZG, e.g. [6]

The collection of self-audits that a Provider performs constitute a "local state" that is regularly checkpointed to Ethereum. Similar to how in Proto-Danksharding these checkpoints are essential to proofs of correctness of shard state, irreversibility of these checkpoints is how our design enforces correctness.

The cohort comprise the verifiers that view the "verification framework". Similar to how in Danksharding this view is essential to Data Availability Sampling, legibility of this view is how our design enforces completeness.

3.4 File Upload, Download, and Maintenance

The Client selects a cohort of Providers for a given dataset such that each Provider stores a redundantly encoded piece of data, called a Block. The Providers in a given cohort participate in regenerating lost Blocks and ensuring that any new Providers meet the Client's selection criteria.

3.4.1 Erasure Coding

In a system where anyone who participates in the directory by staking can receive new Clients[3], some of your Provider will inevitably fail. Providers can be untrustworthy, but even honest Providers experience hardware failure or go offline unexpectedly. Some decentralized storage projects[4] choose to use a reputation system in order to address this problem. Reputation systems, however, are often manipulated. It's common practice on retail platforms, like Amazon or Ebay, for sellers to provide outside incentives for good reviews, or to create fake Client profiles to leave good reviews on their products. This undermining of incentives makes reputation a poor proxy for trustworthiness.

Instead, Orchid's protocol allows for Provider failure and optimizes the speed and cost of repair. Providers are incentivized to act according to the protocol, as discussed in Section 3.3. Other storage projects[5] address Provider failure by relying on multiple Providers who each hold full copies of the data. This strategy is called **replication**. Replication helps protect against certain kinds of failures, but it is not efficient. A superior strategy to replication is **erasure coding**. Erasure coding is a way for the Client to give each Provider a piece of the data so that if the Client receives pieces from *any* sufficiently large subset of the Providers, the Client is able to reconstruct the data in full.

For example, in a classic Reed-Solomon erasure coding scheme, parameters n and k are chosen, with $k \leq n$, and a source file is redundantly encoded into n overlapping pieces, called *erasure blocks*, such that *any* k of these blocks can be used to reconstruct the source file. Different choices of k and n offer varying degrees of durability. Allowing the Client to choose these parameters empowers them to fine-tune the cost of storage versus their desired level of durability.

3.4.2 Twin-coding

Erasure coding schemes provide highly efficient data retrieval and durability, but tend to have one major bottleneck: namely, the cost to replace a missing block. In a typical erasure coding scheme, replacing a single missing Block requires the *entire* dataset to be reconstructed. Then the missing Block is regenerated and the rest of the data is thrown away. This is a massive waste of bandwidth, which we estimate to be the largest cost to the Client in a decentralized system. As such, although the choice of erasure coding scheme is left to the Client, we strongly recommend, and enable by default, a **Twin-coding** scheme. This erasure coding scheme breaks the cohort of Providers into two groups, called Type 0 and Type 1. Then, the data is encoded using two linear, but potentially different, erasure coding schemes. The Client is able to choose these two schemes as well as the parameters (e.g. n and k for a Reed-Solomon code) for each. In a typical repair, if a Type 0 Provider fails, then it is repaired by Type 1 Providers, and if a Type 1 Provider fails, then it is repaired by a Type 0 Provider[6]. The biggest advantage to this scheme is that to repair a Block, only precisely the number of bits in that Block need to be sent, dramatically reducing bandwidth costs.

[3]Providers are randomly selected in proportion to their stake. See 3.1 for more information on this.

[4]Such as Storj

[5]such as Filecoin

[6]It is, however, still possible for Providers of a given type to indirectly repair a Provider of the same type

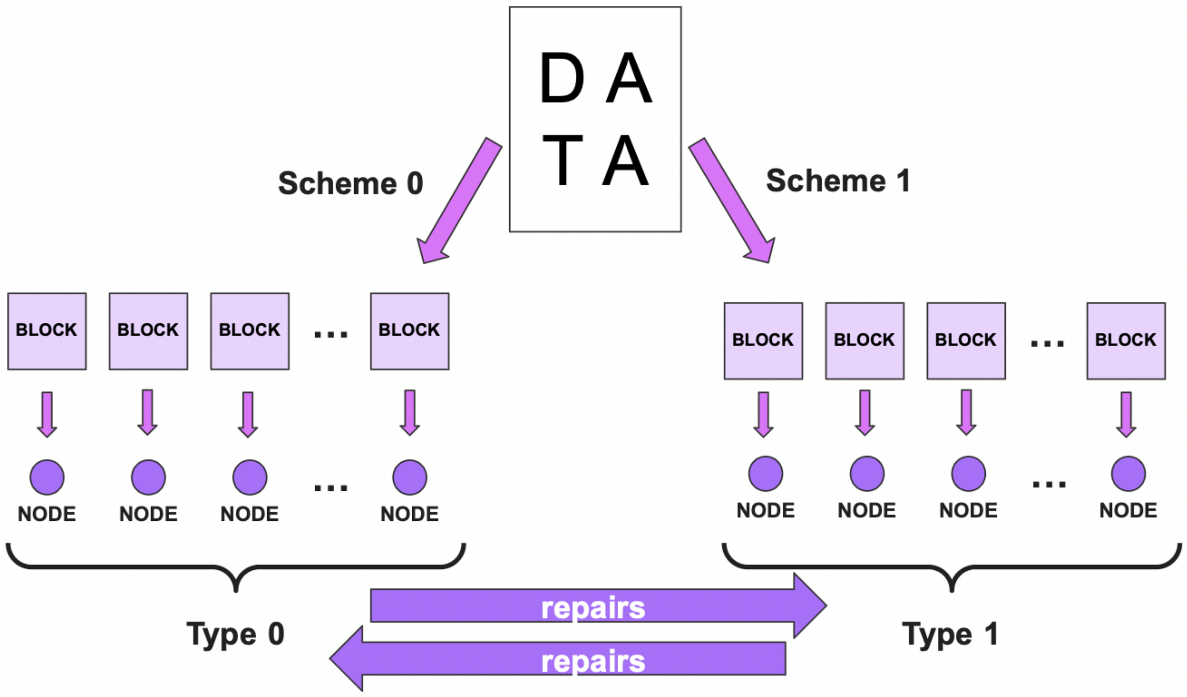

Figure 1: Type 0 nodes repair Type 1 nodes and Type 1 nodes repair Type 0 nodes

3.4.3 Repairs

If a Provider in a cohort stops responding to audits or posts a faulty or incomplete audit, a repair process is initiated. The cohort is incentivized to cooperate to construct a new Rate Certificate, select a new Provider according to rules defined by the Client, and help the new Provider restore the missing Block, along with any necessary metadata.

The Client is either online or offline when a repair needs to happen. When the Client is online, the repair protocol is as follows:

1. A Provider makes a commitment that is missing the Client's data.
2. The Client notices the missing subcommitment.
3. The Client hires a new Provider: the repair Target.
4. The Client orchestrates the cohort of Providers to send data to the Target. The participating members of the cohort profit from the bandwidth.
5. After a successful repair, the Target notifies the Blockchain.
6. The bond posted by the failed Provider is partially destroyed and the rest is awarded to the Target.

When the Client is offline, the repair protocol is as follows:

1. A Provider makes a commitment that is missing the Client's data.
2. Other Providers in the cohort notice the missing subcommitment.
3. Providers in the cohort select a new Provider, the repair Target, on behalf of the Client. They use a message which is authenticated by the Client to prove to the Target that Provider selection was done correctly.

4. These Providers also reconstruct the Rate Certificate and send it to the Target, allowing the Target to potentially refuse on the basis of price.
5. The cohort of Providers send data to the Target, profiting from the bandwidth.
6. After a successful repair, the Target notifies the Blockchain.
7. The bond posted by the failed Provider is partially destroyed and the rest is awarded to the Target.

In this protocol, cohort members profit from being among the first to participate in the repair by contacting the repair Target to sell their contributions to the missing block. With this, the Client can be assured that when they are offline, the cohort is self-healing and furthermore that repairs are incentivized to happen quickly. When the Client is offline, the repair Target is chosen according to the Client's selection algorithm. Any provider who attempts to select an alternative Target will only be successful if the Target is dishonest and a majority of cohort members are also in collaboration with the same dishonest Target. As a further protection, the repair Target may later be required to demonstrate the provenance of their Rate Certificate to show that they were correctly selected.

4 Comparison To Other Projects

At the time of writing, there are several competing decentralized storage projects. What makes Orchid's design different from the storage projects that are already out there?

4.1 Decentralization

Many projects promise decentralization while retaining centralizing components. For example, the design of Storj relies on satellite nodes, which are responsible for storage maintenance tasks such as holding metadata, performing audits and repairs, and maintaining storage node reputation. To date, these satellite nodes are exclusively operated by Storj, who also determines pricing.

4.2 Erasure coding

Erasure coding is both more durable and more efficient than data replication. We choose to use erasure coding, which we discuss in Section 3.4.1. Arweave and Filecoin utilize replication. Filecoin Providers use Proof-of-Replication to prove that data has been replicated and is being stored in unique locations. Whereas Storj uses erasure coding, they exhibit costly repairs. In order to amortize these costs, Storj undermines data durability by waiting until there are several block failures before repairing. Orchid's design uses lightweight audits to circumvent the need for cumbersome Proof-of-Replication and uses Twin-coding to circumvent deliberate accumulation of failures and costly repairs.

4.3 Bandwidth Costs

In a decentralized storage project, we estimate the bandwidth needed for repairs to be the dominant cost. To cope with this, some decentralized storage projects set uptime or quality of service standards, to inhibit the need for repairs. However, in practice, we find these standards to be both unenforceable and centralizing. Other projects, such as Storj, mitigate these costs by over-allocating Providers, then deliberately accumulating failures before initiating a repair. Our solution is to use Twin-coding, as outlined in Section 3.4.1, which requires less bandwidth to perform repairs, allowing them to be undertaken more aggressively without additional accumulation of risk.

4.4 Reputation Systems vs Stake-Weighted Random Selection

Reputation systems are subject to manipulation and thus not a reliable metric. Some decentralized storage projects use reputation as a means of evaluating nodes and deciding how to connect Clients to Providers. Storj, for example, uses satellite nodes to keep track of reputation.

Instead of a reputation system, we use stake-weighted random selection, as discussed in Section 3.1.

4.5 Provider Requirements

Below are some of the requirements to become a storage Provider for other decentralized storage systems:

Filecoin storage Providers must stake, at the time of writing, 6.4 filecoin tokens per 1TB of storage provided. However, the hardware required for a full deployment to meet the specs required by the reference architecture[7] is financially costly enough to present a form of centralization itself.[?price estimate $200k?]

To become a Storj node operator, the recommended hardware is 8TB of disk capacity, 16+TB of symmetric 100Mbps bandwidth per month, and a maximum downtime of 5hrs/month.

Our design has no minimum storage, bandwidth, uptime, or token holding requirements to participate. Providers receive new Clients in proportion to the amount they stake.

4.6 Auditing

Filecoin uses Proof-of-Spacetime to audit storage providers. Proof-of-Spacetime ensures that a storage provider has been storing data for a specified amount of time. This is unnecessarily computationally heavy as intermittent audits are sufficient to show that data is available when the client needs it.

Other storage projects, such as Storj, have an actor who audits the storage Providers to ensure that they are acting in the Client's interest. This merely relocates the need for the Client to trust the Provider to instead need to trust the Auditor. Even a central authority can't reliably prevent Auditors from conspiring with Providers. This is why Orchid has opted for Providers to self-audit, which is outlined in Section 3.3.

5 Conclusion

Orchid's design for a marketplace for decentralized data storage gives control and autonomy to users. It enables providers to supply storage with very few limitations and empowers clients to configure nearly all aspects of their data storage and be able to go offline with the assurance that their data will be there when they come back. Orchid's design epitomizes decentralization, guaranteeing that interactions occur without third-party mediation. Because of the careful use of incentives, clients and providers alike can be assured that they will be protected from those with malicious intent.

6 Acknowledgements

The authors would like to thank Dan Montgomery, Patrick Dietzen, Patrick Niemeyer, and Travis Cannell for their invaluable feedback, advice, and help in preparing this work. The authors also thank Dr. Steven Waterhouse for his support throughout this process.

References

[1] Arweave: A Protocol for Economically Sustainable Information Permanence. `https://www.arweave.org/yellow-paper.pdf`.

[2] Danksharding. `https://ethereum.org/en/roadmap/danksharding/`.

[3] Decentralized storage: A pillar of web3. `https://6pjecoitbb3mbacc67rvmct3gbugplnty5ptok4t6tkgvxnoj2ya.arweave.net/89JBORMIdsCAQvfjVgp7MGhnrbPHXzcrk_TUat2uTrA`.

[4] Filecoin: A Decentralized Storage Network. `https://filecoin.io/filecoin.pdf`.

[5] Investigating the prevalence and impact of fake reviews. `https://www.gov.uk/government/publications/investigating-the-prevalence-and-impact-of-fake-reviews`.

[6] KZG Polynomial Commitments. `https://dankradfeist.de/ethereum/2020/06/16/kate-polynomial-commitments.html`.

[7] Filecoin

[7] Orchid: A Decentralized Network Routing Market. `https://www.orchid.com/whitepaper/english.pdf`.

[8] Storj: A Decentralized Cloud Storage Network Framework. `https://www.storj.io/storjv3.pdf`.

[9] Threshold Cryptography, MPC, and MultiSigs: A Complete Overview. `https://blog.pantherprotocol.io/threshold-cryptography-an-overview/`.

[10] KV Rashmi, Nihar B Shah, and P Vijay Kumar. Enabling node repair in any erasure code for distributed storage. In *2011 IEEE international symposium on information theory proceedings*, pages 1235–1239. IEEE, 2011.